KRATAK UVOD U ANALIZU ALGORITAMA

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
DEPARTMAN ZA MATEMATIKU I INFORMATIKU

Igor Dolinka

KRATAK UVOD U ANALIZU ALGORITAMA

NOVI SAD, 2008.

Naslov: KRATAK UVOD U ANALIZU ALGORITAMA

Autor: DR IGOR DOLINKA
redovni profesor PMF-a u Novom Sadu
dockie@im.ns.ac.yu

Recenzenti: DR SINIŠA CRVENKOVIĆ
redovni profesor PMF-a u Novom Sadu

DR DRAGAN MAŠULOVIĆ
vanredni profesor PMF-a u Novom Sadu

Izdavač: PRIRODNO-MATEMATIČKI FAKULTET U NOVOM SADU

Za izdavača: PROF. DR MIROSLAV VESKOVIĆ, dekan Fakulteta

Slog: AUTOR

Štampa: FUTURA, Novi Sad

Tiraž: 200 primeraka

Štampanje i upotrebu ovog udžbenika odobrilo je Nastavno-naučno veće Prirodno-matematičkog fakulteta na svojoj sednici od 14.12.2007.

CIP – Каталогизација у публикацији
Библиотека Матице српске, Нови Сад

510.51(075.8)

ДОЛИНКА, Игор
Kratak uvod u analizu algoritama / Igor Dolinka. — Novi Sad : Prirodno-matematički fakultet, Departman za matematiku i informatiku, 2008 (Novi Sad : Futura). — xii, 157 str. : ilustr. ; 24 cm.

Tiraž 200. — Bibliografija. — Registar.

ISBN: 978-86-7031-165-7

a) Теорија алгоритама

COBISS.SR-ID 232484615

Onaj koji isključivo ceni praksu bez teorijskih osnova
sličan je moreplovcu koji ulazi u brod bez krme i busole
ne znajući kuda se plovi.

LEONARDO DA VINČI

Predgovor

Ova knjiga je napisana sa namerom da pruži uvid u osnovne elemente matematičke teorije algoritama i njihove složenosti. Kao takva, ona nalazi svoje mesto na (ne preterano oštroj) granici između matematike i računarstva (eng. *computer science*). Teorija izračunljivosti i teorija računske složenosti — dve discipline teorijske informatike koje ćemo izučavati na narednim stranama — predstavljaju veoma značajne etape u obrazovanju svakog profesionalnog računarca, ali i matematičara.

Na fakultetu na kome predajem (a prema najnovijoj verziji studijskih programa u trenutku pisanja ovog teksta), mešavina ove dve teorije poprima dve inkarnacije: prva je kurs ***I152: Analiza algoritama*** na III godini osnovnih studija informatike, dok druga pod imenom ***Teorija algoritama*** figuriše kao izborni predmet na doktorskim studijama iz matematike. Moja je nada da će ova knjiga naći svoju primenu u odnosu na oba predmeta — naravno, u različitim ulogama. Ona je prvenstveno koncipirana kao osnovni udžbenik za prvi od dva navedena kursa. Kada je u pitanju drugi predmet, ona treba da posluži kao polazna osnova za dalje širenje i produbljivanje znanja. To širenje može krenuti u većem broju pravaca: ka dubljem izučavanju teorije algoritama kao grane matematičke logike (pre svega kroz detaljnije ispitivanje fenomena rekurzije), ka naprednijim konceptima dizajna i analize algoritama (kao što su to paralelni, slučajni i aproksimativni algoritmi, kriptografija, itd.), ili, na primer, ka istraživanjima u vezi algoritamskih problema u algebri.

S druge strane, treba naglasiti da je misija teorije algoritama i njihove složenosti u sklopu studija informatike značajno drugačija od one koju ima predmet

Strukture podataka i algoritmi. Upoznavanje sa bazičnim fondom struktura podataka, algoritamskih ideja i tehnika, predstavlja na određeni način "temelj" programerske tehnologije, skup osnovnih alatki u dizajnu softvera. Međutim, teorija algoritama predstavlja nastavak i nadgradnju ovih znanja: njen cilj je da sa aspekta matematičke nauke kao vrhunskog analitičko-interpretativnog sredstva osvetli samu suštinu pojma algoritma, te da usvojene tehnike uklopi i ujedini u jednu širu sliku informatike kao racionalne i sistematske (a ne *ad hoc*) ljudske delatnosti. Moje je najdublje uverenje da — čak i kada matematički koncepti i metode nisu eksplicitno prisutni u razvoju informacionih tehnologija — upravo *matematički stil razmišljanja* jeste onaj ključni element koji vodi osmišljenom rešavanju ogromne većine relevantnih problema u toj oblasti. Matematička konceptualzacija algoritma, jednog od ključnih informatičkih pojmova, predstavlja dobar deo "krme i busole" kojima možemo da se pouzdano rukovodimo u traganju za naprednijim i kvalitetnijim IT rešenjima.

Imajući sve ovo u vidu, želeo sam da napišem knjigu koja je u izvesnom smislu "minimalistička". Većina najpoznatijih referenci iz oblasti teorije algoritama, odnosno teorije računske složenosti (od kojih se jedan deo nalazi u literaturi), jesu znatno obimnija štiva. Ovaj tekst je, međutim, nastao iz skripti sa predavanja iz *Analize algoritama* (držanih na smerovima primenjene matematike), koje sam negde od jeseni 2005. naovamo postepeno "brusio" i usavršavao. Zbog toga on obuhvata tek malo više od materijala koji se (prema ovdašnjim metodama nastave) može obraditi za jedan semestar: "viškovi" (npr. dokaz rekurzivnosti Akermanove funkcije, rekurzivne nabrojivosti rekurzivnih skupova, korektnosti Dajkstrinog algoritma, Kuk-Levinove teoreme, $\mathbf{NP}$-kompletnosti problema Hamiltonovih kontura i sl.) predstavljaju fakultativnu domaću lektiru za ambicioznije studente. Istovremeno, hteo sam da maksimalno istaknem najznačajnije motive i ideje koji čine okosnicu kursa, što ne bi bilo moguće da sam udžbenik opteretio dodatnim gradivom koje se neposredno ne uklapa u centralni tok zapleta priče. Najzad, morao sam i da opravdam naslov knjige: ne bi valjalo da je ona premašila obim malo duže novele.

Matematičke pretpostavke za savladavanje ovog kursa nisu prevelike i one su uobičajene za ovaj tip materije. Uglavnom, radi se o elementima matematičke logike (osnovi matematičke pismenosti, iskazna i predikatska logika, prirodni brojevi, matematička indukcija, naivna teorija skupova, relacije i funkcije), kao i o jednostavnijim konceptima i rezultatima diskretne matematike (reči, skupovi i multiskupovi, particije i druge osnovne kombinatorne konfiguracije, rekurentne relacije, grafovi i digrafovi). Poželjno je (iako ne i neophodno) neko iskustvo sa polaznim kursevima programiranja.

Veliku zahvalnost dugujem mom učitelju, profesoru Siniši Crvenkoviću. Sarađujući sa njim dugi niz godina i crpeći iz njegovog iskustva (a — budimo pošteni — i izvanrednih beleški) sam napredovao (i) kao nastavnik ovog predmeta. Takođe, razgovori koje sam na temu algoritama vodio sa kolegama prof. dr Rozáliom Sz. Madarász, prof. dr Đurom Paunićem, dr Draganom Mašulovićem i dr Petrom Markovićem su doprineli ne samo kristalizaciji kursa, nego i boljem viđenju celokupne oblasti. Među već pomenutim imenima nalaze se i recenzenti, pa koristim priliku da im se i ovim putem zahvalim na korisnim sugestijama i pruženoj podršci.

Na kraju, ali pre svih, zahvalnost upućujem mojoj supruzi Szabó Teréz, koja mi je tokom čitavog rada na ovom naslovu, uz njeno strpljenje i razumevanje, predstavljala nepresušan izvor inspiracije.

NOVI SAD, 22.9.2007. *Igor Dolinka*

Sadržaj

1

Pojam algoritma

1.1 Malo istorije: poreklo reči *algoritam*

Reč *algoritam* (eng. *algorithm*[1]) je latinizovani oblik prezimena jednog čoveka. U pitanju je *Abu Džafar Muhamed ibn-Musa* ***al-Horezmi*** (≈780–850), astronom, astrolog, geograf i jedan od prvih značajnih srednjevekovnih arapskih[2] matematičara. Rođen je u Horezmu (danas Hiva, Uzbekistan), u tadašnjoj persijskoj provinciji Horasan, a najveći deo svog života proveo je radeći kao naučnik u Bagdadu u tzv. *Kući mudrosti*[3]. U naslovu njegovog najpoznatijeg i najznačajnijeg dela, *Al-kitab al-muhtasar fi hisab al-džabr val-mukabala* ("Sažeta knjiga o računanju putem dopunjavanja i uravnoteženja", u latinskom prevodu: *Liber algebræ et almucabalæ*) krije se koren još jednog osnovnog termina savremene matematike: *algebre*. Zaista, al-Horezmijeva "Sažeta knjiga", napisana oko 820. godine, predstavlja zbir i sistematizaciju tada poznatih veština u rešavanju algebarskih jednačina. Pod time, naravno, podrazumevamo kvadratne

[1] U XIX veku i početkom XX veka u upotrebi je bio i termin *algorism*.

[2] Al-Horezmi je poreklom bio Persijanac, ali celokupna njegova delatnost predstavlja doprinos razvoju arapske kulture.

[3] Kuća mudrosti, *bejt al-hikma*, velika dvorska biblioteka, osnovana je u drugoj polovini VIII veka, kada je prestonica Arapskog carstva premeštena iz Damaska u Bagdad dolaskom dinastije Abasida na vlast 750. godine. Najplodnije godine rada al-Horezmija vezuju se za vladavinu *al-Mamuna* (813–833), sedmog kalifa ove dinastije, koji je po majci bio persijskog porekla, i koji je do dolaska na presto bio guverner Horasana.

jednačine, koje al-Horezmi deli na šest osnovnih tipova, do kojih dolazi primenama pravila *al-džabr* (sabirak može da pređe na drugu stranu jednačine uz promenu znaka) i *al-mukabala* (potiranje istih sabiraka na obe strane jednačine). Proći će čitavih sedam vekova pre nego što će italijanska renesansna matematika učiniti korak dalje otkrićem postupaka za rešavanje jednačina trećeg i četvrtog stepena.

Međutim, za nastanak reči "algoritam" od ključne važnosti je drugo najznačajnije delo al-Horezmija. Arapski original ovog spisa je izgubljen i ono je sačuvano samo u latinskom prevodu iz prve polovine XII veka. Čak je i taj latinski prevod nekompletan (nedostaje nekoliko prvih stranica, pa stoga naslov dela nije pouzdano uvrđen), a tekst na prvoj sačuvanoj stranici počinje rečima: *Dixit Algorismi:...* ("Kaže al-Horezmi:..."). Kasnije, knjizi je dat latinski naslov *Algorismi de numero Indorum* ("Al-Horezmi o indijskoj veštini računanja"), a pretpostavlja se (na osnovu drugih izvora) da je izvorni naslov glasio *Al kitab al-džam val-tafrik bi hisab al-Hind* ("Knjiga o sabiranju i oduzimanju u indijskom računu").

Otkud ta "indijska veština računanja" u bagdadskoj Kući mudrosti? Naime, VII vek i prva polovina VIII veka predstavljali su vreme pojave i brzog širenja nove religije na svetskoj mapi: *islama*. Osnivač islama, prorok Muhamed je do smrti 632. godine uspeo da ujedini celo arapsko poluostrvo, na kome su do tada živela više-manje nepovezana nomadska plemena. Muhamedovi naslednici, *kalifi*[4], osvajanjima su znatno proširili teritoriju arapske države, koja se na vrhuncu moći prostirala od Atlanskog okeana, pa sve do reke Ind. 637. godine, Arapi su osvojili Persiju, zbacivši poslednjeg vladara sasanidske dinastije. Ovaj događaj je značajan kako sa stanovišta ogromnog uticaja drevne persijske civilizacije i njenih tekovina koje su Arapi asimilovali, tako i u pogledu kontakta sa dostignućima indijske kulture. Jedno od za nas najznačajnijih takvih dostignuća bio je pozicioni brojevni sistem sa deset simbola:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9.$$

Ove simbole su Arapi nazvali *indijske cifre*. Kasnije, zaslugom upravo al-Horezmija, indijski decimalni sistem računanja je ušao u praktičnu upotrebu u celom arapskom svetu.

[4] Većinska denominacija unutar islama, *suniti*, smatraju kalife za legitimne naslednike Muhameda, kao istovremene nosioce duhovne i svetovne vlasti u tadašnjem islamskom svetu. Međutim, *šiiti* smatraju da duhovno vođstvo i pravo na tumačenje islamskog učenja pripada isključivo Muhamedovim potomcima, ćerki Fatimi Zahri i zetu Aliju (koji je, inače, bio četvrti kalif), dok su kalifi (sa izuzetkom Alija) priznati samo kao svetovni vladari.

S druge strane, Arapi su 711. godine preko Gibraltara upali u Evropu, da bi ih tek 732. porazio franački vojskovođa Karlo Martel (deda Karla Velikog) u bici kod Poatjea. Njihovo napredovanje je zaustavljeno, ali će Arapi (u Evropi tada poznati pod imenom Mavri) ostati na Iberijskom poluostrvu narednih sedam i po vekova[5]. Kao politički i kulturni centar mavarske Španije nametnuo se grad Kordoba, u kome su delovali mnogi učeni ljudi tog vremena. Čak i kasnije, tokom opadanja mavarske moći (ključni događaj u tom smislu je osvajanje Toleda od strane hrišćana 1085. godine), Kordoba i Toledo su bili centri u kojima se odvijalo prevođenje velikog broja arapskih rukopisa na latinski. Upravo na taj način je nastao i prevod *Algorismi de numero Indorum*, a istim putem je za današnja pokolenja sačuvana (posredstvom arapskih prevoda sačinjenih u Kući mudrosti) većina dela Aristotela, Euklida, Ptolemeja i čitavog niza drugih antičkih filozofa i matematičara, kao i drugih filozofskih i naučnih spisa starogrčkog, arapskog i indijskog porekla. Tako je početkom XII veka Evropa po prvi put došla u dodir sa indijskim ciframa, koje su (iz sada sasvim razumljivih razloga) Evropljani nazvali *arapske cifre*. Arapski decimalni sistem je bio daleko praktičniji i lakši za upotrebu od nezgrapnih rimskih brojeva. Stoga će sa renesansom i reformacijom, kao i ubrzanim ekonomskim razvojem Evrope koji će uslediti — prvenstveno u vidu rapidnog jačanja trgovine i prve pojave manufakturnog kapitalizma — arapski brojevi u praksi potpuno istisnuti rimske, koji će zadržati samo protokolarnu, ali ne i upotrebnu vrednost.

Tako su, istorijski gledano, udareni temelji *pojma algoritma*, koji je prvobitno označavao veštinu računanja pomoću arapskih (indijskih) cifara.

1.2 Intuitivni pojam algoritma

Svrha svakog algoritma jeste rešavanje određenog *problema*. Svaki problem odlikuju dve karakteristike: *ulazni* i *izlazni podaci*. Ovi podaci jesu matematički objekti koji pripadaju nekom konačnom ili prebrojivom skupu, dok se sam problem sastoji u iznalaženju određene uzročno-posledične veze između ulaza i izlaza, metodologije kojom se ulazne veličine transformišu u željeni (matematički definisan) rezultat. Zbog toga je najpogodnije da intuitivni pojam problema matematički formalizujemo kroz koncept diskretne funkcije (funkcije čiji su domen i kodomen najviše prebrojivi).

[5] 755. godine, pet godina nakon svrgavanja sa vlasti u Arapskom carstvu, dinastija Omajada će osnovati nezavisni kalifat sa prestonicom u Kordobi. Tek će 1492. Ferdinand Aragonski osvojiti Kraljevinu Granadu, poslednju islamsku državu na tlu Španije.

Sa stanovišta informatike i ljudske prakse uopšte, najinteresantniji su oni problemi čije se rešavanje (transformacija ulaza u izlaz) odvija na automatizovan, "jednoobrazan", rutinski način. Drugim rečima, posmatramo one probleme koji se mogu rešiti upotrebom *algoritama*, tj. primenom standardizovanih postupaka računanja, što u krajnjoj liniji omogućava da se posao rešavanja problema prepusti računskim mašinama. Na nivou diskretnih funkcija kao apstraktnog modela problema, to znači da tragamo za klasom funkcija koje po svojim svojstvima odgovaraju našim (nematematičkim) intuitivnim kriterijumima algoritamske izračunljivosti. Te funkcije najčešće označavamo terminom *izračunljive funkcije* (eng. *computable functions*).

Naravno, neizostavno se nameće pitanje o matematičkom konceptu (tj. matematički definisanoj potklasi diskretnih funkcija) koji treba opredeliti intuitivnom pojmu izračunaljive funkcije, tako da odabrani koncept najbolje odgovara našim zamislima i potebama. Bitno je naglasiti da je svaka odluka ili dogovor po tom pitanju čin koji po svom karakteru leži van okvira matematike, jer se radi upravo o tome da se neki matematički formalizovan objekat *po konvenciji* izjednači sa pojmom algoritma — dakle, idejom koja u suštini pripada domenu ljudske psihologije. Zbog toga je pitanje matematičke konceptualizacije pojma izračunljivosti mnogo više upućeno filozofiji nauke, nego samoj nauci.

Donald E. Knuth (Donald Knut, 1938) u uvodu svoje izuzetno uticajne monografije *The Art of Computer Programming* [20], ističe pet ključnih svojstava intuitivnog pojma algoritma:

- ulaz,
- izlaz,
- konačnost,
- definitnost (određenost),
- efektivnost.

Prve dve osobine — postojanje uzlaza i izlaza — govore zapravo o tome da su algoritmi sredstvo za rešavanje problema. Na primer, u Euklidovom algoritmu ulaz se sastoji od dva pozitivna cela broja, dok je izlaz njihov najveći zajednički delilac. S druge strane, u algoritmima za pretraživanje grafova ulaz se sastoji iz konačnog (orijentisanog ili neorijentisanog) grafa $\mathcal{G}$ i dva čvora $s, t \in V(\mathcal{G})$, dok je izlaz binaran (DA ili NE) u zavisnosti od toga da li postoji put $s \rightsquigarrow t$ u grafu $\mathcal{G}$.

U načelu, od algoritma očekujemo da bude niz intuitivno jasnih koraka. Zahtev *konačnosti* postulira da broj tih elemenatrnih koraka mora biti konačan. U tome se algoritam razlikuje od procesa: proces može biti jasno definisan, ali beskonačan niz koraka. Na primer, iterativni postupak za rešavanje neke jednačine (kao što je to Njutnov postupak) nije algoritam, već je u pitanju (beskonačan) proces računanja. Međutim, definisanjem margine greške, on postaje (numerički) algoritam za približno rešavanje posmatrane jednačine, jer konvergencija postupka garantuje da će se nakon konačno mnogo iteracija doći do zadovoljavajućeg približnog rešenja.

Definitnost izražava potrebu da svi pomenuti koraci algoritma budu precizno i nedvosmisleno formulisani, te da su lišeni svih oblika subjektivnosti. Između ostalog, definitnost jeste jedna od glavnih razlika između algoritma i (kulinarskog) recepta. Očekujemo da koraci algoritma budu do te mere egzaktno određeni da se oni bez ikakvih daljih interpretacija mogu sprovesti na računaru. Programski jezici predstavljaju tipičan primer sredstva kojim se ostvaruje zahtev za definitnošću.

Najzad, očekujemo da algoritmi budu *efektivni*: koraci koji čine algoritam moraju biti dovoljno *elementarni*, a ceo tok algoritma (bar u načelu) lako proverljiv od strane čoveka.

1.3 Reči i jezici, formalizacija problema

Pre nego što se opredelimo za matematički model izračunljivih funkcija, iz tehničkih razloga je zgodno da najpre malo preciziramo reprezentaciju samih problema.

Najpre, svaki matematički objekat — pa tako i ulazni i izlazni podaci nekog problema — reprezentuje se kao konačan niz simbola. Svi simboli koji stoje na raspolaganju za zapisivanje određenog tipa objekata čine *azbuku*. S obzirom na zahteve koje smo postavili u vezi sa intuitivnim pojmom algoritma, sve azbuke koje razmatramo moraju biti konačne ili prebrojive.

Polazeći od simbola azbuke Σ, možemo graditi konačne nizove elemenata Σ, koje zovemo *reči*. Pri tome je uobičajeno da se pri zapisivanju reči ne koriste nikakvi dodatni simboli za razdvajanje elemenata niza, tako da se u slučaju $\Sigma = \{a, b\}$ jedna od reči nad Σ kraće zapisuje kao $abbababa$ umesto (a, b, b, a, b, a, b, a). Skup svih reči nad Σ označava se sa Σ^*, što uključuje i *praznu reč* λ (niz dužine 0).

Jezik L (nad Σ) je proizvoljan skup reči, $L \subseteq \Sigma^*$.

Prema tome, ako je Σ azbuka kojom je zapisan neki problem, tada ulazne i izlazne podatke možemo posmatrati kao elemente skupa Σ^*. Naravno, ne mora pri tome svaka reč iz Σ^* predstavljati smislen zapis ulaznog podatka našeg problema. Na primer, ako posmatramo problem u kome je ulazni podatak iskazna formula ϕ, a izlaz npr. binarni zapis broja različitih valuacija za koje ϕ ima vrednost $\top$, tada kao azbuku ovog problema možemo definisati $\Sigma = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,), x_1, x_2, x_3, \ldots, 0, 1\}$. Jasno, $\phi = \neg x_1 \vee (x_2 \Rightarrow x_3)$ jeste korektno izgrađena iskazna formula nad datom azbukom i pri tome je $f(\phi) = 111$ (gde je f funkcija koja modelira dati problem), dok $x_{15})(\Leftrightarrow x_6 \neg 0$ i $1 \wedge \vee \Rightarrow) x_{26}$ jesu reči nad Σ, ali ne predstavljaju zapise iskaznih formula.

Bez velikog umanjenja opštosti, možemo naše posmatranje problema svesti na slučaj kada svaki problem ima *jedan* izlaz, tj. kada izlazne podatke tretiramo kao jedinstven objekat. Ako je Σ azbuka uočenog problema, a n broj ulaznih podataka, tada taj problem možemo poistovetiti sa nekom parcijalnom funkcijom

$$f : (\Sigma^*)^n \to \Sigma^*.$$

Oblast definisanosti funkcije f je *domen problema*, a elementi domena su *instance*[6]. U prethodnom primeru smo imali $n = 1$, funkcija f je bila definisana samo za reči koje jesu zapisi iskaznih formula, a kodomen smo bez suštinskih izmena mogli suziti i na skup $\{0, 1\}^*$. Ta funkcija je očigledno izračunljiva, jer ako pođemo od formule ϕ, sastavimo njenu istinitosnu tablicu, prebrojimo broj simbola $\top$ u poslednjoj koloni i zapišemo taj broj u binarnom sistemu, dobijamo $f(\phi)$. S druge strane, za sve nabrojane potprobleme postoje dobro poznati algoritmi koji ih rešavaju.

Posebnu klasu čine problemi kod kojih je izlazni podatak binaran (DA/NE), tj. kod kojih se pitamo da li objekat (ili objekti) predstavljeni ulaznim podatkom imaju ili nemaju neko svojstvo. To su tzv. *problemi odlučivanja*. Njihovo matematičko modeliranje se dodatno pojednostavljuje. Naime, primetimo da je problem odlučivanja u potpunosti određen sa dva jezika: domenom problema $\Gamma \subseteq \Sigma^*$ i skupom $L \subseteq \Gamma$ svih instanci koje rezultuju izlazom DA. Stoga problem odlučivanja i pišemo kao par (Γ, L). U slučaju $\Gamma = \Sigma^*$ dobijamo upravo problem pripadnosti (eng. *membership problem*) za jezik L: za datu reč $w \in \Sigma^*$ pitamo se da li $w \in L$. Tako se opšti problem odlučivanja svodi na dva problema pripadnosti jezika: najpre za jezik Γ, putem kojeg se utvrđuje da li je data reč uopšte instanca posmatranog problema (npr. da li je u pitanju korektno konstru-

[6] Uvođenje parcijalnih funkcija je neophodno upravo iz razloga opisanih u prethodnom pasusu: nije neophodno da svaki niz reči nad Σ predstavlja instancu za razmatrani problem.

isana iskazna formula), a zatim za L, putem kojeg se dobija rešenje problema za reči koje predstavljaju instance.

1.4 Reči i prirodni brojevi

Pored prikazanog "lingvističkog" pristupa formalizaciji pojma problema koji je baziran na konceptima iz teorije formalnih jezika, postoji i drugi, "aritmetički" pristup koji je u pojedinim situacijama pogodniji i pregledniji, naročito kada su u pitanju same matematičke osnove teorije izračunljivosti. Osnovna ideja je da se ulazni i izlazni podaci problema prikažu kao prirodni brojevi, tj. elementi skupa $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

Neka je data konačna azbuka $\Sigma = \{a_1, \ldots, a_k\}$. Definišemo bijekciju $||\cdot|| : \Sigma^* \to \mathbb{N}$ tako da je $||\lambda|| = 0$, dok je za $w = a_{\alpha_1} \ldots a_{\alpha_n}$ ($n \geqslant 1$),

$$||w|| = \alpha_1 k^{n-1} + \ldots + \alpha_{n-1} k + \alpha_n.$$

Ova bijekcija redom lista reči nad Σ dužine $0, 1, 2, \ldots$ poređane (unutar svake grupe) po "leksikografskom" redosledu.

Na ovaj način, svakoj (parcijalnoj) funkciji $f : (\Sigma^*)^n \to \Sigma^*$ pridružujemo (parcijalnu) aritmetičku funkciju $||f|| : \mathbb{N}^n \to \mathbb{N}$ datu sa

$$||f||(||w_1||, \ldots, ||w_n||) = ||f(w_1, \ldots, w_n)||$$

za sve $(w_1, \ldots, w_n)$ iz domena f. Takođe, svaki jezik $L \subseteq \Sigma^*$ možemo identifikovati sa skupom brojeva

$$||L|| = \{||w|| : \ w \in L\}.$$

S druge strane, zapis prirodnih brojeva se u nekom (binarnom, decimalnom, heksadecimalnom,...) brojevnom sistemu nije ništa drugo nego reč nad skupom svih cifara kao azbukom. Prema tome, aritmetičke funkcije takođe možemo koristiti kao formalne modele problema, dok probleme odlučivanja možemo reprezentovati podskupovima skupa $\mathbb{N}$.

1.5 Nastanak teorije algoritama i Čerčova teza

Početak XX veka doneo je pravu revoluciju u razvoju matematičke logike. Istraživanja usmerena na same osnove matematičke metodologije i njenog jezika počela su da zaokupljaju matematičare (kao npr. Džordža Bula) već polovinom

XIX veka, i ta stremljenja bila su krunisana krajem veka pojavom Kantorove teorije skupova. Međutim, ubrzo su uočene suštinske logičke slabosti te teorije, poznate kao *paradoksi* teorije skupova (npr. Raselov paradoks i drugi). Kao odgovor na probleme koji su se javili u samim temeljima matematičke nauke, a inspirisan monumentalnim delom *Principia Mathematica* [50] Vajtheda i Rasela, pojavio se pokret unutar matematike (i filozofije matematike) čiji je cilj bio da celokupnu matematiku svede na *sintaksu*, skup formalnih pravila za manipulaciju matematičkim simbolima, nezavisno od *semantike*, smisla i značenja koje pridajemo tim simbolima. Naravno, cilj tog pokreta bio je da se eliminišu logički paradoksi koji su se javljali u sve većem broju, i da se na taj način — uvođenjem "reda i zakona" u matematičke teorije — sama matematika postavi na zdravije osnove. Ovaj pokret je u istoriji matematike postao poznat kao *formalistički program*. Upravo zahvaljujući ovom programu došlo je do velikog razvitka matematičke logike, a što je u krajnjoj liniji vodilo pojavi računarstva kao naučne discipline.

Predvodnik formalističkog pokreta bio je nemački matematičar *David Hilbert* (1862–1943), najuticajniji matematičar kraja XIX i početka XX veka. Hilbert je verovao u bezgranične mogućnosti ljudskog saznanja i to uverenje ilustruje i natpis na njegovom nadgrobnom spomeniku: *Wir müssen wissen, wir werden wissen*[7]. Formalisti su smatrali da se do svake matematičke istine može doći unutar neke aksiomatske teorije, dakle polazeći od izvesnog broja aksioma i pravila izvođenja koja se primenjuju na mehanički način. To je, između ostalog, značilo i da su Hilbert i formalisti (kao i mnogi matematičari pre njih) verovali da za svaki (adekvatno formulisan) problem postoji algoritam koji ga rešava. Zaista, maksimalni cilj formalističkog programa bio je konstrukcija *dokazivača teorema* — algoritma koji bi za svaku teoremu neke formalne teorije dao njen formalni dokaz, dok bi za preostale formule saopštavao da one nisu posledice datih aksioma.

Na primer, jedna od "najpopularnijih" formalnih teorija tog vremena bila je *Peanova aritmetika* (PA), koja je zamišljena da služi kao formalna osnova teorije brojeva. U skladu sa formalističkim idejama, smatralo se da je sistem aksioma PA dovoljan kako bi se na automatski (tj. algoritamski) način izvela sva tvrđenja koja važe na strukturi prirodnih brojeva $\mathbb{N}$.

Međutim, ovi ambiciozni ciljevi formalističkog projekta doživeli su potpuni krah. 1931. godine, mladi austrijski matematičar *Kurt Gödel* (Kurt Gedel, 1906–1978) objavljuje svoja otkrića [11] koja su stavila tačku na sva nadanja Hilberta

[7] Mi moramo saznati, mi ćemo saznati. (nem.)

i formalista. Naime, ispostavilo se da postoje tvrđenja (tj. formule prvog reda) koje važe u $\mathbb{N}$, ali ipak nisu teoreme formalne teorije PA! Ali, tu se nije radilo samo o pukom propustu, manjkavosti u formulaciji PA koja bi se mogla "zakrpiti" dodavanjem novih aksioma. Gedel je zapravo dokazao da ukoliko pođemo od bilo kojeg skupa formula Θ koje važe na $\mathbb{N}$ takvog da:

- Θ proširuje PA, tj. sadrži sve njene aksiome, i
- *postoji algoritam* koji rešava problem pripadnosti date formule ψ skupu aksioma Θ (a to je osnovni zahtev da bismo uopšte imali aksiomatsku teoriju),

tada uvek postoji formula ϕ takva da ϕ važi na $\mathbb{N}$, ali nije formalna posledica od Θ. Drugim rečima, *ne postoji aksiomatska teorija koja potpuno formalizuje prirodne brojeve*. Kao direktna posledica, dobija se da ne postoji algoritam koji bi za datu formulu ϕ odlučivao da li $\mathbb{N} \models \phi$. Dakle, problem odlučivanja za teoriju strukture $\mathbb{N}$ nije algoritamski rešiv (kažemo još da je teorija $Th(\mathbb{N})$ *neodlučiva*). Bio je to prvi algoritamski nerešiv problem u istoriji matematike.

Naravno, ovi rezultati su u prvi plan izbacili već pomenuto pitanje o matematičkom modelu intuitivnog pojma algoritma. Sam Gedel je pažljivo analizirao taj pojam i kao njegov matematički "pandan" odabrao *(parcijalno) rekurzivne funkcije*, posebnu klasu aritmetičkih funkcija koje ćemo izučavati u narednoj glavi. Tako, Gedel u svojim dokazima nematematičko tvrđenje 'ne postoji algoritam koji rešava problem' zamenjuje sa 'aritmetička funkcija f (koja modelira problem u smislu prethodnog odeljka) nije rekurzivna'. Stoga njegova *teorema nekompletnosti aritmetike* glasi: ne postoji konzistentno rekurzivno proširenje PA koje aksiomatizuje teoriju prirodnih brojeva.

Nedugo zatim, pojavio se čitav niz sistema koji su predlagani za formalni okvir koncepta algoritamske izračunljivosti. Najpre su američki matematičari *Alonzo Church* (Alonzo Čerč, 1903–1995) i *Stephen C. Kleene* (Stiven Klini, 1909–1994) sredinom tridesetih godina razvili *λ-račun*, formalni sistem koji leži u osnovi većine funkcionalnih programskih jezika. Aritmetičke funkcije koje je ovaj sistem mogao da računa bile su upravo Gedelove rekurzivne funkcije. Čerč je smatrao da je ta činjenica više od puke koincidencije, što ga je verovatno navelo da 1936. godine u radu [5] predloži konvenciju, "dogovor" (danas poznat pod imenom *Čerčova teza*) da se *intuitivni pojam algoritma poklapa sa λ-izračunljivošću / rekurzivnim funkcijama*. Međutim, samo nekoliko nedelja nakon što je Čerčov rad izašao iz štampe, engleski matematičar *Alan M. Turing* (Alan Tjuring, 1912–1954) podneo je uredništvu časopisa Londonskog

matematičkog društva svoj rad [47] u kojem uvodi koncept izračunljivosti preko *računskog modela*, idealizacije računske mašine koja će kasnije postati poznata pod imenom *Tjuringova mašina*. Za recenzenta Tjuringovog rada odabran je upravo Čerč, i na njegov predlog, Londonsko matematičko društvo je zahtevalo od Tjuringa da dopuni svoj rad tako što bi svoj koncept izračunljivosti uporedio sa λ-računom. Tjuring je to ubrzo i učinio, i ispostavilo se da su ta dva sistema potpuno ekvivalentna po računskoj moći. Tako je nastala "Tjuringova verzija" Čerčove teze (*Čerč-Tjuringova teza*): *problem je algoritamski rešiv ako i samo ako se može rešiti (isprogramirati) na Tjuringovoj mašini*.

Iste 1936. godine *Emil L. Post* (1897–1954) uvodi *Postove sisteme*, od kojih su kasnije razvijene *formalne gramatike*, a osim toga su se pojavili i *kombinatorna logika*, *normalni algoritmi Markova*, *mašine Minskog*, ***while** programi*, kao i mnogi drugi formalizmi. Svi ovi sistemi računanja su ekvivalentni u smislu da mogu da simuliraju jedni druge, te da je klasa diskretnih funkcija koje ti sistemi mogu da izračunavaju ista u svim slučajevima. Prema tome, reč je o jednoj izuzetno robusnoj klasi funkcija, "otpornoj" na prilično radikalne promene računskih modela, što sve sugeriše da je u pitanju upravo tražena klasa izračunljivih funkcija. Savremena interpretacija Čerčove teze izražava upravo to uverenje: da fenomen simultano obuhvaćen svim pobrojanim formalizmima odgovara našoj intuitivnoj ideji o izračunljivosti. U ravni filozofije nauke, Čerčova teza bi se mogla pobiti jedino ukoliko bi neko prezentovao postupak oko koga bi postojala opšta saglasnost da zadovoljava kriterijume algoritma, a koji ne bi mogao, na primer, biti implementiran na Tjuringovim mašinama. Iako načelna osporavanja Čerčove teze ne prestaju sve do današnjih dana, niko do sada nije uspeo da pronađe takav postupak.

Tako, 1936. možemo uzeti za godinu rođenja nove discipline — *teorije algoritama* (eng. *theory of algorithms*, a ponekad je u upotrebi i termin *teorija izračunljivosti*, eng. *computability theory*). Po Donaldu Knutu, teorija algoritama jeste oblast nauke koja se prvenstveno bavi pitanjem postojanja ili nepostojanja algoritama koji rešavaju pojedine probleme; kao takva, ona leži u sferi matematičke logike. Pojava elektronskih računara nakon II svetskog rata (u čemu je Tjuring ponovo imao ne malu ulogu) dala je ogroman podstrek i opravdanje teoriji algoritama, ali je i dala doprinos pojavi nove grane koja se bavi algoritmima. Budući da je sa stanovišta prakse najinteresantniji pristup pojmu izračunljivosti putem računskog modela (kao što je to Tjuringova mašina ili neka sofisticiranija idealizacija računara, npr. *RAM mašina*), otvara se pitanje ne samo egzistencije algoritma za rešavanje nekog problema, već i njegove efikasnosti. Naime, implementacija algoritma na nekom računskom modelu

troši resurse tog modela, prvenstveno (procesorsko) vreme i prostor (memoriju). Taj utrošak se meri u odnosu na dimenzije ulaznih podataka i naziva se *vremenska / prostorna složenost* razmatranog algoritma. Naravno, složenost jeste kvalitet koji se vezuje za pojedinačni algoritam, ali postoje ograničenja u tom smislu koja su svojstvena samom problemu koji se rešava. Ovakvim pitanjima se bavi *analiza algoritama* (eng. *analysis of algorithms*, ili *teorija računske složenosti*, eng. *computational complexity theory*). Analiza algoritama predstavlja jedan od kamena temeljaca *teorijskog računarsva* (eng. *theoretical computer science*), a od matematičkih metoda najviše koristi tehnike diskretne matematike, matematičke logike i teorije formalnih jezika.

U preostalih pet glava ovog teksta dotaći ćemo neka osnovna pitanja kako teorije, tako i analize algoritama. U naredne dve glave upoznaćemo principe teorije rekurzivnih funkcija, kao i pojedinosti u vezi Tjuringovih mašina, koje su glavna pretpostavka za izučavanje računske složenosti. U četvrtoj glavi ćemo prikazati algoritme za rešavanje nekoliko najpoznatijih problema koji rade u polinomnom vremenu u odnosu na veličinu ulaznih podataka. Najzad, poslednje dve glave su posvećene značajnom fenomenu nedeterminizma, tj. konceptu nedeterminističkog algoritma. Posebnu pažnju obratićemo na tzv. $\mathbf{NP}$-kompletne probleme: pitanje mogućnosti suštinskog ubrzanja algoritama za rešavanje ovih problema predstavlja jednu od ključnih nepoznanica savremenog teorijskog računarstva.

2

Rekurzivne funkcije

Kao što smo to već pomenuli u prethodnoj glavi, Kurt Gedel je 1931. godine, tragajući za adektvatnim matematičkim modelom izračunljivih funkcija, izučavao *rekurzivne funkcije*[8]. U ovoj glavi prikazujemo neke osnovne elemente te teorije. Najpre ćemo upoznati jednu značajnu klasu rekurzivnih funkcija, *prosto rekurzivne funkcije*.

2.1 Prosto rekurzivne funkcije

Klasa prosto rekurzivnih funkcija definiše se induktivnim putem. Najpre definišemo tzv. *osnovne funkcije*:

(1) unarna *nula funkcija* data sa $N(x) = 0$ za sve $x \in \mathbb{N}$,

(2) *sledbenička funkcija* data sa $S(x) = x + 1$, $x \in \mathbb{N}$,

(3) *n-arne projekcije*, funkcije $I_k^n : \mathbb{N}^n \to \mathbb{N}$, $n \geqslant 1$, $1 \leqslant k \leqslant n$, definisane sa

$$I_k^n(x_1, \ldots, x_n) = x_k$$

za sve $x_1, \ldots, x_n \in \mathbb{N}$.

[8] Paralelno sa Gedelom, rekurzivne funkcije je sa sličnom motivacijom razmatrao i *Jacques Herbrand* (Žak Erbran, 1908–1931).

Polazeći od navedenih funkcija, konstruišemo nove funkcije koriseći sledeća dva postupka.

(1) *Kompozicija funkcija.* Pretpostavimo da su date funkcije $g : \mathbb{N}^k \to \mathbb{N}$ i $h_1, \ldots, h_k : \mathbb{N}^n \to \mathbb{N}$. Tada kažemo da je funkcija $f : \mathbb{N}^n \to \mathbb{N}$ data sa

$$f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_k(x_1, \ldots, x_n))$$

za sve $x_1, \ldots, x_n \in \mathbb{N}$ dobijena *kompozicijom* (slaganjem, superpozicijom) funkcija g, odnosno $h_1, \ldots, h_k$.

(2) *Šema proste rekurzije.* Neka su date funkcije $g : \mathbb{N}^n \to \mathbb{N}$ i $h : \mathbb{N}^{n+2} \to \mathbb{N}$. Kažemo da je funkcija $f : \mathbb{N}^{n+1} \to \mathbb{N}$ dobijena iz g, h primenom *šeme proste rekurzije* ukoliko važe sledeće dva uslova:

(i) $f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n)$ za sve $x_1, \ldots, x_n \in \mathbb{N}$,

(ii) za sve $x_1, \ldots, x_n, y \in \mathbb{N}$ važi

$$f(x_1, \ldots, x_n, y + 1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)).$$

Ovde se promenljive $x_1, \ldots, x_n$ obično zovu *parametri rekurzije*, dok je y *glavna promenljiva* rekurzije (tj. rekurzija je definisana "po y").

Aritmetičke funkcije koje se dobijaju od osnovnih konačnom primenom postupaka (1) i (2) su *prosto rekurzivne funkcije*. Primetimo da su sve prosto rekurzivne funkcije *totalne* — domen prosto rekurzivne funkcije od n promenljivih je ceo skup $\mathbb{N}^n$.

Šema proste rekurzije izgleda veoma neobično pri prvom susretu, pre svega zbog svoje "indirektnosti". Tu je funkcija definisana "korak po korak", induktivno: nakon što je zadat početni uslov (uslov (i)), vrednost funkcije f u tački $S(y) = y + 1$ je — funkcijom h — determinisana parametrima rekurzije, vrednošću glavne promenljive, i "prethodnom" vrednošću funkcije f, tj. $f(x_1, \ldots, x_n, y)$.

Ispostavlja se da nam je rekurzija kao postupak neophodna da bismo uopšte definisali osnovne računske radnje na skupu prirodnih brojeva.

Primer 2.1 Od sabiranja, naravno, očekujemo da bude asocijativno, da 0 bude neutralni element, kao i da bude saglasno sa sledbeničkom funkcijom (tj. da zbir $x + 1$ bude istovetan sa vrednošću $S(x)$ za sve $x \in \mathbb{N}$). Ovi zahtevi su već dovoljni da omoguće zadavanje binarne funkcije $+$ šemom proste rekurzije,

gde je prvi sabirak parametar (radi preglednosti koristimo prefiksnu notaciju $+(x, y)$, pored uobičajene infiksne $x + y$):

$$\begin{aligned} +(x, 0) &= x, \\ +(x, y + 1) &= x + (y + 1) = (x + y) + 1 = S(+(x, y)). \end{aligned}$$

Prema tome, sabiranje je dobijeno primenom šeme proste rekurzije na funkcije $g(x) = I_1^1(x)$ i $h(x, y, z) = S(z)$ (u h su promenljive x, y "fiktivne", ali to je u redu, jer možemo formalno pisati $h(x, y, z) = S(I_3^3(x, y, z))$).

Što se tiče množenja, ovde je 0 zaista "nula", 1 je neutralni element, dok $\cdot$ treba da bude distributivno u odnosu na $+$. Stoga (ponovo uz prefiksnu notaciju) imamo:

$$\begin{aligned} \cdot(x, 0) &= 0, \\ \cdot(x, y + 1) &= x(y + 1) = xy + x = +(\cdot(x, y), x). \end{aligned}$$

Tako je ovde $g(x) = N(x)$, a $h(x, y, z) = +(z, x)$ (preciznije, u pitanju je $+(I_3^3(x, y, z), I_1^3(x, y, z))$), pa je množenje prosto rekurzivna funkcija, budući da smo upravo malopre pokazali da je to i sabiranje. □

S druge strane, uobičajeno oduzimanje i deljenje uopšte nisu aritmetičke funkcije, pošto skup $\mathbb{N}$ nije zatvoren na ove operacije (npr. $2 - 6$ i $5/3$ nisu prirodni brojevi). Zbog toga uvodimo aritmetičku funkciju $\dot{-}$ čiji je zadatak da "zameni" oduzimanje:

$$x \dot{-} y = \begin{cases} x - y & x > y, \\ 0 & x \leqslant y. \end{cases}$$

Primer 2.2 Najpre dokazujemo da je $f(x) = x \dot{-} 1$ prosto rekurzivna funkcija. Zaista,

$$\begin{aligned} f(0) &= 0, \\ f(x + 1) &= (x + 1) \dot{-} 1 = x = I_1^1(x) \end{aligned}$$

jeste šema proste rekurzije koja definiše f. Koristeći ovu funkciju, konstruišemo šemu proste rekurzije za $x \dot{-} y$:

$$\begin{aligned} \dot{-}(x, 0) &= x \dot{-} 0 = x, \\ \dot{-}(x, y + 1) &= x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1 = f(\dot{-}(x, y)). \end{aligned}$$

Prema tome, operacija $\dot{-}$ je prosto rekurzivna, a to je i apsolutna vrednost razlike dva prirodna broja $|x - y|$, budući se lako pokazuje da važi formula

$$|x - y| = (x \dot{-} y) + (y \dot{-} x).$$

□

Kada je u pitanju deljenje, njega možemo zameniti parcijalnom funkcijom celobrojnog deljenja $\lfloor x/y \rfloor$, definisanom za $y \neq 0$ (ili na određen način dopunjenom do totalne funkcije za $y = 0$). Međutim, ovom funkcijom ćemo se pozabaviti nešto kasnije.

Zadaci.

1. Dokazati da su sledeće funkcije prosto rekurzivne:

(a) n-arna nula funkcija data sa $N_n(x_1, \ldots, x_n) = 0$, $x_1, \ldots, x_n \in \mathbb{N}$,

(b) konstantna funkcija data sa $f(x_1, \ldots, x_n) = a$, $x_1, \ldots, x_n \in \mathbb{N}$, gde je a fiksiran prirodan broj,

(c) $f(x) = \operatorname{sg} x$,

(d) $f(x) = \overline{\operatorname{sg}}\, x$, funkcija *antisignum*, definisana sa $\overline{\operatorname{sg}}\, 0 = 1$ i $\overline{\operatorname{sg}}\, x = 0$ za sve $x > 0$,

(e)
$$\operatorname{leq}(x, y) = \begin{cases} 1 & x \leqslant y, \\ 0 & x > y, \end{cases}$$

(f)
$$\operatorname{geq}(x, y) = \begin{cases} 1 & x \geqslant y, \\ 0 & x < y, \end{cases}$$

(g) $\min(x, y)$,

(h) $\max(x, y)$.

2. Neka je $f : \mathbb{N}^n \to \mathbb{N}$ prosto rekurzivna funkcija i neka su $i_1, \ldots, i_n \in \{1, \ldots, n\}$ proizvoljni brojevi. Dokazati da je tada funkcija $g : \mathbb{N}^n \to \mathbb{N}$ data sa

$$g(x_1, \ldots, x_n) = f(x_{i_1}, \ldots, x_{i_n})$$

za sve $x_1, \ldots, x_n \in \mathbb{N}$ takođe prosto rekurzivna.

3. Date su dve funkcije $f, f' : \mathbb{N}^n \to \mathbb{N}$ čije se vrednosti razlikuju u samo konačno mnogo tačaka. Dokazati: ako je funkcija f prosto rekurzivna, onda je to i f'.

2.2 Teorema rekurzije

Naravno, glavno pitanje koje se sada postavlja jeste da li je šema proste rekurzije uopšte "legalan", logički korektan način da se definiše aritmetička funkcija. Da li u opštem slučaju postoji funkcija koja za dato g i h zadovoljava uslove (i) i (ii) iz definicije šeme proste rekurzije, i, ako je odgovor na to pitanje potvrdan, da li je funkcija koja zadovoljava te uslove uopšte jedinstvena? Intuicija govori da je tako, ali se ona mora i formalno potvrditi. Ispravnost šeme proste rekurzije kao metode definisanja funkcija posledica je sledećeg opšteg tvrđenja.

Teorema 2.3 (Teorema rekurzije) *Neka su A i B neprazni skupovi i neka su date funkcije $g : A \to B$ i $h : A \times \mathbb{N} \times B \to B$. Tada postoji jedinstvena funkcija $f : A \times \mathbb{N} \to B$ tako da važe sledeći uslovi:*

(i) $f(x, 0) = g(x)$ *za sve* $x \in A$,

(ii) $f(x, n + 1) = h(x, n, f(x, n))$ *za sve* $x \in A$ *i* $n \in \mathbb{N}$.

Dokaz. Najpre ćemo pokazati *jedinstvenost* funkcije f (polazeći od toga da postoji bar jedna funkcija sa traženim uslovima).

Pretpostavimo, dakle, da su $f_1(x, n)$ i $f_2(x, n)$ funkcije koje zadovoljavaju uslove (i) i (ii). Indukcijom po n pokazujemo da za sve $x \in A$ važi $f_1(x, n) = f_2(x, n)$. Zaista, za $n = 0$ imamo

$$f_1(x, 0) = g(x) = f_2(x, 0).$$

Sada pretpostavimo da željena jednakost važi za određenu vrednost n. Tada je

$$f_1(x, n + 1) = h(x, n, f_1(x, n)) = h(x, n, f_2(x, n)) = f_2(x, n + 1),$$

što okončava induktivni dokaz.

Preostaje da pokažemo *egzistenciju* funkcije f sa uslovima (i) i (ii).

U tu svrhu uvodimo novi pojam. Za podskup $S \subseteq A \times \mathbb{N} \times B$ kažemo da je (g, h)-*skup* ako važe sledeći uslovi:

(1) $(x, 0, g(x)) \in S$ za sve $x \in A$,

(2) za sve $x \in A$, $n \in \mathbb{N}$ i $y \in B$ takve da $(x, n, y) \in S$ važi

$$(x, n + 1, h(x, n, y)) \in S.$$

Očigledno, sam skup $A \times \mathbb{N} \times B$ je (g, h)-skup (što pokazuje da (g, h)-skupovi uopšte postoje). Takođe, veoma lako je pokazati da je presek proizvoljne familije (g, h)-skupova takođe (g, h)-skup (provera ovog tvrđenja se prepušta čitaocima).

Označimo sa $\mathcal{F}$ familiju svih (g, h)-skupova. Sada definišemo

$$f = \bigcap_{S \in \mathcal{F}} S.$$

Drugim rečima, u pitanju je *najmanji* (g, h)-skup (imajući u vidu zatvorenost ove klase skupova na preseke, konstatovanu malopre). Naš cilj je da najpre pokažemo da je f funkcija, a zatim i da ona zadovoljava (i) i (ii) iz formulacije teoreme.

Da bi f bila funkcija potrebno je i dovoljno da za sve $x \in A$ i $n \in \mathbb{N}$ postoji *tačno jedno* $y \in B$ tako da je $(x, n, y) \in f$. Ovu tvrdnju dokazujemo indukcijom po n.

Za $n = 0$, po uslovu (1) za proizvoljno $x \in A$ imamo $(x, 0, g(x)) \in f$. Ukoliko bismo imali $(x, 0, y') \in f$ za neko $y' \in B$ tako da je $y' \neq g(x)$, mogli bismo da posmatramo skup trojki $f' = f \setminus \{(x, 0, y')\}$. Sada je dovoljno primetiti da je f' takođe (g, h)-skup: zaista, nijedan od gornjih uslova (1), (2) nije narušen uklanjanjem trojke $(x, 0, y')$ iz f (uslov (2), naime, govori da pod određenim pretpostavkama izvesna trojka čija je druga komponenta $\geqslant 1$ mora biti u f, što nema nikakvih "dodirnih tačaka" sa $(x, 0, y')$). Ovo jasno protivreči minimalnosti skupa f.

Slično postupamo i u samom induktivnom koraku. Naime, polazimo od pretpostavke da za proizvoljno $x \in A$ postoji jedinstveno $y \in B$ tako da $(x, n, y) \in f$. Zbog uslova (2), sledi $(x, n + 1, h(x, n, y)) \in f$. Preostaje da se pokaže da je nemoguće da $(x, n + 1, \hat{y}) \in f$ za neko $\hat{y} \neq h(x, n, y)$. U suprotnom, posmatramo skup $\hat{f} = f \setminus \{(x, n + 1, \hat{y})\}$. Očigledno, uslov (1) iz definicije (g, h)-skupova nije narušen. Međutim, nije narušen ni uslov (2), jer da bi se to dogodilo, potrebno bi bilo da $(x, n, z) \in \hat{f}$, a da pri tom $\hat{y} = h(x, n, z)$ (u kom slučaju bismo imali $(x, n + 1, \hat{y}) \notin \hat{f}$, suprotno uslovu (2)). Ali, po definiciji $\hat{f}$ i po induktivnoj pretpostavci, ovakva situacija je moguća samo ako je $z = y$, pa je $\hat{y} = h(x, n, y)$ — suprotno izboru $\hat{y}$. Prema tome, $\hat{f}$ je (g, h)-skup. Međutim, $\hat{f} \subset f$, pa opet imamo kontradikciju sa minimalnošću skupa f.

Na osnovu dobijene kontradikcije, zaključujemo da je f funkcija. Zapisujući sada uslove (1) i (2) u notaciji uobičajenoj za funkcije, sledi da je $f(x, 0) = g(x)$ i da $y = f(x, n)$ implicira $f(x, n + 1) = h(x, n, y)$; drugim rečima,

$f(x, n+1) = h(x, n, f(x, n))$. Stoga je funkcija f upravo ona koju smo i želeli da konstruišemo, pa je dokaz okončan. □

2.3 Teoreme o zbiru i proizvodu

Kao što je rekurzija jedan od osnovnih postupaka u programiranju, tako je to i *iteracija*. Često smo suočeni sa situacijom da se veličina (tj. funkcija) koju u datoj situaciji želimo da izračunamo ne dobija putem neke zatvorene formule, već ona nastaje kao rezultat iterativnog procesa: sumiranja, proizvoda i slično. Postavlja se pitanje kakav je odnos iteracije i rekurzije. Nije teško videti da je prva zapravo specijalni oblik druge, a formalnu potvrdu takvog viđenja ilustruju i teoreme o zbiru i proizvodu, koje tvrde da suma i proizvod po nekoj promenljivoj prosto rekurzivne funkcije daju ponovo prosto rekurzivnu funkciju.

Teorema 2.4 (Teorema o zbiru) *Neka je $g : \mathbb{N}^n \to \mathbb{N}$ prosto rekurzivna funkcija, $n \geqslant 1$. Tada je funkcija $f : \mathbb{N}^n \to \mathbb{N}$ definisana sa*

$$f(x_1, \ldots, x_{n-1}, x_n) = \sum_{i=0}^{x_n} g(x_1, \ldots, x_{n-1}, i)$$

takođe prosto rekurzivna.

Dokaz. Pokazujemo da se f dobija iz prosto rekurzivnih funkcija primenom šeme proste rekurzije (po x_n). Naime,

$$f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1}, 0),$$

dok je

$$\begin{aligned} f(x_1, \ldots, x_{n-1}, x_n + 1) &= \sum_{i=0}^{x_n+1} g(x_1, \ldots, x_{n-1}, i) = \\ &= \left(\sum_{i=0}^{x_n} g(x_1, \ldots, x_{n-1}, i) \right) + g(x_1, \ldots, x_{n-1}, x_n + 1) = \\ &= f(x_1, \ldots, x_{n-1}, x_n) + g(x_1, \ldots, x_{n-1}, S(x_n)). \end{aligned}$$

Prema tome, prosto rekurzivna funkcija

$$h(x_1, \ldots, x_{n-1}, x_n, y) = y + g(x_1, \ldots, x_{n-1}, S(x_n))$$

generiše traženu šemu. □

Teorema o zbiru ima i uopštenja koja se odnose na granice sumiranja.

Posledica 2.5 *Neka je $g : \mathbb{N}^n \to \mathbb{N}$ prosto rekurzivna funkcija, $n \geqslant 1$. Tada je funkcija $f^* : \mathbb{N}^{n+1} \to \mathbb{N}$ definisana sa*

$$f^*(x_1, \ldots, x_{n-1}, y, z) = \begin{cases} \sum_{i=y}^{z} g(x_1, \ldots, x_{n-1}, i) & y \leqslant z, \\ 0 & y > z, \end{cases}$$

prosto rekurzivna.

Dokaz. Dovoljno je primetiti da važi

$$f^*(x_1, \ldots, x_{n-1}, y, z) = \mathrm{leq}(y, z) \cdot \sum_{i=0}^{z \dot{-} y} g(x_1, \ldots, x_{n-1}, y + i).$$

Naime, ako je $f_1(x_1, \ldots, x_n, x_{n+1})$ funkcija koja nastaje primenom teoreme o zbiru na funkciju $g_1(x_1, \ldots, x_{n-1}, x_n, i) = g(x_1, \ldots, x_{n-1}, x_n + i)$, tada imamo da je

$$f^*(x_1, \ldots, x_{n-1}, y, z) = \mathrm{leq}(y, z) f_1(x_1, \ldots, x_{n-1}, y, z \dot{-} y),$$

odakle sledi tvrđenje. □

Posledica 2.6 *Neka su $g : \mathbb{N}^n \to \mathbb{N}$ i $h, k : \mathbb{N}^{n-1} \to \mathbb{N}$ prosto rekurzivne funkcije, $n \geqslant 1$. Tada je funkcija $\tilde{f} : \mathbb{N}^{n-1} \to \mathbb{N}$ definisana sa*

$$\tilde{f}(x_1, \ldots, x_{n-1}) = \begin{cases} \sum_{i=h(\bar{x})}^{k(\bar{x})} g(x_1, \ldots, x_{n-1}, i) & h(\bar{x}) \leqslant k(\bar{x}), \\ 0 & h(\bar{x}) > k(\bar{x}), \end{cases}$$

prosto rekurzivna (gde je $\bar{x} = (x_1, \ldots, x_{n-1})$).

Dokaz. Imajući u vidu funkciju iz prethodne posledice, dovoljno je uvrstiti funkcije granica sumiranja:

$$\tilde{f}(x_1, \ldots, x_{n-1}) = f^*(x_1, \ldots, x_{n-1}, h(x_1, \ldots, x_{n-1}), k(x_1, \ldots, x_{n-1})).$$

□

Primer 2.7 Posmatrajmo funkciju celobrojnog deljenja q nastalu kompletiranjem odgovarajuće parcijalne funkcije $\lfloor x/y \rfloor$:

$$q(x,y) = \begin{cases} \left\lfloor \frac{x}{y} \right\rfloor & y > 0, \\ x & y = 0. \end{cases}$$

Primetimo da važi:

$$q(x,y) = \sum_{i=1}^{x} \mathrm{geq}(x, iy).$$

Naime, gornja jednakost je trivijalna za $y = 0$. Ako je $y > 0$, tada se zbir sa desne strane sastoji od nekoliko uzasptopnih sabiraka 1 iza kojih sledi niz sabiraka 0. Jasno, broj jedinica u toj sumi jednak je najvećem celobrojnom rešenju (po i) nejednačine $x \geqslant iy$, a to je upravo $\lfloor x/y \rfloor$.

Odavde neposredno sledi da je funkcija $\mathrm{rest}(x,y)$, koja za $y = 0$ vraća 0, a za $y > 0$ ostatak pri deljenju x sa y, prosto rekurzivna, pošto je

$$\mathrm{rest}(x,y) = x \dot{-} q(x,y) \cdot y.$$

Samim tim, prosto rekurzivna je i funkcija div, *indikator deljivosti*:

$$\mathrm{div}(x,y) = \begin{cases} 1 & y \mid x, \\ 0 & y \nmid x, \end{cases}$$

budući da je $\mathrm{div}(x,y) = \mathrm{sg}\, y \cdot \overline{\mathrm{sg}}\, \mathrm{rest}(x,y)$. □

Primer 2.8 Dokazaćemo prostu rekurzivnost funkcije $\mathrm{NZD}(x,y)$ za koju važi $\mathrm{NZD}(0,0) = 0$, dok za ostale tačke $\mathrm{NZD}(x,y)$ vraća najveći zajednički delilac brojeva x i y (pri tome je, naravno, $\mathrm{NZD}(x,0) = \mathrm{NZD}(0,x) = x$ za sve $x > 0$).

U tu svrhu uvodimo pomoćnu funkciju $\delta(x,y,z)$ za koju je $\delta(0,0,z) = 0$ za sve $z \in \mathbb{N}$, dok je za ostale vrednosti para (x,y), $\delta(x,y,z)$ jednako najvećem zajedničkom deliocu brojeva x i y koji nije veći od z. Po dogovoru ćemo definisati $\delta(x,y,0) = 0$ (kao što će se ispostaviti, ova početna vrednost je zapravo nebitna za šemu proste rekurzije koja sledi). Ukoliko je $(x,y) \neq (0,0)$, vrednost $\delta(x,y,z+1)$ se dobija iz $\delta(x,y,z)$ na sledeći način:

- ako $(z+1) \mid x$ i $(z+1) \mid y$, tada je $\delta(x,y,z+1) = z+1$;
- u suprotnom, $\delta(x,y,z+1) = \delta(x,y,z)$.

Zbog toga, važi jednakost:

$$\begin{aligned}\delta(x,y,z+1) &= \mathrm{sg}(x+y)\cdot\Big((z+1)\mathrm{div}(x,z+1)\mathrm{div}(y,z+1)+\\ &\quad+\delta(x,y,z)\overline{\mathrm{sg}}\,[\mathrm{div}(x,z+1)\mathrm{div}(y,z+1)]\Big).\end{aligned}$$

(Između ostalog, važi $\delta(x,y,1)=1$ — nezavisno od toga kako smo definisali $\delta(x,y,0)$.) Time je zadata šema proste rekurzije za δ, pa je reč o prosto rekurzivnoj funkciji. Na kraju, primetimo da je NZD dva broja $\leqslant$ od svakog od njih, sem u slučaju kada je jedan od brojeva 0. Stoga je

$$\mathrm{NZD}(x,y)=\delta(x,y,\max(x,y)),$$

pa je funkcija NZD prosto rekurzivna.

Koristeći poznatu relaciju iz teorije brojeva

$$\mathrm{NZD}(x,y)\mathrm{NZS}(x,y)=xy$$

koja važi za $x,y>0$, dobijamo da je

$$\mathrm{NZS}(x,y)=q(xy,\mathrm{NZD}(x,y)),$$

odakle sledi prosta rekurzivnost funkcije NZS. Najzad, prosto rekurzivna je i *Ojlerova funkcija* φ, pošto je po samoj njenoj definiciji

$$\varphi(x)=\sum_{i=1}^{n}\overline{\mathrm{sg}}\,|\mathrm{NZD}(x,i)-1|.$$

□

Potpuno analogna teoremi o zbiru je teorema o proizvodu.

Teorema 2.9 (Teorema o proizvodu) *Neka je $g:\mathbb{N}^n\to\mathbb{N}$ prosto rekurzivna funkcija, $n\geqslant 1$. Tada je funkcija $f:\mathbb{N}^n\to\mathbb{N}$ definisana sa*

$$f(x_1,\ldots,x_{n-1},x_n)=\prod_{i=0}^{x_n}g(x_1,\ldots,x_{n-1},i)$$

takođe prosto rekurzivna.

Dokaz. Šema proste rekurzije koja određuje funkciju f je sledeća:

$$f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1}, 0),$$

odnosno

$$\begin{aligned} f(x_1, \ldots, x_{n-1}, x_n + 1) &= \prod_{i=0}^{x_n+1} g(x_1, \ldots, x_{n-1}, i) = \\ &= \left(\prod_{i=0}^{x_n} g(x_1, \ldots, x_{n-1}, i) \right) \cdot g(x_1, \ldots, x_{n-1}, x_n + 1) = \\ &= f(x_1, \ldots, x_{n-1}, x_n) g(x_1, \ldots, x_{n-1}, S(x_n)). \end{aligned}$$

□

Takođe, i za proizvod važe tvrđenja koja su analogna malopređašnjim Posledicama 2.5 i 2.6.

Posledica 2.10 *Neka je $g : \mathbb{N}^n \to \mathbb{N}$ prosto rekurzivna funkcija, $n \geqslant 1$. Tada je funkcija $f^\circ : \mathbb{N}^{n+1} \to \mathbb{N}$ definisana sa*

$$f^\circ(x_1, \ldots, x_{n-1}, y, z) = \begin{cases} \prod_{i=y}^{z} g(x_1, \ldots, x_{n-1}, i) & y \leqslant z, \\ 1 & y > z, \end{cases}$$

prosto rekurzivna.

Posledica 2.11 *Neka su $g : \mathbb{N}^n \to \mathbb{N}$ i $h, k : \mathbb{N}^{n-1} \to \mathbb{N}$ prosto rekurzivne funkcije, $n \geqslant 1$. Tada je funkcija $\hat{f} : \mathbb{N}^{n-1} \to \mathbb{N}$ definisana sa*

$$\hat{f}(x_1, \ldots, x_{n-1}) = \begin{cases} \prod_{i=h(\bar{x})}^{k(\bar{x})} g(x_1, \ldots, x_{n-1}, i) & h(\bar{x}) \leqslant k(\bar{x}), \\ 1 & h(\bar{x}) > k(\bar{x}), \end{cases}$$

prosto rekurzivna (gde je $\bar{x} = (x_1, \ldots, x_{n-1})$).

Primer 2.12 Na osnovu Posledice 2.10, $x!$ je prosto rekurzivna funkcija, pošto važi

$$x! = \prod_{i=1}^{x} i\,.$$

□

Zadaci.

1. Dokazati prostu rekurzivnost sledećih funkcija:

(a) $\gamma(x)$ – broj delitelja broja x,

(b) $\sigma(x)$ – zbir delitelja broja x,

(c) $\mathrm{Pr}(x)$ – funkcija koja vraća 1 ako je x prost broj, a 0 u suprotnom,

(d) $\pi(x)$ – broj prostih brojeva $\leqslant x$,

(e) x^y (pri čemu definišemo da je $0^0 = 1$),

(f) $x!!$.

2.4 Operator minimalizacije

U prethodnim odeljcima istakli smo da rekurzija i iteracija spadaju u temeljne postupke u izgradnji algoritama. U te postupke svakako možemo uvrstiti i *pretraživanje*: unapred nam je poznato da za dati niz parametara neki uslov (tj. jednačina $g(x_1, \ldots, x_n, y) = 0$) ima rešenje u prirodnim brojevima (po y), a mi želimo da dođemo do minimalnog rešenja "najprimitivnijim" mogućim putem — direktnom proverom. Upravo ovaj koncept formalizuje *operator minimalizacije*.

Najpre ćemo definisati *neograničeni (eng. unbounded)* operator minimalizacije: u opštem slučaju, on deluje na parcijalne funkcije i rezultuje takođe parcijalnom funkcijom. Neka je $g : \mathbb{N}^{n+1} \to \mathbb{N}$ parcijalna funkcija i $x_1, \ldots, x_n \in \mathbb{N}$ proizvoljni prirodni brojevi. Pišemo da je

$$\mu_y(g(x_1, \ldots, x_n, y) = 0) = a$$

ako i samo ako je $g(x_1, \ldots, x_n, y)$ definisano i različito od 0 za sve $y < a$, dok je $g(x_1, \ldots, x_n, a) = 0$. U suprotnom, vrednost $\mu_y(g(x_1, \ldots, x_n, y) = 0)$ nije definisana. Na taj način, dobijamo n-arnu parcijalnu funkciju

$$f(x_1, \ldots, x_n) = \mu_y(g(x_1, \ldots, x_n, y) = 0),$$

za koju kažemo da je dobijena iz g primenom operatora minimalizacije.

Za parcijalnu aritmetičku funkciju f kažemo da je *parcijalno rekurzivna* ako je dobijena od osnovnih funkcija konačnom primenom kompozicije, šeme proste rekurzije i operatora minimalizacije. Pri tome, za kompoziciju i šemu proste rekurzije primenjene na parcijalne funkcije važe očekivana ograničenja u smislu domena funkcija koje figurišu u definiciji odgovarajućeg postupka.

Primer 2.13 Funkcija celobrojnog deljenja $D(x, y) = \lfloor x/y \rfloor$ je parcijalna, jer je definisana ako i samo ako je $y \neq 0$ (u Primeru 2.7 razmatrali smo njeno kompletiranje $q(x, y)$, koje je prosto rekurzivna funkcija). Razmotrimo sada (pod pretpostavkom $y \neq 0$) jednačinu

$$\left\lfloor \frac{x}{y} \right\rfloor = a.$$

Po definiciji celog dela, ona je ekvivalentna sistemu nejednačina

$$a \leqslant \frac{x}{y} < a + 1,$$

što je zbog $y > 0$ ekvivalentno sa

$$ya \leqslant x < y(a + 1).$$

Prema tome, tražena vrednost a je zapravo *minimalna vrednost promenljive* t za koju važi uslov $y(t + 1) \not\leqslant x$, tj. posredi je minimalno rešenje jednačine

$$\mathrm{leq}(y(t + 1), x) = 0.$$

Otuda je $D(x, y) = \mu_t(\mathrm{leq}(y(t + 1), x) = 0)$, pa je u pitanju parcijalno rekurzivna funkcija. □

Naravno, može se dogoditi da se primenom kompozicija, šema prostih rekurzija i operatora minimalizacije dobije parcijalno rekurzivna funkcija $f : \mathbb{N}^n \to \mathbb{N}$ čiji je domen definisanosti čitav skup $\mathbb{N}^n$. Za takvu parcijalno rekurzivnu funkciju kažemo da je *svuda definisana*.

Drugi način da se u našem razmatranju ograničimo isključivo na totalne funkcije jeste korišćenje *ograničenog* operatora minimalizacije, koji deluje na totalne funkcije i rezultuje totalnom funkcijom. Radi se naprosto o tome da se ograničava *mogućnost* primene operatora μ. Naime, neka je $g : \mathbb{N}^{n+1} \to \mathbb{N}$ (totalna) aritmetička funkcija, takva da je funkcija $f : \mathbb{N}^n \to \mathbb{N}$ data sa

$$f(x_1, \ldots, x_n) = \mu_y(g(x_1, \ldots, x_n, y) = 0)$$

takođe totalna, tj. za sve $x_1, \ldots, x_n \in \mathbb{N}$ postoji $y \in \mathbb{N}$ tako da je

$$g(x_1, \ldots, x_n, y) = 0.$$

Tada je, naravno, funkcija f dobijena iz g primenom operatora minimalizacije; međutim, ukoliko funkcija g ne ispunjava navedeni kriterijum, tada primena ograničenog operatora minimalizacije na g *nije dopuštena*.

Aritmetička funkcija f je *rekurzivna* ako je dobijena od osnovnih funkcija konačnom primenom kompozicije, šeme proste rekurzije i ograničenog operatora minimalizacije.

Primer 2.14 Posmatrajmo totalnu aritmetičku funkciju $f(x) = \lfloor\sqrt{x}\rfloor$. Slično kao i u prethodnom primeru, posmatrajmo jednačinu

$$\lfloor\sqrt{x}\rfloor = a.$$

Ona je ekvivalentna sa $a \leqslant \sqrt{x} < a + 1$ i, dalje, sa

$$a^2 \leqslant x < (a+1)^2.$$

Prema tome, zaključujemo da je a najmanja vrednost promenljive y koja zadovoljava uslov $(y+1)^2 \not\leqslant x$, zbog čega je

$$f(x) = \mu_y(\mathrm{leq}((y+1)^2, x) = 0).$$

Stoga je f rekurzivna funkcija. □

Trivijalno, svaka rekurzivna funkcija je istovremeno i svuda definisana parcijalno rekurzivna funkcija. Međutim, važi i obratno tvrđenje.

Teorema 2.15 *Funkcija f je svuda definisana parcijalno rekurzivna funkcija ako i samo ako je rekurzivna.*

Netrivijalna (i matematički veoma suptilna) implikacija ($\Rightarrow$) gornje teoreme biće dokazana u završnom odeljku ove glave.

Zadaci.

1. Dokazati da je parcijalna funkcija

$$f(x, y) = \lfloor\sqrt[y]{x}\rfloor,$$

definisana za $y > 0$, parcijalno rekurzivna.

2. Dokazati da je parcijalna funkcija

$$f(x, y) = \lfloor\log_y x\rfloor,$$

definisana za $x > 0$ i $y \geqslant 2$, parcijalno rekurzivna.

2.5 Teorema o majoraciji

Pogodnost operatora minimalizacije leži u tome što on u velikom broju slučajeva omogućava da se razmatrana funkcija izrazi preko prostijih funkcija na veoma kompaktan način. Međutim, kao što to sugeriše poređenje Primera 2.7 i 2.13, upotreba operatora minimalizacije u određenim slučajevima nije *neophodna*: razmatrana funkcija f — dobijena primenom ograničenog operatora minimalizacije — može se dobiti iz osnovnih funkcija (možda na komplikovaniji način) samo primenom kompozicija i šema proste rekuruzije (drugim rečima, f je prosto rekurzivna). Uslove pod kojima nastupa takva situacija opisuje sledeća teorema.

Teorema 2.16 (Teorema o majoraciji) *Neka su $g : \mathbb{N}^{n+1} \to \mathbb{N}$ i $\alpha : \mathbb{N}^n \to \mathbb{N}$ prosto rekurzivne funkcije, pri čemu g ispunjava uslove za primenu ograničenog operatora minimalizacije (u odnosu na poslednju promenljivu). Neka je*

$$f(x_1, \ldots, x_n) = \mu_y(g(x_1, \ldots, x_n, y) = 0)$$

i neka za sve $x_1, \ldots, x_n \in \mathbb{N}$ važi nejednakost

$$f(x_1, \ldots, x_n) \leqslant \alpha(x_1, \ldots, x_n).$$

Tada je f prosto rekurzivna funkcija.

Dokaz. Za fiksirane vrednosti $x_1, \ldots, x_n$ označimo (radi kraćeg zapisa) $a = \mu_y(g(x_1, \ldots, x_n, y) = 0)$. Definišemo novu funkciju $h : \mathbb{N}^{n+1} \to \mathbb{N}$ sa:

$$h(x_1, \ldots, x_n, z) = \prod_{i=0}^{z} g(x_1, \ldots, x_n, i).$$

Po teoremi o proizvodu, h je prosto rekurzivna.

Ključna primedba jeste da važi sledeća ekvivalencija:

$$h(x_1, \ldots, x_n, z) = 0 \quad \Longleftrightarrow \quad z \geqslant a.$$

Ovaj zaključak sledi na osnovu izbora a — naime, to je najmanja vrednost promenljive i koja anulira $g(x_1, \ldots, x_n, i)$. Osim toga, po uslovima teoreme važi

$$a \leqslant \alpha(x_1, \ldots, x_n).$$

Na osnovu gornje nejednakosti i prethodne ekvivalencije, sledi

$$a = \sum_{j=0}^{\alpha(x_1,\ldots,x_n)} \operatorname{sg}(h(x_1,\ldots,x_n,j)).$$

Kako izneti niz argumenata važi za proizvoljne parametre $x_1,\ldots,x_n$, dobijamo

$$f(x_1,\ldots,x_n) = \sum_{j=0}^{\alpha(x_1,\ldots,x_n)} \operatorname{sg} \prod_{i=0}^{j} g(x_1,\ldots,x_n,i),$$

pa neposredno zaključujemo da je f prosto rekurzivna funkcija. □

Primer 2.17 U Primeru 2.14 zaključili smo da je

$$\lfloor \sqrt{x} \rfloor = \mu_y(\operatorname{leq}((y+1)^2, x) = 0).$$

Pošto je $g(x,y) = \operatorname{leq}((y+1)^2, x)$ prosto rekurzivna funkcija i za sve $x \in \mathbb{N}$ važi nejednakost $\lfloor \sqrt{x} \rfloor \leqslant x$, po teoremi o majoraciji sledi da je funkcija $f(x) = \lfloor \sqrt{x} \rfloor$ prosto rekurzivna. □

Primer 2.18 (Niz prostih brojeva) Jedna od značajnijih posledica teoreme o majoraciji je prosta rekurzivnost niza prostih brojeva. Pod terminom *niz* podrazumevamo unarnu aritmetičku funkciju $f : \mathbb{N} \to \mathbb{N}$. Rastući niz prostih brojeva označićemo sa p (pri čemu ćemo umesto $p(n)$ često pisati i p_n), tako da je $p(0) = 2$, $p(1) = 3$, $p(2) = 5$, ...

Koristimo funkciju raspodele $\pi(x)$ koja za dato x daje broj prostih brojeva $\leqslant x$ (vidi Zadatak 2.3.1 (d)). Primetimo da za sve $x \in \mathbb{N}$ važi

$$\pi(x+1) - \pi(x) \in \{0, 1\},$$

pri čemu je $\pi(x+1) - \pi(x) = 1$ ako i samo ako je $x+1$ prost broj. Dakle, $\pi(p_x) = x+1$ (jer su prosti brojevi $\leqslant p_x$ baš $p_0, p_1, \ldots, p_x$), dok je $\pi(p_x - 1) = x$. Stoga dobijamo da je p_x zapravo minimalno rešenje (po y) jednačine

$$\pi(y) = x + 1.$$

Otuda sledi

$$p(x) = \mu_y(|\pi(y) - (x+1)| = 0),$$

pa je $p(x)$ rekurzivna funkcija.

Preostaje da pokažemo prostu rekurzivnost niza prostih brojeva primenom teoreme o majoraciji. Kako bismo uspeli u tome, potrebno je da $p(x)$ ograničimo odozgo nekom prosto rekurzivnom funkcijom $\alpha(x)$. Mi ćemo pokazati indukcijom po x da važi nejednakost

$$p_x \leqslant 2^{2^x}.$$

Očigledno, za $x = 0$ imamo jednakost. Sada pretpostavimo da važi $p_n \leqslant 2^{2^n}$ za sve $n \leqslant x$ i dokažimo da je $p_{x+1} \leqslant 2^{2^{x+1}}$. To ćemo dobiti iz sledeće dve nejednakosti:

$$p_{x+1} \leqslant p_0 p_1 \dots p_x + 1 \leqslant 2^{2^{x+1}}.$$

Desna nejednakost sledi neposredno iz induktivne pretpostavke:

$$\begin{aligned} p_0 p_1 \dots p_x + 1 \leqslant 2^{2^0} 2^{2^1} \dots 2^{2^x} + 1 &= 2^{2^0+2^1+\dots+2^x} + 1 \\ &= 2^{2^{x+1}-1} + 1 \leqslant 2^{2^{x+1}}. \end{aligned}$$

Što se tiče leve nejednakosti, posmatrajmo proizvoljan prost faktor p broja

$$p_0 p_1 \dots p_x + 1.$$

To ne može biti ni jedan od $p_0, p_1, \dots, p_x$ (u suprotnom bismo dobili da p_i deli 1 za neko $i \leqslant x$), pa je $p \geqslant p_{x+1}$. Kako je, s druge strane, $p_0 p_1 \dots, p_x + 1 \geqslant p$, sledi željeni zaključak. □

Primer 2.19 Za $x > 0$, označimo sa $\exp_y x$ najviši stepen kojim prost broj p_y deli x, dok je $\exp_y 0 = 0$ za sve $y \in \mathbb{N}$. Uz nešto slobodniju upotrebu notacije (a imajući u vidu osnovnu teoremu aritmetike), tada za svako $x > 0$ možemo pisati

$$x = p_0^{\exp_0 x} p_1^{\exp_1 x} \cdots p_i^{\exp_i x} \cdots$$

Po definiciji, $\exp_y x$ je *najveći* broj k sa osobinom da $p_y^k \mid x$. Primetimo da se taj broj može istovremeno opisati i kao *najmanji* broj k sa osobinom da $p_y^{k+1} \nmid x$. Prema tome,

$$\exp_y x = \mu_k(x \cdot \mathrm{div}(x, p(y)^{k+1}) = 0)$$

(faktor x je dodat kako bismo obezbedili $\exp_y 0 = 0$). Kako je očito $\exp_y x \leqslant x$, po teoremi o majoraciji sledi da je $\exp_y x$ prosto rekurzivna funkcija. □

2.6 Rekurzije višeg reda

Čuveni niz *Fibonačijevih brojeva* definisan je sa $F(0) = F(1) = 1$ i

$$F(x+2) = F(x+1) + F(x)$$

za sve $x \geqslant 0$. Ova definicija je krajnje jednostavna i veoma liči na šemu proste rekurzije, ali ipak nije saglasna sa njom. Da podsetimo, šema proste rekurzije koja definiše niz f određena je brojem $g \in \mathbb{N}$ i binarnom funkcijom $h(x, y)$, tako da je $f(0) = g$ i $f(x+1) = h(x, f(x))$ za sve $x \in \mathbb{N}$.

Očigledno, problem je u tome što se čini da šema proste rekurzije dopušta samo rekurzije koraka 1, dok je Fibonačijev niz definisan rekurzijom koraka 2. Kako, dakle, usaglasiti rekurzije veće (čak, proizvoljne) dubine sa definicijom rekurzivnih funkcija? Odgovor leži u adekvatnoj upotrebi niza prostih brojeva (vidi Primer 2.18). Uz njihovu pomoć, korišćenjem osnovne teoreme aritmetike i funkcije $\exp_y x$, moguće je kodirati proizvoljan konačan niz brojeva jednim jedinim brojem.

Kako bismo pokazali prostu rekurzivnost Fibonačijevog niza, uvodimo pomoćnu funkciju

$$\psi(x) = 2^{F(x)}3^{F(x+1)}.$$

Odavde je

$$\begin{aligned} F(x) &= \exp_0\psi(x), \\ F(x+1) &= \exp_1\psi(x). \end{aligned}$$

Prva od gornje dve jednakosti znači da bi prosta rekurzivnost funkcije ψ imala za posledicu prostu rekurzivnost niza F. Međutim, ψ se sada lako dobija od poznatih prosto rekurzivnih funkcija putem šeme proste rekurzije, budući da je $\psi(0) = 6$ i

$$\begin{aligned} \psi(x+1) &= 2^{F(x+1)}3^{F(x+2)} = 2^{F(x+1)}3^{F(x+1)+F(x)} = \\ &= 2^{\exp_1\psi(x)}3^{\exp_1\psi(x)+\exp_0\psi(x)}. \end{aligned}$$

Opisani "trik" se može uopštiti kroz teoremu o "dubinskoj" rekurziji.

Teorema 2.20 *Neka su $g : \mathbb{N}^n \to \mathbb{N}$ i $h : \mathbb{N}^{n+2} \to \mathbb{N}$ prosto rekurzivne funkcije i neka je funkcija $f : \mathbb{N}^{n+1} \to \mathbb{N}$ data uslovima:*

(1) $f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n)$ *za sve* $x_1, \ldots, x_n \in \mathbb{N}$,

(2) $f(x_1, \ldots, x_n, y{+}1) = h(x_1, \ldots, x_n, y, f^\sharp(x_1, \ldots, x_n, y))$ *za sve* $x_1, \ldots, x_n, y \in \mathbb{N}$,

gde je

$$f^\sharp(x_1, \ldots, x_n, y) = \prod_{i=0}^{y} p_i^{f(x_1,\ldots,x_n,i)}.$$

Tada je f prosto rekurzivna funkcija.

Dokaz. Kako je

$$f(x_1, \ldots, x_n, y) = \exp_y f^\sharp(x_1, \ldots, x_n, y),$$

dovoljno je dokazati da dati uslovi povlače prostu rekurzivnost funkcije $f^\sharp$. Formiramo šemu proste rekurzije:

$$f^\sharp(x_1, \ldots, x_n, 0) = 2^{f(x_1,\ldots,x_n,0)} = 2^{g(x_1,\ldots,x_n)},$$

kao i

$$\begin{aligned} f^\sharp(x_1, \ldots, x_n, y+1) &= \prod_{i=0}^{y+1} p_i^{f(x_1,\ldots,x_n,i)} \\ &= p_{y+1}^{f(x_1,\ldots,x_n,y+1)} \prod_{i=0}^{y} p_i^{f(x_1,\ldots,x_n,i)} \\ &= p_{y+1}^{h(x_1,\ldots,x_n,y,f^\sharp(x_1,\ldots,x_n,y))} f^\sharp(x_1, \ldots, x_n, y). \end{aligned}$$

Tvrđenje teoreme sledi na osnovu proste rekurzivnosti niza prostih brojeva (Primer 2.18). □

2.7 Akermanova funkcija

Imajući u vidu teoremu o rekurziji i primere koji ilustruju njenu primenu, prirodno je zapitati se: da li uopšte postoji rekurzivna funkcija koja nije prosto rekurzivna? Odgovor je pozitivan; međutim, upravo teorema o rekurziji pokazuje zbog čega je teško konstruisati odgovarajući primer. Naime, takva funkcija mora (ako ništa drugo, u asimptotskom smislu) da raste brže od ma koje prosto rekurzivne funkcije!

Sam Gedel je krajem dvadesetih godina XX veka, u prvim pokušajima da dokaže nekompletnost i neodlučivost formalne aritmetike, radio isključivo sa

prosto rekurzivnim funkcijama kao modelom izračunljivosti. Ali, ubrzo su se pojavili prvi primeri rekurzivnih (i očigledno izračunljivih) funkcija u čijoj je izgradnji primena ograničene minimalizacije neizostavna, pa je Gedel morao da modifikuje svoj pristup. Takve funkcije su pronašli — nezavisno jedan od drugog — Hilbertovi učenici *Gabriel Sudan* i *Wilhelm Ackermann* (Vilhelm Akerman, 1896–1962). Prvobitna Akermanova funkcija prezentovana u radu [1] imala je tri promenljive, a binarna funkcija koju danas nazivamo Akermanovom je rezultat pojednostavljenja izvornog primera za koje su zaslužni *Raphael M. Robinson* (1911–1995) i *Rózsa Péter (Politzer)* (1905–1977).

Akermanova funkcija $A : \mathbb{N}^2 \to \mathbb{N}$ je definisana sa sledeća tri uslova koji važe za sve $x, y \in \mathbb{N}$:

(A1) $A(0, y) = y + 1$,

(A2) $A(x + 1, 0) = A(x, 1)$,

(A3) $A(x + 1, y + 1) = A(x, A(x + 1, y))$.

Naravno, pre bilo kakvog razmatranja ove interesantne funkcije, potrebno je dokazati sledeće tvrđenje.

Teorema 2.21 *Uslovi (A1)–(A3) određuju jedinstvenu funkciju.*

Dokaz. Kao i u teoremi rekurzije, najpre dokazujemo *jedinstvenost* Akermanove funkcije. Pretpostavimo da A_1 i A_2 zadovoljavaju uslove (A1)–(A3). Naš cilj je da dokažemo da važi $A_1(x, y) = A_2(x, y)$ za sve $x, y \in \mathbb{N}$. Ovo tvrđenje dokazujemo indukcijom po x.

Za $x = 0$, na osnovu pravila (A1) imamo $A_1(0, y) = y + 1 = A_2(0, y)$. Pretpostavimo sada da za neki fiksni broj $x_0 \in \mathbb{N}$ važi $A_1(x_0, y) = A_2(x_0, y)$ za sve $y \in \mathbb{N}$. Indukcijom po y dokazujemo da važi $A_1(x_0+1, y) = A_2(x_0+1, y)$.

Za $y = 0$, pravilo (A2) i induktivna pretpostavka daju

$$A_1(x_0 + 1, 0) = A_1(x_0, 1) = A_2(x_0, 1) = A_2(x_0 + 1, 0).$$

Sada pretpostavljamo da za neko $y_0 \in \mathbb{N}$ važi $A_1(x_0 + 1, y_0) = A_2(x_0 + 1, y_0)$. Otuda je, na osnovu (A3),

$$A_1(x_0 + 1, y_0 + 1) = A_1(x_0, A_1(x_0 + 1, y_0)) = A_1(x_0, A_2(x_0 + 1, y_0)) =$$

$$= A_2(x_0, A_2(x_0 + 1, y_0)) = A_2(x_0 + 1, y_0 + 1),$$

pri čemu pretposlednja jednakost sledi iz induktivne pretpostavke prve indukcije. Ovim su oba induktivna dokaza istovremeno okončana.

Za dokaz *egistencije* funkcije $A(x,y)$ koja zadovoljava (A1)–(A3), uvodimo relaciju strogog poretka $\prec$ na skupu $\mathbb{N}^2$ svih parova prirodnih brojeva, tako da je

$$(a,b) \prec (c,d) \quad \Longleftrightarrow \quad a < c \text{ ili } a = c,\ b < d.$$

Za $a, b \in \mathbb{N}$ označavamo

$$M_{(a,b)} = \{(x,y) :\ (x,y) \prec (a,b)\}.$$

Ovaj skup zapravo predstavlja skup svih parova koji se nalaze u nultoj, prvoj, $\ldots$, $(a-1)$-voj vrsti (ako $\mathbb{N}^2$ predstavimo u vidu beskonačne "tabele"), kao i onih parova koji se nalaze u a-toj vrsti, ali su "levo" od para (a,b). Za svaki par $(a,b) \in \mathbb{N}^2$ definisaćemo parcijalnu funkciju $A_{(a,b)}$ sa domenom $M_{(a,b)}$ tako da je za $(a,b) \prec (a',b')$ funkcija $A_{(a,b)}$ restrikcija od $A_{(a',b')}$ (tj. $A_{(a',b')}$ proširuje $A_{(a,b)}$), tako da $A = \bigcup_{(a,b)\in\mathbb{N}^2} A_{(a,b)}$ ima željene osobine. Pri tome, možemo se ograničiti na slučaj $a \geqslant 1$, pošto je na $M_{(1,0)}$ funkcija A eksplicitno definisana (naime, $(x,y) \prec (1,0)$ znači da je $x = 0$, i iz (A1) imamo $A(0,y) = y+1$). Zato ćemo definisati $A_{(1,0)}(0,y) = y+1$.

Sada za $(a,b) \succ (1,0)$ treba da konstruišemo funkciju $A_{(a,b)}$, polazeći od pretpostavke da je $A_{(\alpha,\beta)}$ definisano za sve $(\alpha,\beta) \prec (a,b)$. Razlikujemo tri slučaja. Ako je $b > 1$, tada je

$$A_{(a,b)}(x,y) = \begin{cases} A_{(a,b-1)}(x,y) & (x,y) \prec (a,b-1), \\ A_{(a,b-1)}(a-1, A_{(a,b-1)}(a,b-1)) & (x,y) = (a,b-1), \end{cases}$$

pri čemu je ova definicija dobra budući da $(a,b-1), (a-1,c) \in M_{(a,b)}$ za sve $c \in \mathbb{N}$. Za $b = 1$ definišemo

$$A_{(a,1)}(x,y) = \begin{cases} A_{(a,0)}(x,y) & x < a, \\ A_{(a,0)}(a-1,1) & (x,y) = (a,0), \end{cases}$$

gde je dovoljno zapaziti da $(a-1,1) \in M_{(a,1)}$. Najzad, ako je $b = 0$, stavljamo $A_{(a,0)} = \bigcup_{\alpha<a} \bigcup_{\beta\in\mathbb{N}} A_{(\alpha,\beta)}$.

Preostaje da proverimo da funckija $\bigcup_{(a,b)\in\mathbb{N}^2} A_{(a,b)}$ ima osobine (A1)–(A3). Međutim, ovo lako sledi iz gornjih definicija i činjenice da za sve $(x,y) \prec (a,b)$ važi $A(x,y) = A_{(a,b)}(x,y)$. □

U narednom pokazujemo da funkcija $A(x,y)$ nije prosto rekurzivna. Ispostaviće se da Akermanova funkcija raste toliko brzo da je nijedna prosto rekurzivna funkcija ne može nadmašiti.

Lema 2.22 *Za sve $x, y \in \mathbb{N}$ važe sledeće nejednakosti:*

(1) $A(x, y) > y$,

(2) $A(x, y + 1) > A(x, y)$,

(3) $A(x + 1, y) > A(x, y)$,

(4) $A(x + 1, y) \geqslant A(x, y + 1)$.

Dokaz. Ilustracije radi, dokazujemo samo nejednakost (1), dok se ostale dokazuju na sličan način. Dokaz sprovodimo (kao u Teoremi 2.21, a i u većini tvrđenja o Akermanovoj funkciji) dvostrukom indukcijom. Najpre, indukcijom po x započinjemo dokaz nejednakosti $A(x, y) > y$.

Za $x = 0$ imamo neposredno $A(0, y) = y + 1 > y$ za sve $y \in \mathbb{N}$. Pretpostavimo sada da za neko $x_0 \in \mathbb{N}$ važi $A(x_0, y) > y$ za sve $y \in \mathbb{N}$. Nejednakost $A(x_0 + 1, y) > y$ dokazujemo indukcijom po y.

Za $y = 0$ imamo (po uslovu (A2) i induktivnoj pretpostavci prve indukcije) $A(x_0+1, 0) = A(x_0, 1) > 1 > 0$. Dalje, pretpostavimo da važi $A(x_0+1, y_0) > y_0$ za neko $y_0 \in \mathbb{N}$. Tada sledi:

$$A(x_0 + 1, y_0 + 1) = A(x_0, A(x_0 + 1, y_0)) > A(x_0 + 1, y_0) > y_0,$$

pri čemu prva nejednakost sledi po induktivnoj pretpostavci prve, a druga po induktivnoj pretpostavci druge indukcije. Kako smo u gornjem nizu koristili *dve* stroge nejednakosti, dobijamo da je $A(x_0 + 1, y_0 + 1) \geqslant y_0 + 2$, pa zapravo važi bolja ocena

$$A(x_0 + 1, y_0 + 1) > y_0 + 1,$$

što je i trebalo dobiti. Ovaj zaključak istovremeno okončava oba induktivna dokaza. □

Lema 2.23 *Za proizvoljne $x_1, \ldots, x_n, y \in \mathbb{N}$, $n \geqslant 2$, važi*

$$\sum_{i=1}^{n} A(x_i, y) < A(x, y),$$

gde je $x = \max(x_1, \ldots, x_n) + 4(n - 1)$.

Dokaz. Lako se vidi da je dovoljno lemu dokazati za slučaj $n = 2$. Tada je za $z = \max(x_1, x_2)$,

$$A(x_1, y) + A(x_2, y) \leqslant 2A(z, y) < 2A(z, y) + 3 = A(2, A(z, y)) <$$

$$< A(z+2, A(z+3, y)) = A(z+3, y+1) \leqslant A(z+4, y) = A(x, y),$$

pri čemu smo koristili prethodnu lemu i činjenicu (koja se lako uočava) da je $A(2, y) = 2y + 3$. □

Lema 2.24 *Neka je $f : \mathbb{N}^n \to \mathbb{N}$ prosto rekurzivna funkcija. Tada postoji $c \in \mathbb{N}$ tako da za sve $x_1, \ldots, x_n \in \mathbb{N}$ važi nejednakost*

$$f(x_1, \ldots, x_n) < A(c, x_1 + \ldots + x_n).$$

Dokaz. Lemu dokazujemo indukcijom po složenosti funkcije f. Najpre razmatramo slučaj ako je f osnovna funkcija. Ako je f nula funkcija, imamo

$$N(x) = 0 < A(0, x) = x + 1,$$

pa je $c = 0$ pogodan izbor. Za sledbeničku funkciju je

$$S(x) = x + 1 = A(0, x) < A(1, x),$$

pa $c = 1$ zadovoljava zahteve leme. Najzad, za projekcije je

$$I_k^n(x_1, \ldots, x_n) = x_k \leqslant x_1 + \ldots + x_n < A(0, x_1 + \ldots + x_n),$$

stoga $c = 0$ ponovo odgovara.

Pređimo na slučaj kada je funkcija f dobijena kompozicijom,

$$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)),$$

pri čemu su induktivne pretpostavke da postoje $c_1, \ldots, c_k, d \in \mathbb{N}$ tako da važe sledeće nejednakosti:

$$\begin{aligned} g_1(x_1, \ldots, x_n) &< A(c_1, x_1 + \ldots + x_n), \\ &\vdots \\ g_k(x_1, \ldots, x_n) &< A(c_k, x_1 + \ldots + x_n), \\ h(y_1, \ldots, y_k) &< A(d, y_1 + \ldots + y_k), \end{aligned}$$

za sve $x_1, \ldots, x_n, y_1, \ldots, y_k \in \mathbb{N}$. Tako je (uz zapis $\bar{x} = (x_1, \ldots, x_n)$):

$$\begin{aligned} f(\bar{x}) &= h(g_1(\bar{x}), \ldots, g_k(\bar{x})) \\ &< A(d, g_1(\bar{x}) + \ldots + g_k(\bar{x})) \\ &< A(d, A(c_1, x_1 + \ldots + x_n) + \ldots + A(c_k, x_1 + \ldots + x_n)) \\ &< A(d, A(c_1 + \ldots + c_k + 4k - 4, x_1 + \ldots + x_n)) \\ &< A(c' + d, A(c' + d + 1, x_1 + \ldots + x_n)) \\ &= A(c' + d + 1, x_1 + \ldots + x_n + 1) \\ &\leqslant A(c' + d + 2, x_1 + \ldots + x_n), \end{aligned}$$

gde smo označili $c' = c_1 + \ldots + c_k + 4k - 4$. Ovde prve dve nejednakosti slede iz induktivne pretpostavke, treća iz prethodne leme, a preostale dve na osnovu Leme 2.22. Prema tome, $c = c' + d + 2$ ispunjava uslove leme.

Najzad, posmatramo slučaj kada je f dobijeno šemom proste rekurzije:

$$\begin{aligned} f(\bar{x}, 0) &= g(\bar{x}), \\ f(\bar{x}, y+1) &= h(\bar{x}, y, f(\bar{x}, y)), \end{aligned}$$

gde imamo

$$\begin{aligned} g(\bar{x}) &< A(c_1, x_1 + \ldots + x_n), \\ h(\bar{x}, y, z) &< A(c_2, x_1 + \ldots + x_n + y + z). \end{aligned}$$

Pokazujemo da važi

$$f(\bar{x}, y) < A(c, x_1 + \ldots + x_n + y),$$

za $c = \max(c_1, c_2) + 4n + 1$, tako što ćemo indukcijom po y dokazati jaču nejednakost

$$x_1 + \ldots + x_n + y + f(\bar{x}, y) < A(c, x_1 + \ldots + x_n + y).$$

Za $y = 0$ sledi

$$\begin{aligned} x_1 + \ldots + x_n + 0 + f(\bar{x}, 0) = x_1 + \ldots + x_n + g(\bar{x}) < \\ < x_1 + \ldots + x_n + A(c_1, x_1 + \ldots + x_n) \leqslant \\ \leqslant nA(0, x_1 + \ldots + x_n) + A(c_1, x_1 + \ldots + x_n) < \\ < A(c_1 + 4n - 4, x_1 + \ldots + x_n) < A(c, x_1 + \ldots + x_n). \end{aligned}$$

Sada posmatramo sledeće nejednakosti:

$$\begin{aligned} x_1 + \ldots x_n + y + 1 + f(\bar{x}, y+1) = \\ = x_1 + \ldots x_n + y + 1 + h(\bar{x}, y, f(\bar{x}, y)) < \\ < (n+1)A(0, x_1 + \ldots + x_n + y) + \\ + A(c_2, x_1 + \ldots + x_n + y + f(\bar{x}, y)) + 1 < \\ < A(c_2 + 4n, A(c, x_1 + \ldots + x_n + y)) + 1 \leqslant \\ \leqslant A(c-1, A(c, x_1 + \ldots + x_n + y)) + 1 = \\ = A(c, x_1 + \ldots + x_n + y + 1) + 1. \end{aligned}$$

Kako se u ovom nizu pojavljuju bar dve stroge nejednakosti, imamo (slično kao u dokazu Leme 2.22):

$$x_1 + \ldots x_n + y + 1 + f(\bar{x}, y+1) < A(c, x_1 + \ldots + x_n + y + 1),$$

što je i trebalo dokazati. □

Propozicija 2.25 *Akermanova funkcija nije prosto rekurzivna.*

Dokaz. Pretpostavimo suprotno. Definišimo funkciju $a(x) = A(x, x)$, za koju sledi da je prosto rekurzivna. Tada, po prethodnoj lemi, postoji $c \in \mathbb{N}$ tako da za sve $x \in \mathbb{N}$ važi nejednakost $a(x) < A(c, x)$. Međutim, za $x = c$, ovo znači

$$A(c, c) = a(c) < A(c, c).$$

Kontradikcija. □

kor.	izraz	skraćenica	Gedelov br.		
0	$A(2,1)$	$2,1$	$2^3 3^2$	$=$	72
1	$A(1,A(2,0))$	$1,2,0$	$2^2 3^3 5^1$	$=$	540
2	$A(1,A(1,1))$	$1,1,1$	$2^2 3^2 5^2$	$=$	900
3	$A(1,A(0,A(1,0)))$	$1,0,1,0$	$2^2 3^1 5^2 7^1$	$=$	2100
4	$A(1,A(0,A(0,1)))$	$1,0,0,1$	$2^2 3^1 5^1 7^2$	$=$	2940
5	$A(1,A(0,2))$	$1,0,2$	$2^2 3^1 5^3$	$=$	1500
6	$A(1,3)$	$1,3$	$2^2 3^4$	$=$	324
7	$A(0,A(1,2))$	$0,1,2$	$2^1 3^2 5^3$	$=$	2250
8	$A(0,A(0,A(1,1)))$	$0,0,1,1$	$2^1 3^1 5^2 7^2$	$=$	7350
9	$A(0,A(0,A(0,A(1,0))))$	$0,0,0,1,0$	$2^1 3^1 5^1 7^2 11^1$	$=$	16170
10	$A(0,A(0,A(0,A(0,1))))$	$0,0,0,0,1$	$2^1 3^1 5^1 7^1 11^2$	$=$	25410
11	$A(0,A(0,A(0,2)))$	$0,0,0,2$	$2^1 3^1 5^1 7^3$	$=$	10290
12	$A(0,A(0,3))$	$0,0,3$	$2^1 3^1 5^4$	$=$	3750
13	$A(0,4)$	$0,4$	$2^1 3^5$	$=$	486
14	5	5	2^6	$=$	64
15		5	2^6	$=$	64
16		5	2^6	$=$	64
$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabela 2.1: Postupak izračunavanja vrednosti $A(2,1)$

Sada prelazimo na dokaz rekurzivnosti Akermanove funkcije. U tu svrhu, bliže ćemo posmatrati postupak izračunavanja vrednosti $A(x, y)$. Na primer, tok izračunavanja vrednosti $A(2, 1)$ dat je u prethodnoj tabeli (gde prva kolona označava redni broj koraka, a druga izraz do kojeg smo došli tokom izračunavanja; o preostale dve kolone će biti reči kasnije).

Primetimo sledeće: postupak izračnavanja teče tako što se na *najviše "ugnežđen"* izraz oblika $A(n, m)$ primenjuje jedno od pravila (A1)–(A3), pri čemu je moguće primeniti tačno jedno od pravila. Pri tome, pravilo (A3) produžava izraz, (A1) ga skraćuje, a (A2) mu ne menja dužinu. Ovde smo pravila (A1)–(A3) u stvari "orijentisali" (leva strana se zamenjuje desnom), što potencijalno može umanjiti opštost izračunavanja. Međutim, upravo dokaz Teoreme 2.21 pokazuje da se ovom strategijom za datu tačku (x, y) uvek dobija vrednost funkcije A u konačno mnogo koraka. Drugim rečima, dokazujući Teoremu 2.21, mi smo zapravo opisali algoritam za izračunavanje $A(x, y)$.

Očigledno, tokom tog algoritma pojavljivaće se isključivo izrazi oblika

$$A(x_0, A(x_1, \ldots, A(x_{k-1}, x_k)).$$

Stoga je suvišno pisati simbol funkcije A i zagrade; izraz do kojeg se u datom koraku došlo je u potpunosti određen nizom brojeva

$$x_0, x_1, \ldots, x_{k-1}, x_k.$$

Sve konačne nizove prirodnih brojeva možemo, međutim, kodirati jednim jedinim brojem, tzv. *Gedelovim brojem* niza:

$$p_0^{x_0+1} p_1^{x_1+1} \ldots p_k^{x_k+1}$$

(dodavanje 1 u eksponentima je nužno kako bi iz Gedelovog broja bilo moguće jednoznačno rekonstruisati niz, kako bi se npr. razlikovali Gedelovi brojevi nizova $1, 2$ i $1, 2, 0$). U tabeli su u trećoj i četvrtoj koloni dati odgovarajući skraćeni zapisi izraza i njihovi Gedelovi brojevi. Pri tome smo stavili da se konačan rezultat izračunavanja neograničeno ponavlja u svim narednim koracima.

Sada uvodimo aritmetičku funkciju $f : \mathbb{N}^3 \to \mathbb{N}$ tako što definišemo da je $f(x, y, z)$ jednako Gedelovom broju niza argumenata izraza dobijenog u z-tom koraku izračunavanja vrednosti $A(x, y)$. Ako se to izračunavanje završava tačno u z_0-tom koraku, tada je $f(x, y, z) = f(x, y, z_0)$ za sve $z \geqslant z_0$. S druge strane, svi brojevi $f(x, y, z)$, $z \leqslant z_0$, moraju biti različiti — u suprotnom bi se izračunavanje $A(x, y)$ zatvorilo u mrtvu petlju i nikada ne bi bilo okončano, a to protivreči algoritmu koji proističe iz Teoreme 2.21 (vidi ranije primedbe).

Prema tome, rezultat izračunavanja $A(x, y)$ prepoznajemo po *prvoj ponovljenoj vrednosti u nizu* $f(x, y, z)$, $z \in \mathbb{N}$. Ta vrednost je, očito, jednaka $2^{A(x,y)+1}$. Prema tome,

$$A(x, y) = \exp_0 f(x, y, \mu_z(|f(x, y, z) - f(x, y, z + 1)| = 0)) \dot{-} 1.$$

Stoga je rekurzivnost Akemanove funkcije direktna posledica sledećeg tvrđenja.

Propozicija 2.26 *Funkcija $f(x, y, z)$ je (prosto) rekurzivna.*

Dokaz. Konstruisaćemo šemu proste rekurzije koja daje funkciju $f(x, y, z)$. Za početak, primetimo da je $f(x, y, 0)$ Gedelov broj niza x, y, pa je zato

$$f(x, y, 0) = 2^{x+1}3^{y+1},$$

što je očito prosto rekurzivna funkcija po x, y.

Sada ćemo pretpostaviti da je n Gedelov broj nekog niza argumenata u izrazu koji nastaje tokom izračunavanja neke vrednosti Akermanove funkcije, i razlikovaćemo slučajeve u odnosu na to koje se od pravila (A1)–(A3) može primeniti na izraz reprezentovan sa n.

Pre razmatranja ovih slučajeva, primetimo da se dužina niza reprezentovanog sa n može dobiti kao broj različitih prostih faktora od n:

$$d(n) = \sum_{i=2}^{n} \mathrm{div}(n, i) \cdot \mathrm{Pr}(i).$$

(1) *Pravilo (A1) se može primeniti na niz argumenata reprezentovan sa n.* Ovaj slučaj nastaje tačno onda kada je je niz arugmenata dužine bar 2 i kada je pretposlednji broj u tom nizu jednak 0. Zbog toga je karakteristična funkcija posmatranog skupa Gedelovih brojeva

$$\chi_1(n) = \mathrm{sg}\left((2 \dot{-} d(n)) + \left|1 - \exp_{d(n)\dot{-}2} n\right|\right),$$

tj. uočeni skup je prosto rekurzivan. U ovom slučaju, niz argumenata

$$x_0, \ldots, 0, x_{d-1}$$

transformiše se u

$$x_0, \ldots, x_{d-1} + 1.$$

Stoga imamo da je

$$n = 2^{x_0+1}3^{x_1+1}\dots p^1_{d(n)-2}p^{x_{d-1}+1}_{d(n)-1},$$

a Gedelov broj $g_1(n)$ novog niza je

$$2^{x_0+1}3^{x_1+1}\dots p^{x_{d-1}+2}_{d(n)-2}.$$

Pošto je $x_{d-1} = \exp_{d(n)\dot{-}1}n \mathbin{\dot{-}} 1$, možemo izraziti

$$g_1(n) = \left\lfloor \frac{n \cdot p(d(n) \mathbin{\dot{-}} 2)^{\exp_{d(n)\dot{-}1}n}}{p(d(n) \mathbin{\dot{-}} 1)^{\exp_{d(n)\dot{-}1}n}} \right\rfloor.$$

(2) *Pravilo (A2) se može primeniti na niz argumenata reprezentovan sa* n*.* Ovaj slučaj nastupa ako i samo ako odgovarajući niz argumenata ima poslednji član 0, a pretposlednji član > 0 (pri čemu, naravno, niz ima bar dva člana). Odgovarajuća karakteristična funkcija je

$$\chi_2(n) = \operatorname{sg}\left((2 \mathbin{\dot{-}} d(n)) + \left|1 - \exp_{d(n)\dot{-}1}n\right| + \left(2 \mathbin{\dot{-}} \exp_{d(n)\dot{-}2}n\right)\right).$$

Niz argumenata

$$x_0, \dots, x_{d-2}, 0$$

transformisao se u

$$x_0, \dots, x_{d-2} - 1, 1,$$

pa je Gedelov broj polaznog niza bio

$$n = 2^{x_0+1}3^{x_1+1}\dots p^{x_{d-2}+1}_{d(n)-2}p^1_{d(n)-1}$$

a novog

$$2^{x_0+1}3^{x_1+1}\dots p^{x_{d-2}}_{d(n)-2}p^2_{d(n)-1},$$

tj.

$$g_2(n) = \left\lfloor \frac{n \cdot p(d(n) \mathbin{\dot{-}} 1)}{p(d(n) \mathbin{\dot{-}} 2)} \right\rfloor.$$

(3) *Pravilo (A3) se može primeniti na niz argumenata reprezentovan sa* n*.* Ovaj slučaj imamo ako i samo ako je niz argumenata dužine bar 2, a poslednja dva člana su > 0. Toj situaciji odgovara karakteristična funkcija

$$\chi_3(n) = \operatorname{sg}\left((2 \mathbin{\dot{-}} d(n)) + \left(2 \mathbin{\dot{-}} \exp_{d(n)\dot{-}1}n\right) + \left(2 \mathbin{\dot{-}} \exp_{d(n)\dot{-}2}n\right)\right).$$

U ovom slučaju, pošli smo od niza argumenata

$$x_0, \ldots, x_{d-2}, x_{d-1},$$

a dobijamo

$$x_0, \ldots, x_{d-2} - 1, x_{d-2}, x_{d-1} - 1.$$

To znači da smo od Gedelovog broja

$$n = 2^{x_0+1}3^{x_1+1} \ldots p_{d(n)-2}^{x_{d-2}+1} p_{d(n)-1}^{x_{d-1}+1}$$

dobili

$$2^{x_0+1}3^{x_1+1} \ldots p_{d(n)-2}^{x_{d-2}} p_{d(n)-1}^{x_{d-2}+1} p_{d(n)}^{x_{d-1}},$$

odnosno

$$g_3(n) = \left\lfloor \frac{n \cdot p(d(n) \dot{-} 1)^{\exp_{d(n)\dot{-}2} n} p(d(n))^{\exp_{d(n)\dot{-}1} n \dot{-} 1}}{p(d(n) \dot{-} 2) \cdot p(d(n) \dot{-} 1)^{\exp_{d(n)\dot{-}1} n}} \right\rfloor .$$

Najzad, preostaje da se dodefiniše da Gedelov broj ostaje n ukoliko nijedno od pravila (A1)–(A3) nije primenljivo (tj. kada smo stigli do rezultata izračunavanja), i da se uoči da važi

$$f(x, y, z + 1) = G(x, y, f(x, y, z)),$$

gde je

$$G(x, y, z) = g_1(z) \cdot \overline{\text{sg}}\, \chi_1(z) + g_2(z) \cdot \overline{\text{sg}}\, \chi_2(z) + g_3(z) \cdot \overline{\text{sg}}\, \chi_3(z) +$$

$$+ z \cdot \chi_1(z)\chi_2(z)\chi_3(z).$$

Kako su funkcije $g_i(x), \chi_i(x)$, $i = 1, 2, 3$, prosto rekurzivne, to je prosto rekurzivna i funkcija $f(x, y, z)$. □

Primetimo da smo u prethodnom uspeli da Akermanovu funkciju izrazimo korišćenjem *samo jednog* operatora minimalizacije. Ovo nije slučajno: može se pokazati (što ćemo i učiniti u Odeljku 2.10) da se svaka parcijalno rekurzivna funkcija može dobiti iz osnovnih funkcija korščenjem kompozicija, šema prostih rekurzija i *najviše jednog* operatora minimalizacije.

2.8 Kantorova enumeracija

Kantorova enumeracija je bijekcija $c : \mathbb{N}^2 \to \mathbb{N}$. Ona se, zapravo, pojavljuje u Kantorovom dokazu o ekvipotentnosti skupova $\mathbb{Q}$ i $\mathbb{N}$. Ispitivanje ove funkcije i utvrđivanje njene proste rekurzivnosti predstavlja zanimljivu vežbu, ali će imati i značajnu ulogu u narednim odeljcima.

Kako se uspostavlja ova bijekcija? Najpre se svi parovi prirodnih brojeva poređaju u beskonačnu pravougaonu šemu:

$$\begin{array}{cccccc} (0,0) & (0,1) & (0,2) & (0,3) & \cdots \\ (1,0) & (1,1) & (1,2) & (1,3) & \cdots \\ (2,0) & (2,1) & (2,2) & (2,3) & \cdots \\ (3,0) & (3,1) & (3,2) & (3,3) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

Zatim se parovi enumerišu "dijagonalno" — na istoj dijagonali su parovi sa istim zbirom komponenti (tako se 0-ta dijagonala sastoji samo iz para $(0,0)$, prva sadrži dva para $(0,1)$ i $(1,0)$, itd.), a prebrajanje na datoj dijagonali ide odozgore naniže, tj. zdesna nalevo. Tako je, na primer, $c(0,0) = 0$, $c(0,1) = 1$, $c(1,0) = 2$, $c(0,2) = 3$, itd.

Naš prvi zadatak je da izvedemo opšti izraz za $c(x,y)$ i da se tako uverimo da je c prosto rekurzivna funkcija. U tom smislu ključna primedba je da je $c(x,y)$ ništa drugo nego broj parova koji prethode paru (x,y) u opisanom poretku.

Najpre, primetimo da se par (x,y) nalazi na dijagonali koju "odlikuje" zbir komponenti $x+y$. Prema tome, "prethodni" parovi su oni koji se nalaze na dijagonalama $0, 1, \ldots, x+y-1$, kao i prethodni parovi na dijagonali broj $x+y$, a to su

$$(0, x+y), \ldots, (x-1, y+1).$$

Njih ima x. Takođe, primetimo da se na dijagonali br. d nalazi $d+1$ par. Na osnovu izloženog, dobijamo

$$c(x,y) = (1 + \ldots + (x+y)) + x = \left\lfloor \frac{(x+y)(x+y+1)}{2} \right\rfloor + x.$$

Naravno, zanimljiv je i problem određivanja inverzne funkcije $c^{-1} : \mathbb{N} \to \mathbb{N}^2$ — za dato $n \in \mathbb{N}$ se pitamo koji je to par baš n-ti po redu u Kantorovoj enumeraciji. Radi operativnosti, pišimo

$$c^{-1}(n) = (\ell(n), r(n)).$$

Naš drugi cilj je da odredimo funkcije $\ell(n), r(n)$ i da se uverimo da su one takođe prosto rekurzivne.

Prema tome, potrebno je da iz uslova

$$\frac{(x+y)(x+y+1)}{2} + x = n$$

odredimo x, y. To se na prvi pogled čini kao nemoguć zadatak: rešavanje dve nepoznate iz jedne jedine jednačine. Međutim, ograničenje da su x, y, n prirodni brojevi i specifični oblik funkcije $c(x, y)$ učiniće ovaj cilj ostvarivim.

Najpre imamo, množenjem sa 2 i sređivanjem:

$$2n = x^2 + y^2 + 2xy + y + 3x,$$

odakle je

$$8n + 1 = 4x^2 + 4y^2 + 8xy + 4y + 12x + 1.$$

Izraz na desnoj strani se može transformisati na dva načina:

$$8n + 1 = (2x + 2y + 1)^2 + 8x = (2x + 2y + 3)^2 - 8y - 8,$$

tako da važe nejednakosti

$$(2x + 2y + 1)^2 \leqslant 8n + 1 < (2x + 2y + 3)^2.$$

Korenovanjem i dodatnim elementarnim transformacijama sledi

$$x + y + 1 \leqslant \frac{\sqrt{8n+1}+1}{2} < x + y + 2,$$

drugim rečima,

$$\left\lfloor \frac{\sqrt{8n+1}+1}{2} \right\rfloor = x + y + 1.$$

Gornja jednakost omogućava da se iz izraza za $c(x, y)$ u potpunosti eliminiše $x + y$, pa preostaje linearna jednačina po x koju odmah neposredno rešavamo. Tako je

$$\ell(n) = x = n \mathbin{\dot{-}} \left\lfloor \frac{\left(\left\lfloor \frac{\sqrt{8n+1}+1}{2} \right\rfloor \mathbin{\dot{-}} 1 \right) \left\lfloor \frac{\sqrt{8n+1}+1}{2} \right\rfloor}{2} \right\rfloor,$$

kao i

$$r(n) = \left\lfloor \frac{\sqrt{8n+1}+1}{2} \right\rfloor \mathbin{\dot{-}} (\ell(n) + 1).$$

Prosta rekurzivnost ovih funkcija je posledica proste rekurzivnosti funkcije

$$\alpha(n) = \left\lfloor \frac{\sqrt{8n+1}+1}{2} \right\rfloor,$$

koja je, sa svoje strane posledica identiteta

$$\left\lfloor \frac{x}{a} \right\rfloor = \left\lfloor \frac{\lfloor x \rfloor}{a} \right\rfloor,$$

za $x \in \mathbb{R}$, $a \in \mathbb{Z} \setminus \{0\}$, budući da je zbog njega

$$\left\lfloor \frac{\sqrt{8n+1}+1}{2} \right\rfloor = \left\lfloor \frac{\lfloor \sqrt{8n+1} \rfloor + 1}{2} \right\rfloor.$$

2.9 Rekurzivni i rekurzivno nabrojivi skupovi

U prvoj glavi (Odeljak 1.3) izdvojili smo posebnu klasu problema čiji je izlaz binaran: u pitanju su *problemi odlučivanja*. Ovi problemi su konceptualno najjednostavniji, pa su zbog toga pogodni za naša teorijska razmatranja. S druge strane, oni su dovoljno reprezentativni i značajni da pruže vernu sliku o pojmu algoritma u celini.

Videli smo da se problem odlučivanja može formalizovati tako što se najpre fiksira azbuka Σ, konačan ili prebrojiv skup simbola (pomoću kojih se izražavaju svi relevantni elementi problema), a zatim se problem odlučivanja identifikuje sa parom jezika (Γ, L) nad Σ (podskupova od Σ^*), pri čemu Γ predstavlja skup *instanci*, svih potencijalnih ulaznih podataka datog problema, dok se jezik $L \subseteq \Gamma$ sastoji od svih instanci koje rezultuju izlazom DA.

U praksi, ova formalizacija se može dodatno pojednostaviti. Naime, razumno je da *a priori* pođemo od pretpostavke da je problem pripadnosti date reči $w \in \Sigma^*$ skupu instanci Γ algoritamski rešiv[9], jer bi se u suprotnom teško moglo uopšte govoriti o korektno formulisanom problemu odlučivanja. Na primer, bilo bi prilično besmisleno razmatrati pitanje da li je data iskazna formula tautologija ili ne ukoliko ne bismo imali algoritam koji prepoznaje da li je uneti niz simbola uopšte iskazna formula. Zbog toga, problem odlučivanja u ovom nešto uprošćenom pristupu možemo identifikovati sa jezikom $L \subseteq \Sigma^*$. Prelaskom

[9] Pri tome, u najvećem broju slučajeva koji figurišu u primenama, odgovarajući algoritmi su prilično jednostavni.

na "aritmetičku" verziju ovog formalizma (vidi Odeljak 1.4), umesto jezika L možemo razmatrati skup prirodnih brojeva

$$||L|| = \{||w|| : \ w \in L\}.$$

Na opisani način, dobijamo da je problem odlučivanja određen nekim podskupom od $\mathbb{N}$.

Ako je $A \subseteq \mathbb{N}$, definišemo *karakterističnu funkciju* skupa A sa

$$\chi_A(x) = \begin{cases} 0 & x \in A, \\ 1 & x \notin A. \end{cases}$$

Skup A je *(prosto) rekurzivan* ako je takva njegova karakteristična funkcija, dok je jezik L (prosto) rekurzivan ako je $||L||$ (prosto) rekurzivan skup. Pri tome, smatramo da je problem odlučivanja modeliran sa L algoritamski rešiv ukoliko je L rekurzivan jezik. Klasu svih rekurzivnih jezika označavamo sa $\mathbf{R}$.

Pojmu rekurzivnog skupa srodan je pojam rekurzivno nabrojivog skupa. Naime, za skup $A \subseteq \mathbb{N}$ kažemo da je *rekurzivno nabrojiv* ako postoji binarna prosto rekurzivna funkcija f tako da je

$$a \in A \iff (\exists x \in \mathbb{N})\, f(a, x) = 0.$$

Drugim rečima, rekurzivno nabrojivi skupovi su skupovi parametara koji neku prosto rekurzivnu jednačinu čine rešivom. Slično kao i malopre, za jezik L kažemo da je rekurzivno nabrojiv ako je $||L||$ rekurzivno nabrojiv skup. Klasu svih rekurzivno nabrojivih jezika označavamo sa $\mathbf{RE}$.

Odmah se uočava da važi

Lema 2.27 *Svaki prosto rekurzivan skup je rekurzivno nabrojiv.*

Dokaz. Posmatrajmo funkciju

$$f(a, x) = \chi_A(a) + x.$$

Jasno, postoji $x \in \mathbb{N}$ tako da je $f(a, x) = 0$ ako i samo ako je $\chi_A(a) = 0$, tj. $a \in A$ (i tada je jedino moguće rešenje $x = 0$). □

U svetlu iznetog, izraz "rekurzivno nabrojiv" deluje malo neobično. Taj naziv je, međutim, u priličnoj meri opravdan sledećim tvrđenjem.

Teorema 2.28 *Neprazan skup $A \subseteq \mathbb{N}$ je rekurzivno nabrojiv ako i samo ako postoji unarna prosto rekurzivna funkcija φ tako da je A njen kodomen, tj.*

$$A = \{\varphi(n) : \ n \in \mathbb{N}\}.$$

Dokaz. ($\Leftarrow$) Posmatrajmo prosto rekurzivnu funkciju $f(a,x) = |\varphi(x) - a|$. Jasno, jednačina $f(a,x) = 0$ ima rešenje po x ako i samo ako postoji $x \in \mathbb{N}$ tako da je $a = \varphi(x)$, tj. ako i samo ako je $a \in A$. Stoga f pokazuje da je A rekurzivno nabrojiv.

($\Rightarrow$) Pretpostavimo da je A rekurzivno nabrojiv skup, tako da je $a \in A$ ako i samo ako $f(a,x) = 0$ ima rešenje po x. Sada definišemo

$$\varphi(n) = \ell(n) \,\overline{\text{sg}}\, f(\ell(n), r(n)) + b \,\text{sg}\, f(\ell(n), r(n)),$$

gde je $b \in A$ proizvoljan, ali fiksiran element. Naravno, funkcije ℓ, r su komponente inverzne Kantorove enumeracije (tako je par $(\ell(n), r(n))$ n-ti po redu). Naš cilj je da pokažemo da je A baš skup svih slika funkcije φ.

Razmatramo dva slučaja. Ako je $f(\ell(n), r(n)) = 0$, to znači da je jednačina $f(\ell(n), x) = 0$ rešiva (jedno rešenje je $x = r(n)$), pa $\ell(n) \in A$. Ali, sada se istovremeno lako dobija $\varphi(n) = \ell(n)$, pa $\varphi(n) \in A$. U suprotnom, važi $f(\ell(n), r(n)) \neq 0$, pa je $\varphi(n) = b$; ponovo je $\varphi(n) \in A$. To pokazuje da je skup svih vrednosti funkcije φ sadržan u A.

Preostaje da pokažemo da je svaki element $a \in A$ oblika $\varphi(n)$ za pogodno n. Kako znamo da $f(a,x) = 0$ mora imati bar jedno rešenje po x, odaberimo jedno takvo rešenje x_0. Neka je sada $n = c(a, x_0)$, tj. $a = \ell(n)$, i $x_0 = r(n)$. Dobijamo:

$$\varphi(n) = a \,\overline{\text{sg}}\, f(a, x_0) + b \,\text{sg}\, f(a, x_0) = a,$$

što se i tražilo. □

Ovde treba napomenuti da važi i opštije tvrđenje od Leme 2.27.

Teorema 2.29 *Svaki rekurzivan skup je rekurzivno nabrojiv.*

Kao direktnu posledicu gornje teoreme, dobijamo $\mathbf{R} \subseteq \mathbf{RE}$ (u Glavi 3 ćemo pokazati da je ova inkluzija stroga). Gornju teoremu ćemo dokazati u narednom odeljku. Ovaj odeljak okončavamo jednim kriterijumom rekurzivnosti koji na efektan način demonstrira ulogu i značaj pojma rekurzivne nabrojivosti. Naime, lako se vidi da je komplement rekurzivnog skupa takođe rekurzivan: dovoljno je samo posmatrati antisignum karakteristične funkcije polaznog

skupa. Međutim, nije tačno da je komplement rekurzivno nabrojivog skupa uvek rekurzivno nabrojiv. Naprotiv, situaciju u kojoj se to dešava precizno određuje sledeći rezultat.

Teorema 2.30 (E.L.Post) *Neka je $A \subseteq \mathbb{N}$. Ako su oba skupa A i $\overline{A}$ rekurzivno nabrojivi, onda je skup A rekurzivan.*

Dokaz. S obzirom na pretpostavke, date su nam dve binarne prosto rekurzivne funkcije f, g tako da $f(a, x) = 0$ ima rešenje po x ako i samo ako $a \in A$, dok $g(a, x) = 0$ ima rešenje po x ako i samo ako $a \in \overline{A}$, tj. $a \notin A$.

Definišimo funkciju

$$h(a) = \mu_x(f(a, x)g(a, x) = 0).$$

Tvrdimo da je

$$\chi_A(a) = \operatorname{sg} f(a, h(a)).$$

Posmatrajmo dva slučaja. Ako $a \in A$, tada jednačina $f(a, x) = 0$ ima rešenje, a $g(a, x) = 0$ ga nema. Kako je $h(a)$ definisano kao najmanja vrednost za x koja anulira proizvod $f(a, x)g(a, x)$, sledi da u ovom slučaju imamo $f(a, h(a)) = 0$, pa je

$$\chi_A(a) = 0 = \operatorname{sg} f(a, h(a)).$$

Ako pak $a \notin A$, situacija je upravo obratna: mora biti $g(a, h(a)) = 0$, pa je zato $f(a, h(a)) \neq 0$. Tako imamo

$$\chi_A(a) = 1 = \operatorname{sg} f(a, h(a)).$$

Ovo potvrđuje naše tvrđenje i okončava dokaz teoreme, budući da je sada χ_A očigledno kompozicija rekurzivnih funkcija. □

Intuitivno, rekurzivno nabrojiv skup A treba zamisliti kao ”crnu kutiju”, proceduru koja u nekom nepoznatom redosledu i sa eventualnim ponavljanjima ispisuje elemente skupa A. Ta procedura, u načelu, *nije* dovoljna kao algoritam koji bi odlučio problem pripadnosti skupu A: ona može samo da *potvrdi* da neki broj x pripada A (jer će nakon konačno mnogo vremena ispisati broj x), ali ne da to i *opovrgne* (jer ”crna kutija” nakon konačno mnogo vremena može da ispiše samo konačno ”parče” od A, a o ostatku tog skupa ništa ne znamo). S druge strane, ako imamo *dve* ”crne kutije”, po jednu za A i $\overline{A}$, njih dve zajedno jesu ekvivalentne algoritmu: nakon konačno mnogo vremena broj x će biti ispisan

od strane jedne od njih, i tada preostaje samo da se utvrdi koja ”kutija” je to učinila da bi se algoritamski odgovorilo na pitanje da li je $x \in A$. Ova mentalna predstava jeste zapravo ideja koja je formalizovana kroz Postovu teoremu.

Zadaci.

1. Dokazati da je skup prostih brojeva prosto rekurzivan.

2. Dokazati da je skup svih stepena prostih brojeva prosto rekurzivan.

3. Dokazati da je za svako $k \geqslant 2$, skup svih k-tih stepena prirodnih brojeva prosto rekurzivan.

4. Dokazati da je svaka

(a) aritmetička progresija

(b) geometrijska progresija

koju čine prirodni brojevi prosto rekurzivan skup.

5. Neka je φ unarna prosto rekurzivna funkcija za koju važi $\varphi(x) \geqslant x$ za sve $x \in \mathbb{N}$. Dokazati da je skup $A = \{\varphi(n) : n \in \mathbb{N}\}$ prosto rekurzivan.

2.10 Parcijalno rekurzivne funkcije i rekurzivna nabrojivost

Glavni rezultat završnog odeljka ove glave je Teorema 2.33 koja uz Lemu 2.32 daje ”parametarski oblik” parcijalno rekurzivne funkcije $f : \mathbb{N}^n \to \mathbb{N}$, pri čemu se ona posmatra kao $(n+1)$-arna relacija na $\mathbb{N}$, tj. podskup od $\mathbb{N}^{n+1}$. Ova teorema ima sama po sebi veliki teorijski značaj, a omogućiće nam i da dokažemo Teoremu 2.29 (tj. inkluziju $\mathbf{R} \subseteq \mathbf{RE}$), kao i još nekoliko zanimljivih posledica.

Kako bismo uopšte bili u mogućnosti da formulišemo Teoremu 2.33, potrebno je da proširimo pojmove rekurzivnosti i rekurzivne nabrojivosti sa skupova prirodnih brojeva na skupove uređenih n-torki prirodnih brojeva. Kao što se to radi i kod drugih prebrojivih kolekcija matematičkih objekata (reči, formula, grafova,...), veza se ostvaruje izborom neke standardne bijekcije između odgovarajućeg domena problema i $\mathbb{N}$. Na primer, u Odeljku 2.8 smo upoznali *Kantorovu enumeraciju*, bijekciju $c : \mathbb{N}^2 \to \mathbb{N}$. Na osnovu nje, možemo lako konstruisati bijekciju $\mathbb{N}^n \to \mathbb{N}$ za proizvoljno $n \geqslant 2$. Naime, neka je $c^2(x_1, x_2)$

"sinonim" za $c(x_1, x_2)$. Tada za $n \geqslant 2$ induktivno definišemo funkcije

$$c^{n+1}(x_1, x_2, \ldots, x_{n+1}) = c^n(c(x_1, x_2), x_3, \ldots, x_{n+1}).$$

Jasno, pri tome je $c^n : \mathbb{N}^n \to \mathbb{N}$. Broj $c^n(x_1, \ldots, x_n)$ zovemo *Kantorov indeks* n-torke $(x_1, \ldots, x_n)$. Očigledno, sve posmatrane funkcije oblika c^n su prosto rekurzivne. Takođe, ako ℓ, r predstavljaju komponente inverzne bijekcije c^{-1}, tada se lako dobija da su rešenja jednačine

$$c(x_1, \ldots, x_n) = y$$

sledeća:

$$\begin{aligned} x_n &= r(y), \\ x_{n-1} &= r\ell(y), \\ &\vdots \\ x_2 &= r\ell \ldots \ell(y), \\ x_1 &= \ell\ell \ldots \ell(y). \end{aligned}$$

Prema tome, i komponente inverzne funkcije $(c^n)^{-1}$ su takođe prosto rekurzivne. U daljem ćemo desne strane gornjih jednakosti redom označavati sa $c_{n,n}(y), \ldots, c_{n,2}(y), c_{n,1}(y)$. Dakle, imamo identitete

$$c^n(c_{n,1}(x), \ldots, c_{n,n}(x)) = x$$

i

$$c_{n,i}(c^n(x_1, \ldots, x_n)) = x_i.$$

Kažemo da je skup $A \subseteq \mathbb{N}^n$ *rekurzivan* / *rekurzivno nabrojiv* ako je skup njegovih Kantorovih indeksa $c^n(A)$ rekurzivan / rekurzivno nabrojiv podskup od $\mathbb{N}$.

Sada dajemo dve karakterizacije rekurzivno nabrojivih podskupova od $\mathbb{N}^n$.

Lema 2.31 *Skup $A \subseteq \mathbb{N}^n$ je rekurzivno nabrojiv ako i samo ako postoji prosto rekurzivna funkcija $F : \mathbb{N}^{n+m} \to \mathbb{N}$, $m \geqslant 1$, tako da važi:*

$$(a_1, \ldots, a_n) \in A \iff (\exists x_1) \ldots (\exists x_m) F(a_1, \ldots, a_n, x_1, \ldots, x_m) = 0.$$

Dokaz. ($\Rightarrow$) Neka je A rekurzivno nabrojiv skup, tj. neka je $c^n(A)$ rekurzivno nabrojiv. Tada postoji prosto rekurzivna funkcija $g : \mathbb{N}^2 \to \mathbb{N}$ tako da $a \in c^n(A)$ ako i samo ako jednačina $g(a, x) = 0$ ima rešenje po x.

Uslov $(a_1, \ldots, a_n) \in A$ ekvivalentan je uslovu

$$c^n(a_1, \ldots, a_n) \in c^n(A),$$

što je po pretpostavci dalje ekvivalentno sa rešivošću jednačine

$$g(c^n(a_1, \ldots, a_n), x) = 0$$

po x. Prema tome, prosto rekurzivna funkcija

$$F(a_1, \ldots, a_n, x) = g(c^n(a_1, \ldots, a_n), x)$$

zadovoljava traženu ekvivalenciju, uz izbor $m = 1$.

($\Leftarrow$) Neka je F prosto rekurzivna funkcija i A skup koji zadovoljava datu ekvivalenciju. Posmatrajmo funkciju

$$f(a, x) = F(c_{n,1}(a), \ldots, c_{n,n}(a), c_{m,1}(x), \ldots, c_{m,m}(x)).$$

Ova funkcija je prosto rekurzivna. Tvrdimo da $f(a, x)$ "svedoči" da je $c^n(A) \subseteq \mathbb{N}$ rekurzivno nabrojiv skup.

Zaista, ako $a \in c^n(A)$, tada je $a = c^n(a_1, \ldots, a_n)$ za neku n-torku $(a_1, \ldots, a_n) \in A$. U tom slučaju, postoji $(x_1, \ldots, x_m) \in \mathbb{N}^m$ tako da je

$$F(a_1, \ldots, a_n, x_1, \ldots, x_m) = 0,$$

pa po osobinama Kantorove enumeracije sledi da je

$$f(a, c^m(x_1, \ldots, x_m)) = 0.$$

Obratno, ako je $f(a, x) = 0$, iz date ekvivalencije sledi da

$$(c_{n,1}(a), \ldots, c_{n,n}(a)) \in A,$$

odakle $a \in c^n(A)$. □

Lema 2.32 *Neprazan skup $A \subseteq \mathbb{N}^n$ je rekurzivno nabrojiv ako i samo ako postoje unarne prosto rekurzivne funkcije $\varphi_1, \ldots, \varphi_n$ tako da je*

$$A = \{(\varphi_1(t), \ldots, \varphi_n(t)) : \ t \in \mathbb{N}\}.$$

Dokaz. ($\Rightarrow$) Ako je $c^n(A)$ rekurzivno nabrojiv skup, po Teoremi 2.28 postoji unarna prosto rekurzivna funkcija φ čiji je skup vrednosti baš $c^n(A)$. Ali, tada je A skup vrednosti funkcije $(c^n)^{-1}\varphi$, tj.

$$A = \{(c_{n,1}(\varphi(t)), \ldots, c_{n,n}(\varphi(t))) : \ t \in \mathbb{N}\}.$$

($\Leftarrow$) Skup Kantorovih indeksa datog skupa A je

$$\{c^n(\varphi_1(t), \ldots, \varphi_n(t)) : \ t \in \mathbb{N}\},$$

tj. $c^n(A)$ je skup vrednosti neke unarne prosto rekurzivne funkcije. Po Teoremi 2.28, A je rekurzivno nabrojiv skup. □

Gornja lema je zapravo višedimenzionalno uopštenje Teoreme 2.28.

Najavljeno glavno tvrđenje ovog odeljka je sledeće.

Teorema 2.33 *Neka je $f : \mathbb{N}^n \to \mathbb{N}$ parcijalna funkcija. Tada je f parcijalno rekurzivna ako i samo ako je f rekurzivno nabrojiv podskup od $\mathbb{N}^{n+1}$.*

Dokaz. Tvrđenje teoreme je očigledno ako f nije nigde definisana, tj. $f = \varnothing$, pa ćemo u daljem pretpostaviti da je f definisana u bar jednoj tački.

($\Leftarrow$) Neka je f rekurzivno nabojiv podskup od $\mathbb{N}^{n+1}$. Tada je, po Lemi 2.32,

$$f = \{(\varphi_1(t), \ldots, \varphi_{n+1}(t)) : \ t \in \mathbb{N}\}$$

za odgovarajuće prosto rekurzivne funkcije $\varphi_1, \ldots, \varphi_{n+1}$. Primetimo da je parcijalna funkcija f definisana za n-torku $(x_1, \ldots, x_n) \in \mathbb{N}^n$ ako i samo ako postoji t koje je rešenje jednačine

$$|\varphi_1(t) - x_1| + \ldots + |\varphi_n(t) - x_n| = 0.$$

Za svako takvo rešenje t važi $f(x_1, \ldots, x_n) = \varphi_{n+1}(t)$. Prema tome,

$$f(x_1, \ldots, x_n) = \varphi_{n+1}(\mu_t(|\varphi_1(t) - x_1| + \ldots + |\varphi_n(t) - x_n| = 0)),$$

pri čemu je desna strana definisana tačno za n-torke $(x_1, \ldots, x_n)$ iz domena definisanosti f. Stoga je f parcijalno rekurzivna funkcija.

($\Rightarrow$) Ovu implikaciju dokazujemo indukcijom po složenosti parcijalno rekurzivne funkcije f. Kako se za osnovne funkcije lako pokazuje njihova rekurzivna nabrojivost, naš induktivni dokaz se svodi na sledeća tri tvrđenja.

(A) Parcijalna funkcija dobijena primenom kompozicije na rekurzivno nabrojive parcijalne funkcije je rekurzivno nabojiv skup.

(B) Parcijalna funkcija dobijena primenom šeme proste rekurzije na rekurzivno nabrojive parcijalne funkcije je rekurzivno nabojiv skup.

(C) Parcijalna funkcija dobijena primenom operatora minimalizacije na rekurzivno nabrojivu parcijalnu funkciju je rekurzivno nabojiv skup.

Dokaz za (A). Neka je

$$f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_k(x_1, \ldots, x_n)),$$

pri čemu je desna strana definisana ako i samo ako $(x_1, \ldots, x_n)$ pripada domenu definisanosti f. Imajući u vidu Lemu 2.32, polazimo od pretostavke da je

$$g = \{(\alpha_1(t), \ldots, \alpha_{k+1}(t)) : \ t \in \mathbb{N}\},$$

kao i

$$h_i = \{(\beta_{i,1}(t), \ldots, \beta_{i,n+1}(t)) : \ t \in \mathbb{N}\}$$

za sve $1 \leqslant i \leqslant k$, gde su sve funkcije $\alpha_i, \beta_{i,j}$ prosto rekurzivne. Po definiciji kompozicije (parcijalnih) funkcija, $(a_1, \ldots, a_n, b) \in f$ ako i samo ako postoje $c_1, \ldots, c_k$ tako da

$$(a_1, \ldots, a_n, c_i) \in h_i$$

za sve $1 \leqslant i \leqslant k$, i

$$(c_1, \ldots, c_k, b) \in g.$$

Po našim pretpostavkama, egzistencija ovakvih prirodnih brojeva c_i, $1 \leqslant i \leqslant k$, ekvivalentna je egzistenciji $t_0, t_1, \ldots, t_k$ tako da važe sledeće jednakosti:

$$\begin{aligned}
&\beta_{i,1}(t_i) = a_1, \quad \ldots \quad \beta_{i,n}(t_i) = a_n, \qquad (1 \leqslant i \leqslant k)\\
&\alpha_1(t_0) = \beta_{1,n+1}(t_1), \quad \ldots \quad \alpha_k(t_0) = \beta_{k,n+1}(t_k),\\
&\alpha_{k+1}(t_0) = b.
\end{aligned}$$

Prema tome, ako uvedemo funkciju

$$F(x_1, \ldots, x_n, y, t_0, t_1, \ldots, t_k) =$$

$$= \sum_{i=1}^{k} \sum_{j=1}^{n} |\beta_{i,j}(t_i) - x_j| + \sum_{i=1}^{k} |\alpha_i(t_0) - \beta_{i,n+1}(t_i)| + |\alpha_{k+1}(t_0) - y|,$$

tada $(a_1, \ldots, a_n, b) \in f$ ako i samo ako jednačina

$$F(a_1, \ldots, a_n, b, t_0, t_1, \ldots, t_k) = 0$$

ima rešenje po $t_0, t_1, \ldots, t_k$. Po Lemi 2.31, f je rekurzivno nabrojiv skup.

Dokaz za (B). Neka je parcijalna funkcija $f : \mathbb{N}^{n+1} \to \mathbb{N}$ dobijena od parcijalnih funkcija g, h primenom šeme proste rekurzije, tako da je

$$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n)$$

i

$$f(x_1, \ldots, x_n, y + 1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$$

za sve $x_1, \ldots, x_n, y \in \mathbb{N}$, pri čemu su obe strane gornje dve jednakosti definisane za iste vrednosti promenljivih. Uočimo (u smislu Leme 2.32) parametarske zapise ovih funkcija:

$$\begin{aligned} g &= \{(\gamma_1(t), \ldots, \gamma_{n+1}(t)) : \ t \in \mathbb{N}\} \\ h &= \{(\delta_1(t), \ldots, \delta_{n+3}(t)) : \ t \in \mathbb{N}\}. \end{aligned}$$

Funkciju f ćemo razbiti na dva podskupa: prvi, f_1, će činiti $(n+2)$-torke $(x_1, \ldots, x_n, y, z)$ kod kojih je $y = 0$, dok one kod kojih je $y > 0$ čine f_2. Po Lemi 2.32, f_1 je rekurzivno nabrojiv skup, budući da je

$$f_1 = \{(\gamma_1(t), \ldots, \gamma_n(t), N(t), \gamma_{n+1}(t)) : \ t \in \mathbb{N}\}.$$

Pošto se lako pokazuje da je unija dva rekurzivno nabrojiva skupa rekurzivno nabrojiva, preostaje da se pokaže da je skup f_2 rekurzivno nabrojiv.

Naime, primetimo najpre da je $(a_1, \ldots, a_n, b, c) \in f_2$ ekvivalentno sa postojanjem prirodnih brojeva $u_0, u_1, \ldots, u_{b \dot{-} 1}$ takvih da je

$$\begin{aligned} u_0 &= g(a_1, \ldots, a_n) \\ u_1 &= h(a_1, \ldots, a_n, 0, u_0), \\ u_2 &= h(a_1, \ldots, a_n, 1, u_1), \\ &\vdots \\ c &= h(a_1, \ldots, a_n, b \dot{-} 1, u_{b \dot{-} 1}). \end{aligned}$$

Jednakost oblika $u_{i+1} = h(a_1, \ldots, a_n, i, u_i)$ ekvivalentna je egzistenciji broja t_{i+1} tako da je

$$\delta_1(t_{i+1}) = a_1, \quad \ldots \quad \delta_n(t_{i+1}) = a_n,$$

$$\delta_{n+1}(t_{i+1}) = i, \quad \delta_{n+2}(t_{i+1}) = u_i, \quad \delta_{n+3}(t_{i+1}) = u_{i+1}.$$

Prema tome, $(a_1, \ldots, a_n, b, c) \in f_2$ ako i samo ako jednačina

$$\sum_{j=1}^{n} |\gamma_j(t_0) - a_j| + |\gamma_{n+1}(t_0) - \delta_{n+2}(t_1)| +$$

$$+ \sum_{i=1}^{\dot{b-1}} \left(\sum_{j=1}^{n} |\delta_j(t_i) - a_j| + |\delta_{n+1}(t_i) - i| + |\delta_{n+3}(t_i) - \delta_{n+2}(t_{i+1})| \right) +$$

$$+ |\delta_{n+3}(t_{\dot{b-1}}) - c| = 0 \qquad (*)$$

ima rešenje po $t_0, t_1, \ldots, t_{\dot{b-1}}$. Međutim, dokaz tvrđenja (B) se ovde ne završava, jer nailazimo na ozbiljne tehničke poteškoće. Prvo, broj promenljivih po kojima se ova jednačina rešava nije konstantan, već zavisi od vrednosti promenljive b. Slično važi i za zbir u drugom redu gornje jednačine: granica sumiranja zavisi od parametra b, a, s druge strane, primena teoreme o zbiru nije moguća, jer se vezana promenljiva i pojavljuje u indeksima drugih, slobodnih promeljivih (npr. t_i). Ove probleme moramo prevazići tako što ćemo pronaći način da ove promenljive indekse "podignemo" na nivo pravih promenljivih.

Upravo se taj cilj postiže pomoću *Gedelove* Γ*-funkcije*. Ona je definisana sa

$$\Gamma(x, y) = \mathrm{rest}(\ell(x), 1 + (y + 1)r(x)),$$

$x, y \in \mathbb{N}$, odakle odmah sledi da je u pitanju prosto rekurzivna funkcija. Ključnu osobinu Gedelove funkcije izražava sledeća

Lema 2.34 *Za proizvoljne $n \in \mathbb{N}$ i $a_0, a_1, \ldots, a_n \in \mathbb{N}$ sistem jednačina*

$$\begin{aligned} \Gamma(x, 0) &= a_0, \\ \Gamma(x, 1) &= a_1, \\ &\vdots \\ \Gamma(x, n) &= a_n, \end{aligned}$$

ima bar jedno rešenje x.

Dokaz Leme 2.34. Potražićemo rešenje u obliku $x = c(u, b)$, gde su u, b nove nepoznate. Tada tražimo rešenje sistema

$$\mathrm{rest}(u, 1 + (k + 1)b) = a_k,$$

$0 \leqslant k \leqslant n$. Drugim rečima, moraju biti zadovoljene nejednakosti $a_k < 1 + (k+1)b$, i $u - a_k$ mora biti deljivo sa $m_k = 1 + (k+1)b$. Ako odaberemo

$$b = (\max(n, a_0, \ldots, a_n))! \,,$$

tada će tražene nejednakosti očigledno biti zadovoljene. Osim toga, brojevi $m_0, m_1, \ldots, m_k$ će biti uzajamno prosti. U suprotnom, ako bi $p \mid (m_i, m_j)$ $(i \neq j)$, tada bismo iz

$$m_i - m_j = (i-j)b$$

i činjenice da $(i-j) \mid b$ (pošto je $|i-j| \leqslant n$) imali da $p \mid b$. Međutim, iz $m_i = 1+(i+1)b$ je očigledno da su m_i i b uzajamno prosti, odakle zaključujemo da je $(m_i, m_j) = 1$. Sada u treba da nađemo na osnovu uslova $m_k \mid (u - a_k)$, tj. sistema kongruencija

$$u \equiv a_k (\mathrm{mod}\ m_k),$$

$0 \leqslant k \leqslant n$. Međutim, pošto smo upravo videli da su moduli $m_0, \ldots, m_n$ po parovima uzajamno prosti, egzistencija rešenja ovog sistema sledi po Kinsekoj teoremi o ostacima. $\Diamond$

Ako sada u $(*)$ uvrstimo $\Gamma(w,i)$ umesto t_i za sve $0 \leqslant i \leqslant b \dot{-} 1$, dobićemo jednačinu

$$\sum_{j=1}^{n} |\gamma_j(\Gamma(w,0)) - a_j| + |\gamma_{n+1}(\Gamma(w,0)) - \delta_{n+2}(\Gamma(w,1))| +$$

$$\sum_{i=1}^{b\dot{-}1} \left(\sum_{j=1}^{n} |\delta_j(\Gamma(w,i)) - a_j| + |\delta_{n+1}(\Gamma(w,i)) - i| + \right.$$

$$\left. + |\delta_{n+3}(\Gamma(w,i)) - \delta_{n+2}(\Gamma(w,i+1))| \right) + |\delta_{n+3}(\Gamma(w, b \dot{-} 1)) - c| = 0,$$

čija je leva strana prosto rekurzivna funkcija $G(a_1, \ldots, a_n, b, c, w)$. Ako $(*)$ ima rešenje po $t_0, t_1, \ldots, t_{b\dot{-}1}$, tada po prethodnoj lemi postoji rešenje sistema $\Gamma(w,i) = t_i$, $0 \leqslant i \leqslant b \dot{-} 1$, pa je to rešenje w istovremeno i rešenje gornje jednačine. Obratno, ako gornja jednačina ima rešenje po w (za date parametre $a_1, \ldots, a_n, b, c$), tada brojevi $t_i = \Gamma(w,i)$, $0 \leqslant i \leqslant b \dot{-} 1$, rešavaju $(*)$. Lema 2.31 povlači da je f_2 rekurzivno nabrojiv skup, jer važi $(a_1, \ldots, a_n, b, c) \in f_2$ ako i samo ako $G(a_1, \ldots, a_n, b, c, w) = 0$ ima rešenje po w.

Dokaz za (C). Pretpostavljamo da je parcijalna funkcija f dobijena primenom operatora minimalizacije na sledeći način:

$$f(x_1, \ldots, x_n) = \mu_y(g(x_1, \ldots, x_n, y) = 0),$$

gde je g rekurzivno nabrojiv skup, tj. važi

$$g = \{(\xi_1(t), \ldots, \xi_{n+2}(t)) : \ t \in \mathbb{N}\}$$

za neke prosto rekurzivne funkcije $\xi_1, \ldots, \xi_{n+2}$. Formula $(a_1, \ldots, a_n, b) \in f$ ekvivalentna je sa egzistencijom prirodnih brojeva $c_0, c_1, \ldots, c_b$ takvih da je $g(a_1, \ldots, a_n, i) = c_i$, $0 \leqslant i \leqslant b$, pri čemu je $c_i \neq 0$ za $i < b$ i $c_b = 0$. Drugim rečima, ekvivalentan uslov je postojanje brojeva $t_0, t_1, \ldots, t_b$ takvih da za sve $0 \leqslant j \leqslant b$ važi

$$\xi_1(t_j) = a_1, \ \ \ldots, \ \ \xi_n(t_j) = a_n, \ \ \xi_{n+1}(t_j) = j,$$

kao i $\xi_{n+2}(t_j) \neq 0$ za $0 \leqslant j < b$ i $\xi_{n+2}(t_b) = 0$. Ispunjenost ovih uslova je dalje ekvivalentna rešivosti jednačine

$$\sum_{i=1}^{n} \left(\sum_{j=0}^{b} |\xi_i(t_j) - a_i| + |\xi_{n+1}(t_j) - j| \right) +$$

$$+ b \cdot \overline{\mathrm{sg}} \prod_{j=0}^{b \dot{-} 1} \xi_{n+2}(t_j) + \xi_{n+2}(t_b) = 0$$

(koeficijent b u prvom sabirku u drugom redu služi da obezbedi da se uslovi tipa $\xi_{n+2}(t_j) \neq 0$ proveravaju isključivo ako je $b > 0$). Ako sada, slično kao i u delu (B), zamenimo svako pojavljivanje t_j sa $\Gamma(u, j)$, "promovisaćemo" indekse u promenljive, čime leva strana gornje jednačine postaje prosto rekurzivna funkcija $H(a_1, \ldots, a_n, b, u)$, tako da je $(a_1, \ldots, a_n, b) \in f$ ako i samo ako $H(a_1, \ldots, a_n, b, u) = 0$ ima rešenje po u. Dakle, f je rekurzivno nabrojiv skup. □

Posledica 2.35 *Oblast definisanosti i skup vrednosti svake parcijalno rekurzivne funkcije su rekurzivno nabrojivi skupovi.*

Dokaz. Neka je $f : \mathbb{N}^n \to \mathbb{N}$ parcijalno rekurzivna funkcija. Po Lemi 2.32, tada postoje unarne prosto rekurzivne funkcije $\varphi_1, \ldots, \varphi_n, \varphi_{n+1}$ takve da je

$$f = \{(\varphi_1(t), \ldots, \varphi_n(t), \varphi_{n+1}(t)) : \ t \in \mathbb{N}\}.$$

Oblast definisanosti f je tada $\{(\varphi_1(t), \ldots, \varphi_n(t)) : t \in \mathbb{N}\}$, dok se skup vrednosti f poklapa sa $\{\varphi_{n+1}(t) : t \in \mathbb{N}\}$. Prvi od ova dva skupa je rekurzivno nabrojiv po Lemi 2.32, a drugi po Teoremi 2.28. □

Dokaz Teoreme 2.29. Neka je $A \subseteq \mathbb{N}$ rekurzivan skup. Posmatrajmo prosto rekurzivnu funkciju α datu sa

$$\alpha(x) = \sum_{i=0}^{x} \overline{\mathrm{sg}}\, \chi_A(x)$$

koja "broji" koliko ima elemenata skupa A koji su $\leqslant x$. Koristeći ovu funkciju, dobijamo (strogo rastuću) unarnu rekurzivnu funkciju

$$a(x) = \mu_y(|\alpha(y) - (x+1)| = 0),$$

čiji je skup vrednosti baš A. Po prethodnoj posledici, A je rekurzivno nabrojiv skup. □

U Odeljku 2.4 smo najavili rezultat po kome se svuda definisane parcijalno rekurzivne funkcije poklapaju sa rekurzivnim funkcijama.

Dokaz Teoreme 2.15. Neka je $f : \mathbb{N}^n \to \mathbb{N}$ svuda definisana parcijalno rekurzivna funkcija. Tada je $f = \{(\varphi_1(t), \ldots, \varphi_{n+1}(t)) : t \in \mathbb{N}\}$ za neke prosto rekurzivne funkcije $\varphi_1, \ldots, \varphi_{n+1}$. Pošto je f svuda definisana, za sve $x_1, \ldots, x_n \in \mathbb{N}$ postoji rešenje jednačine

$$|\varphi_1(t) - x_1| + \ldots + |\varphi_n(t) - x_n| = 0.$$

Zbog toga se na levu stranu gornje jednakosti može primeniti ograničena minimalizacija (po t). Kako je

$$f(x_1, \ldots, x_n) = \varphi_{n+1}(\mu_t(|\varphi_1(t) - x_1| + \ldots + |\varphi_n(t) - x_n| = 0)),$$

sledi da je f rekurzivna funkcija. □

Iz gornjeg dokaza, odnosno iz implikacije ($\Leftarrow$) u dokazu Teoreme 2.33, neposredno uočavamo da važi i

Posledica 2.36 *Svaka (parcijalno) rekurzivna funkcija se može dobiti od osnovnih funkcija konačnom primenom kompozicija, šema prostih rekurzija i najviše jednog operatora minimalizacije.*

3

Tjuringove mašine

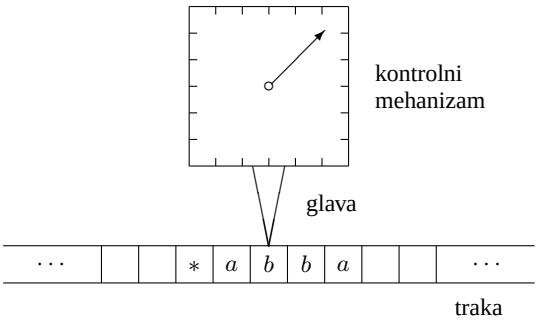
Sa stanovišta pitanja egzistenicje algoritma koji rešava određeni problem, teorija (parcijalno) rekurzivnih funkcija predstavlja potpuno zadovoljavajući formalni ambijent. Ipak, ta teorija ima značajnih nedostataka ako želimo da na sveobuhvatan način izučavamo pojam algoritma. Parcijalno rekurzivne funkcije osvetljavaju uzročno-posledični odnos između ulaznih i izlaznih podataka, ali, istovremeno, kažu veoma malo o jednom drugom iskustvenom fenomenu: *postupku* izračunavanja. Kao što je rečeno u uvodu, algoritam shvatamo kao niz jasno definisanih *koraka*, pri čemu pod korakom podrazumevamo izvođenje neke elementarne operacije na trenutno raspoloživim podacima. Zbog toga je veoma značajno da ponudimo neki matematički model procesa, "mehanizma" koji implementira računanje razmatrane funkcije. Drugim rečima, potreban nam je apstraktni model *računske mašine*. Definisanje pojma izračunljivosti preko računskog modela neposredno omogućava da se precizira šta je to korak algoritma, bez čega je nemoguće da govorimo o merama njegove složenosti (eng. *complexity*) — osnovnog objekta izučavanja u analizi algoritama.

Polazeći upravo od ovakvih razmišljanja, engleski matematičar, osnivač računarstva i tvorac teorije veštačke inteligencije Alan M. Tjuring (1912–1954) je sredinom tridesetih godina prošlog veka pažljivo i detaljno analizirao postupke računanja, kako u svakodnevnom životu ljudi, tako i na do tada poznatim mehaničkim mašinama za računanje. Tako je 1936. godine nastao opšti matematički model računara koji danas nazivamo *Tjuringova mašina* (skraćeno TM).

3.1 Osnovni model Tjuringove mašine

Dizajn "hardvera" osnovnog modela TM je krajnje jednostavan. On se, najpre, sastoji od *kontrolnog mehanizma* koji može da bude u jednom od konačnog broja *stanja*. Ta stanja se menjaju u zavisnosti od samog toka rada mašine. Kontrolni mehanizam je u vezi sa *glavom*, koja operiše trećim delom mašine, *trakom*. Traka je beskonačna na obe strane, i izdeljena je na polja (tako da, u suštini, ima strukturu skupa $\mathbb{Z}$). U svakom polju trake se nalazi neki simbol iz unapred zadate *azbuke* u kojoj data TM radi, s tim da u svakom trenutku skoro sva polja (tj. sva sem konačno mnogo njih) sadrže simbol *blanko*, prazan simbol. Glava u svakom trenutku skenira jedno polje trake — ona je sposobna da čita simbol na tom polju (i tako eventualno izazove promenu stanja kontrolnog mehanizma), ili da taj simbol obriše i umesto njega napiše novi. Takođe, glava može da se pomeri na traci za jedno mesto udesno ili ulevo. Rad TM je određen njenim *programom*: on, u zavsnosti od trenutnog stanja i simbola koji glava čita, definiše jednu od opisanih akcija glave na traci i naredno stanje kontrolnog mehanizma. Osim opisanih koraka, program može za neku kombinaciju stanje-simbol i da naloži obustavljanje rada mašine ("halt").

Ovu mentalnu predstavu, opisani intuitivni model TM, najbolje ilustruje sledeća slika.



Sada možemo preći i na formalnu definiciju TM. *Tjuringova mašina* je uređena sedmorka

$$\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \top, \bot),$$

gde je Q konačni *skup stanja* (kontrolnog mehanizma), Σ je *ulazna azbuka* (skup simbola pomoću kojih su zapisani ulazni podaci), $\Gamma \supset \Sigma$ je *azbuka trake* (koja osim ulaznih simbola sadrži i druge pomoćne simbole koje $\mathcal{M}$ koristi tokom rada, a obavezno simbol $*$ za blanko), $q_0 \in Q$ je *početno stanje*, dok

su $\top, \bot \in Q$ redom stanja *prihvatanja* i *odbijanja* (koja omogućavaju da TM koristimo za rešavanje problema odlučivanja). Najzad, "suština" svake TM je njen program, *funkcija prelaza*

$$\delta : Q \times \Gamma \to (\Gamma \cup \{L, R, H\}) \times Q$$

gde, naravno, simboli L, R, H (koji redom predstavljaju komande 'pomeri glavu levo', 'pomeri glavu desno' i 'stani') ne pripadaju nijednom od gore definisanih skupova. Ova funkcija za svaki par stanje-simbol određuje sledeću akciju na traci i naredno stanje mašine. Tako $\delta(q, a) = (\alpha, q')$ (što ponekad pišemo prosto kao četvorku $q\ a\ \alpha\ q'$) u našem intuitivnom modelu tumačimo kao komandu 'ako si u stanju q, a glava čita na traci simbol a, uradi α i pređi u stanje q''. Pri tome, ako je $\alpha \in \Gamma$, to tumačimo kao akciju brisanja simbola u polju trake kojeg skenira glava i upisivanja α umesto njega, $\alpha = L$ znači pomeranje glave jedno polje ulevo, $\alpha = R$ pomeranje glave jedno polje udesno, a $\alpha = H$ zaustavljanje, prekid rada mašine. Uobičajeno je (iako ne i obavezno) da bude $\delta(q, a) = (H, q)$ za $q \in \{\top, \bot\}$.

Kako bismo pratili rad TM, uvodimo pojam *(trenutne) konfiguracije*. Konfiguracija je uređena trojka $(w_1, q, w_2) \in \Gamma^* \times Q \times \Gamma^*$. Nju tumačimo tako da je u datom trenutku mašina u stanju q, da skenira prvi karakter reči w_2, te da su w_1, w_2 reči na traci redom levo, odnosno desno od pozicije glave. Pri tome, ove reči reprezentuju samo "interesantni" deo trake, dok je ostatak popunjen praznim simbolima $*$. Dakle, ako je $w_1 = a_1 \dots a_n$ i $w_2 = b_1 \dots b_m$, tada je sadržaj trake

$$\cdots * * * a_1 \dots a_n \underline{b_1} b_2 \dots b_m * * * \cdots$$

(podvučeni karakter označava poziciju glave). Uvodimo relaciju $\vdash$ među konfiguracijama tako da je $c_1 \vdash c_2$ za $c_1 = (w_1, q, w_2)$, $w_1 = a_1 \dots a_n$ i $w_2 = b_1 \dots b_m$ (ukoliko je neka od reči w_1, w_2 jednaka λ, uvek joj možemo dodati jedno $*$ ako nam "zafali" u donjim definicijama) ako je:

(1) $c_2 = (w_1, q', b'b_2 \dots b_m)$ u slučaju $\delta(q, b_1) = (b', q')$, $b' \in \Gamma$,

(2) $c_2 = (a_1 \dots a_{n-1}, q', a_n w_2)$ u slučaju $\delta(q, b_1) = (L, q')$,

(3) $c_2 = (w_1 b_1, q', b_2 \dots b_m)$ u slučaju $\delta(q, b_1) = (R, q')$,

(4) u slučaju $\delta(q, b_1) = (H, q')$ takvo c_2 ne postoji.

Relacija $\vdash^*$ predstavlja refleksivno-tranzitivno zatvorenje od $\vdash$: važi $c \vdash^* c'$ ako i samo ako je $c = c'$ ili postoji $n \in \mathbb{N}$, $n \geqslant 1$, tako da je $c = c_0$, $c' = c_n$ i $c_i \vdash c_{i+1}$ za sve $0 \leqslant i < n$.

Tjuringove mašine koristimo u dva režima rada. Prvi od tih režima podrazumeva da pomoću TM rešavamo probleme odlučivanja. Pri tome, domen problema je oblika Σ^* (Σ će tada biti ulazna azbuka mašine), pa se posmatrani problem identifikuje sa nekim jezikom $L \subseteq \Sigma^*$.

U ovom pristupu, *početna konfiguracija* u kojoj mašina uvek započinje rad je trojka (w, q_0, λ), gde je $w \in \Sigma^*$ *ulazna reč*. *Jezik* TM $\mathcal{M}$ jeste skup svih ulaznih reči koje nakon konačno mnogo koraka $\mathcal{M}$ prevode u prihvatno stanje:

$$L(\mathcal{M}) = \{w \in \Sigma^* : (w, q_0, \lambda) \vdash^* (w_1, \top, w_2) \text{ za neke } w_1, w_2 \in \Gamma^*\}.$$

Za mašine $\mathcal{M}_1$ i $\mathcal{M}_2$ kažemo da su *ekvivalentne* ako je $L(\mathcal{M}_1) = L(\mathcal{M}_2)$.

Prvi ključni rezultat koji povezuje Tjurinove mašine i teoriju rekurzivnih funkcija je sledeći.

Teorema 3.1 *Neka je $\Sigma \neq \varnothing$ konačna azbuka i $L \subseteq \Sigma^*$. Postoji Tjuringova mašina $\mathcal{M}$ tako da je $L = L(\mathcal{M})$ ako i samo ako je L rekurzivno nabrojiv jezik.*

Skrećemo pažnju da su za datu mašinu $\mathcal{M}$ i ulaznu reč w u načelu moguća tri ishoda:

(1) $\mathcal{M}$ se zaustavlja u prihvatnom stanju $\top$ (reč w je *prihvaćena*),

(2) $\mathcal{M}$ se zaustavlja u odbijajućem stanju $\perp$ ili nekom drugom stanju $\neq \top$ (reč w je *odbijena*),

(3) $\mathcal{M}$ nikada ne ulazi ni u jedno od stanja $\top, \perp$, već ulazi u "mrtvu petlju" (komanda H se nikada ne realizuje, već se rad $\mathcal{M}$ nastavlja u beskonačnost).

Zbog mogućnosti (3), tj. zbog činjenice da $\mathcal{M}$ za ulaznu reč $w \notin L(\mathcal{M})$ u opštem slučaju može da prati bilo koji od "scenarija" (2), (3), ne možemo proizvoljnu mašinu $\mathcal{M}$ identifikovati sa algoritmom koji rešava problem odlučivanja modeliran sa $L(\mathcal{M})$. Zbog toga ćemo posebno izdvojiti tzv. *totalne TM*, kod kojih svaka ulazna reč proizvodi jednu od situacija (1), (2). One predstavljaju pravi apstraktni model algoritamski rešivog problema, budući da važi

Teorema 3.2 *Neka je $\Sigma \neq \varnothing$ konačna azbuka i $L \subseteq \Sigma^*$. Postoji totalna Tjuringova mašina $\mathcal{M}$ tako da je $L = L(\mathcal{M})$ ako i samo ako je L rekurzivan jezik.*

S druge strane, Tjuringove mašine možemo koristiti i za opštiji zadatak izračunavanja (parcijalnih) funkcija. Slično kao i malopre, polazimo od pretpostavke da se ulazni i izlazni podaci (vrednosti promenljivih i vrednost funkcije

koja se računa) zapisuju u nekoj azbuci Σ, tako da su parcijalne funkcije koje posmatramo oblika $(\Sigma^*)^n \to \Sigma^*$, $n \in \mathbb{N}$ (što uključuje i aritmetičke funkcije, identifikacijom prirodnog broja sa njegovim zapisom u odabranom sistemu).

Kažemo da TM $\mathcal{M}$ *izračunava vrednosti parcijalne funkcije* $f : (\Sigma^*)^n \to \Sigma^*$ ako se $\mathcal{M}$, polazeći od konfiguracije

$$(w_1 * \ldots * w_n, q_0, \lambda),$$

zaustavlja nakon konačno mnogo koraka za sve $w_1, \ldots, w_n \in \Sigma^*$ za koje je $f(w_1, \ldots, w_n)$ definisano, i to sa završnom konfiguracijom

$$(u * f(w_1, \ldots, w_n), q, \lambda)$$

za neku reč $u \in \Gamma^*$ i neko $q \in Q$, dok se za ostale vrednosti $w_1, \ldots, w_n$ mašina $\mathcal{M}$ se ne zaustavlja (ishod (3)).

Dakle, za razliku od primene TM kao sistema za rešavanje problema odlučivanja, gde se komunikacija sa "spoljnim svetom" odvija preko stanja kontrolnog mehanizma u trenutku zaustavljanja mašine, ovde se izlazni podatak (rezultat rada mašine $\mathcal{M}$) čita sa same trake: dovoljno je pogledati prvu reč koja se nalazi levo od položaja glave.

Ispostavlja se da se rekurzivno izračunljive i Tjuring-izračunljive parcijalne funkcije poklapaju. To su upravo funkcije koje na osnovu Čerčove teze izjednačavamo sa intuitivnim pojmom algoritamski rešivog problema.

Teorema 3.3 *Neka je $\Sigma \neq \varnothing$ konačna azbuka. Postoji Tjuringova mašina koja izračunava parcijalnu funkciju $f : (\Sigma^*)^n \to \Sigma^*$ ako i samo ako je njoj pridružena aritmetička parcijalna funkcija $||f|| : \mathbb{N}^n \to \mathbb{N}$, određena uslovom*

$$||f||(||w_1||, \ldots, ||w_n||) = ||f(w_1, \ldots, w_n)||$$

za sve $w_1, \ldots, w_n \in \Sigma^*$, parcijalno rekurzivna.

Kada se pominje *ekvivalentnost* formalizama Tjuringovih mašina i rekurzivne izračunljivosti, pod time se zapravo podrazumeva ekvivalencija upravo u smislu Teorema 3.1, 3.2 i 3.3. Ove teoreme predstavljaju rezime, "kombinaciju" glavnih rezultata sadrženih u radovima Čerča [5] i Tjuringa [47] (videti takođe i [14, 27]).

Primer 3.4 Jedan od tehnički najjednostavnijih (iako sa stanovišta potrošnje polja trake ne baš najekonomičnijih) načina na koji možemo koristiti TM za

izračunavanje aritmetičkih funkcija jeste da prirodne brojeve predstavimo u unarnom sistemu, tj. da radimo sa ulaznom azbukom koja sadrži jedan jedini simbol, *recku*: $\Sigma = \{|\}$. Prirodan broj n tada odgovara nizu od $n+1$ uzastopne recke, $|^{n+1}$ (tako, na primer, jedna recka reprezentuje nulu). Ovako prikazani brojevni podaci se razdvajaju blanko simbolima $*$. U smislu malopređašnjih opštih definicija, kažemo da TM $\mathcal{M}$ izračunava funkciju $f : \mathbb{N}^n \to \mathbb{N}$ ako se za sve $x_1, \ldots, x_n \in \mathbb{N}$ mašina $\mathcal{M}$, polazeći od konfiguracije

$$\underbrace{|||\ldots||}_{x_1+1} * \cdots * \underbrace{|||\ldots||}_{x_n+1} \underline{*} \ ,$$

zaustavlja sa sledećom konfiguracijom:

$$\underbrace{|||\ldots||}_{x_1+1} * \cdots * \underbrace{|||\ldots||}_{x_n+1} * \underbrace{|||||||\ldots|||||}_{f(x_1,\ldots,x_n)+1} \underline{*} \ .$$

Prema Teoremi 3.3, (parcijalne) funkcije koje se na ovaj način mogu izračunati su tačno (parcijalno) rekurzivne funkcije (vidi [14]). □

Zadaci.

1. Dizajnirati totalne TM koje prihvataju sledeće jezike nad ulaznom azbukom $\Sigma = \{0, 1\}$:

(a) $\{0^n 1^n : \ n \in \mathbb{N}\}$,

(b) skup svih reči koje imaju isti broj pojavljivanja simbola 0 i 1,

(c) skup svih palindroma nad Σ.

2. Neka je $n \geqslant 1$. Konstruisati TM u "recka-sistemu" koja desno od n unetih brojeva kopira prvi od njih (posmatrano sleva).

3. Konstruisati TM u "recka-sistemu" koje izračunavaju

(a) zbir

(b) proizvod

data dva broja.

4. Konstruisati TM u "recka-sistemu" koje izračunavaju funkcije $x \dot{-} y$ i $|x - y|$.

5. Konstruisati TM u "recka-sistemu" koja izračunava parcijalnu funkciju $\left\lfloor \frac{x}{y} \right\rfloor$.

3.2 Vremenska i prostorna složenost. Klase složenosti

Nakon definicija iznetih u prethodnom odeljku, prilično je jasno šta podrazumevamo pod korakom algoritima — to je izvršenje jedne komande TM koja implementira odgovarajući problem. Broj koraka koji je potreban da bi mašina završila svoj rad (i tako došla do rezultata izračunavanja) zavisi od ulaznog podatka, te se možemo zapitati na koji način zavisi taj broj koraka od veličine ulaza[10]. Jasno je da je daleko manji broj koraka potreban da bi se za reč dužine 3 utvrdilo da li je palindrom (dovoljno je uporediti prvi i treći simbol), nego što je to slučaj sa rečima dužine 3000. Takođe, broj koraka potreban za sabiranje dva broja zavisi od toga koliko dati brojevi imaju cifara. Ovakva i slična razmatranja vode ka konceptu *vremenske složenosti* algoritma.

S druge strane, možemo meriti i broj polja trake koja data TM koristi tokom određenog izračunavanja (polja koja zauzima ulazni podatak se ne računaju, već samo polja u koje mašina piše tokom rada). Drugim rečima, procenjujemo potrošnju memorije TM od strane razmatranog programa. Ova mera naziva se *prostorna složenost* algoritma.

Formalnije, imamo sledeće definicije. Neka je $\mathcal{M}$ totalna TM. Za funkciju $T_{\mathcal{M}} : \mathbb{N} \to \mathbb{N}$ kažemo da je *ocena vremenske složenosti mašine* $\mathcal{M}$, ukoliko se $\mathcal{M}$, za svaku ulaznu reč w, $|w| \leqslant n$ ($n \in \mathbb{N}$), zaustavlja nakon ne više od $T_{\mathcal{M}}(n)$ koraka. Analogno, funkcija $S_{\mathcal{M}} : \mathbb{N} \to \mathbb{N}$ je *ocena prostorne složenosti mašine* $\mathcal{M}$ ako $\mathcal{M}$ za svaku ulaznu reč w, $|w| \leqslant n$ ($n \in \mathbb{N}$), tokom svog rada piše simbole na ne više od $S_{\mathcal{M}}(n)$ polja trake.

Prilično je jasno da su ove definicije univerzalne u tom smislu da se one mogu bez ikakvih poteškoća primeniti ne samo na (osnovni model) TM, već i na svaki drugi model računske mašine. Pri tome je potrebno precizirati samo dve stvari:

- koja operacija se smatra elementarnim korakom posmatrane mašine,
- šta je osnovna memorijska jedinica te mašine.

Ono što odmah upada u oči u gornjim definicijama jeste njihova neprecizност, ili bolje rečeno "ležernost". One uopšte ne definišu *jedinstvenu* funkciju složenosti mašine. To potiče od opšteg pristupa koji se u načelu prihvata u teoriji složenosti algoritama. Naime, nama će biti od vrlo male koristi da znamo da je *tačna* složenost neke mašine npr. $15n^2 - 6n + 1979$. Umesto toga, mi jedino pokušavamo da dobijemo *ocenu* složenosti i da pri tome pronađemo

[10] Naravno, ovo pitanje ima smisla samo za totalne TM.

što je moguće bolju. Drugim rečima, najviše nas zanima da procenimo *red veličine* razmatrane složenosti, a ne da dobijemo tačnu funkciju (koju je u najvećem broju slučajeva praktično nemoguće odrediti). Tako ćemo u pomenutom primeru reći da je reč o algoritmu *kvadratne složenosti*: zaista, navedena funkcija je $\mathcal{O}(n^2)$. Podsetimo, za dve realne funkcije $f, g : \mathbb{R} \to \mathbb{R}$ pišemo da je $f = \mathcal{O}(g)$ ako postoji konstanta $C > 0$ i $x_0 \in \mathbb{R}$ tako da za sve $x \geqslant x_0$ važi nejednakost

$$f(x) \leqslant Cg(x).$$

Kažemo i da funkcija $g(x)$ *asimptotski ograničava* funkciju $f(x)$. Slična notacija se može definisati i za celobrojne funkcije.

Ovakav asimptotski pristup otvara put ka definisanju *klasa složenosti*. Naš prvenstveni zadatak jeste klasifikacija problema odlučivanja, kao koncepcijski najjednostavnijih (pošto se u njima vrši izračunavanje najmanje jedinice informacije — jednog bita). Zbog toga, razvrstavamo jezike totalnih TM (tj. rekurzivne jezike) u sledeće *vremenske i prostorne klase* (gde je $f : \mathbb{N} \to \mathbb{N}$ data funkcija):

$$TIME(f(n)) = \{L : \text{ postoji } \mathcal{M} \text{ tako da } L(\mathcal{M}) = L,\ T_{\mathcal{M}}(n) = \mathcal{O}(f(n))\},$$

odnosno

$$SPACE(f(n)) = \{L : \text{ postoji } \mathcal{M} \text{ tako da } L(\mathcal{M}) = L,\ S_{\mathcal{M}}(n) = \mathcal{O}(f(n))\}.$$

Na sličan način je moguće definisati i klase rekurzivnih funkcija u odnosu na složenost mašina koje ih računaju.

Od interesa za teoriju je da se od nekih od ovih klasa prave dalji agregati. Tako definišemo klasu

$$\mathbf{P} = \bigcup_{k \geqslant 0} TIME(n^k).$$

Reč je o tzv. *polinomnim algoritmima (odlučivanja)*, klasi jezika odlučivih u vremenu koje se može oceniti polinomnom funkcijom po veličini ulazne reči. Sa stanovišta vremena, ovo su algoritmi koji se smatraju "dobrim", vremenski *efikasnim*. Ostali jezici su "loši", "neobradivi" (eng. *intractable*), jer se smatra da njihovo odlučivanje traje sa praktičnog stanovišta predugo[11].

[11] Međutim, mnogi programi koji se nalaze čak i u komercijalnoj upotrebi nisu polinomni, već troše eksponencijalno vreme, ali funkcionišu zahvaljujući pretpostavljenoj umerenosti veličine ulaznih podataka i velikoj brzini savremenih računara. Tako, reč *intractable* treba uzeti sa dozom rezerve.

Znatno šira klasa

$$\mathbf{EXP} = \bigcup_{a>1} \bigcup_{k\geqslant 0} TIME(a^{n^k})$$

sastoji se iz *jednostruko eksponencijalnih algoritama*.

Slično definišemo i neke interesantne prostorne klase. Klasa prostorno "najboljih" algoritama odlučivanja je:

$$\mathbf{L} = SPACE(\log n).$$

(Primetimo da se osnova logaritma u svakom asimptotskom razmatranju može izostaviti, budući da za sve $a, b > 1$ važi $\log_a x = \mathcal{O}(\log_b x)$.) Takođe, od interesa je i klasa *prostorno polinomnih algoritama*,

$$\mathbf{PSPACE} = \bigcup_{k\geqslant 0} SPACE(n^k),$$

koja je, iako šira (i stoga "slabija") od $\mathbf{L}$, takođe prilično zanimljiva kako sa teorijskog, tako i praktičnog stanovišta.

Zadaci.

1. Oceniti vremensku složenost mašina konstruisanih u Zadatku 3.1.1.

3.3 Višetračne Tjuringove mašine

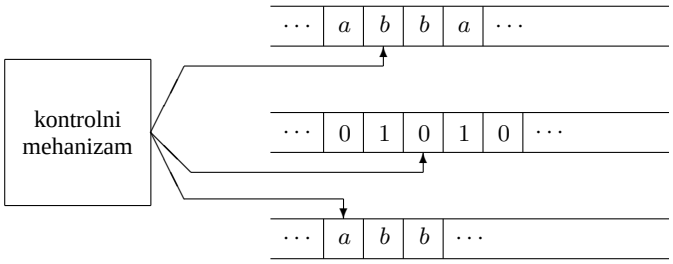
Sa stanovišta analize složenosti algoritama, glavni problem sa osnovnim modelom TM jeste to što je on sa tehničkog stanovišta suviše "primitivan", udaljen od načina na koji savremani računar operiše sa podacima. Razlog za to je loša struktura memorije: traka. Njena linearna uređenost uslovljava da TM može u svakom trenutku svog rada neposredno da pristupi samo jednom simbolu, dok su za pristupanje drugim simbolima potrebne dodatne komande koje će pomeriti glavu na odgovarajuće mesto. Kako bi se prevazišli ovi problemi, pribeglo se usavršavanju osnovnog dizajna TM. Ispostavilo se da je računska moć ovih usavršenih modela ista kao i kod TM (što je bio još jedan dodatni argument u prilog Čerč-Tjuringove teze), a da je razlika u brzini, tj. složenosti mašine koja implementira razmatrani problem.

Kao jedan od značajnih koraka u tom pravcu, izdvajaju se *višetračne Tjuringove mašine* (VTM). Modifikacija se ovde sastoji u tome što umesto jedne,

mašina $\mathcal{M}$ ima veći (ali fiksan) broj traka $k \geqslant 2$. Tada funkcija prelaza ima oblik

$$\delta : Q \times \Gamma^k \to ((\Gamma \cup \{L, R\})^k \cup \{H\}) \times Q.$$

Dakle, jedan korak VTM se sastoji iz k koraka obične TM, po jedan na svakoj traci nezavisno, s tim da je zaustavljanje mašine jedinstvena akcija za sve trake.



Kao što je to objašnjeno u prethodnom odeljku, sada možemo definisati pojam složenosti za VTM analogno kao za TM. Naravno, jedan isti problem može imati različitu složenost na običnoj TM i VTM. To dolazi do izražaja u sledećem primeru.

Primer 3.5 Razmotrimo još jednom problem prepoznavanja palindroma na datoj azbuci sa bar dva simbola (vidi zadatke 3.1.1 (c) i 3.2.1). Nije teško pokazati da ovaj problem zahteva kvadratno vreme na običnoj TM. Naime, odgovarajuća TM mora tokom svog rada da uporedi prvo i poslednje, drugo i pretposlednje, itd. slovo ulazne reči. Ako je n dužina ulazne reči, tada posmatrana mašina mora da (radi opisanih poređenja) načini bar

$$(n-1) + (n-3) + \ldots + \mathrm{rest}(n-1, 2)$$

pokreta glave (ulevo ili udesno). Gornji zbir iznosi $\frac{n^2}{4}$ ako je n parno, a $\frac{n^2-1}{4}$ ako je n neparno.

S druge strane, postoji očigledan algoritam koji radi isti posao na 2-tračnoj TM. Naime, polazeći od konfiguracije gde je ulazna reč na prvoj traci i prva glava je desno od nje, najpre treba iskopirati w na drugu traku čitajući je unazad, zdesna nalevo, zatim vratiti drugu glavu na levi kraj iskopirane reči, i najzad uporediti reči na dvema trakama, karakter po karakter. Ovaj algoritam očigledno radi u vremenu $4n - 1$ ($2n$ koraka za kopiranje, $n - 1$ korak za vraćanje druge glave na prvi karakter sleva kopirane reči i n koraka za poređenje ulazne i "invertovane" reči), pa je reč o algoritmu koji radi u *linearnom* vremenu, gde je $T(n) = \mathcal{O}(n)$. □

Međutim, ispostavlja se da, iako je sa VTM moguće postići bolje rezultate po pitanju složenosti, to poboljšanje ipak nije drastično, dok po pitanju računske moći poboljšanja uopšte i nema.

Teorema 3.6 *Neka je $\mathcal{M}$ k-tračna TM. Tada postoji (obična) TM $\mathcal{M}'$ koja simulira $\mathcal{M}$. Tačnije, mašine $\mathcal{M}$ i $\mathcal{M}'$ su ekvivalentne ($L(\mathcal{M}') = L(\mathcal{M})$), a ukoliko $\mathcal{M}$ izračunava neku parcijalnu funkciju $f_{\mathcal{M}}$, tada $\mathcal{M}'$ izračunava istu tu funkciju. Pri tome, ako je $t : \mathbb{N} \to \mathbb{N}$ funkcija takva da je $T_{\mathcal{M}}(n) = \mathcal{O}(t(n))$, tada je $T_{\mathcal{M}'}(n) = \mathcal{O}([t(n)]^2)$.*

Skica dokaza. Osnovna ideja simulacije se sastoji u tome što $\mathcal{M}'$ na svojoj jedinoj traci čuva niz od k reči u azbuci mašine $\mathcal{M}$ razdvojenih posebnim simbolom $\#$ — sekvencijalni zapis sadržaja traka mašine $\mathcal{M}$. U nekoliko prolaza kroz ovaj zapis, $\mathcal{M}'$ ga ažurira tako da se dobije stanje na trakama $\mathcal{M}$ nakon jednog njenog koraka. Naravno, pri tome $\mathcal{M}'$ u svakom trenutku mora na neki način da "pamti" položaje k glava mašine $\mathcal{M}$ na njenim trakama. To se postiže tako što će azbuka trake za $\mathcal{M}'$ biti šira od azbuke Γ mašine $\mathcal{M}$: osim dodatnog separatora $\#$, mi ćemo zapravo "duplirati" azbuku Γ, tako da će azbuka mašine $\mathcal{M}'$ biti $\Gamma \cup \Gamma' \cup \{\#\}$, gde je

$$\Gamma' = \{\hat{a} : \ a \in \Gamma\}.$$

Između svaka dva uzastopna simbola $\#$ tačno jedan simbol će imati "kapicu", i ona će reprezentovati "virtuelnu glavu": $\hat{a}$ simulira situaciju na posmatranoj traci mašine $\mathcal{M}$ u kojoj njena glava čita simbol a upravo na odgovarajućem mestu na toj traci. Tako, na primer, u simulaciji neke 3-tračne TM možemo imati sledeći sadržaj trake $\mathcal{M}'$:

$$\cdots * * * \#a\hat{b}ba\#01\hat{0}10\#\hat{a}bb\# * * * \cdots,$$

što reprezentuje trake mašine $\mathcal{M}$ čiji je sadržaj redom $abba$, 01010 i abb, dok su glave redom na drugom, trećem, odnosno prvom karakteru ovih reči (vidi sliku na prethodnoj strani). Naravno, za očekivati je da će i skup stanja kontrolnog mehanizma mašine $\mathcal{M}'$ biti u izvesnom smislu širi od izvornog skupa stanja mašine $\mathcal{M}'$.

$\mathcal{M}'$ započinje svoj rad sa ulaznom reči $w = a_1 \dots a_n$ mašine $\mathcal{M}$ i glavom na prvom praznom polju desno od nje. Prvi zadatak koji $\mathcal{M}'$ ima je da inicijalizuje svoju traku tako da se dobije simulirani zapis sadržaja traka mašine $\mathcal{M}$ na početku njenog rada. Drugim rečima, $\mathcal{M}'$ treba da (preko ulazne reči) napiše

$$\cdots * \#a_1 \dots a_n\hat{*}\#\hat{*}\# \cdots \#\hat{*}\# * \cdots$$

Sada prelazimo na opis simulacije jednog koraka VTM $\mathcal{M}$ od strane $\mathcal{M}'$. Radi toga, $\mathcal{M}'$ najpre postavlja svoju glavu na prvi simbol $\#$ sleva i pravi jedan prolaz kroz "interesantni" deo trake, sve dok ne stigne do $(k+1)$-vog simbola $\#$, usput pamteći u svom kontrolnom mehanizmu simbole na pozicijama virtuelnih glava. Kako je ovo moguće? Ako sa Q_1 označimo skup internih, "autohtonih" stanja mašine $\mathcal{M}'$, tada za skup stanja kontrolnog mehanizma $\mathcal{M}'$ možemo uzeti

$$Q_1 \times Q \times \Gamma^k \times \mathbb{N}_k,$$

gde je $\mathbb{N}_k = \{0, 1, \ldots, k\}$. U tom slučaju, stanje naše mašine $\mathcal{M}'$ je uređena $(k+3)$-orka

$$(s, q, \alpha_1, \ldots, \alpha_k, i),$$

gde je $s \in Q_1$, $q \in Q$, $\alpha_1, \ldots, \alpha_k$ su simboli iz Γ i $0 \leqslant i \leqslant k$. Pre opisanog prvog čitanja trake $\mathcal{M}'$, mašina je u stanju $(s_1, q, *, *, \ldots, *, 0)$ za neko $s_1 \in Q_1$, gde je q stanje mašine $\mathcal{M}$ pre simuliranog koraka. Prvi potez joj je kretanje nadesno. Svako čitanje simbola $\#$ uvećeva brojač za jedan, tako da mašina $\mathcal{M}'$ "zna" koju simuliranu traku mašine $\mathcal{M}$ trenutno obrađuje. Ako je trenutna vrednost brojača i, a mašina upravo čita simbol $\hat{a} \in \Gamma'$, tada se stanje $\mathcal{M}'$ menja tako što se na $(i+3)$-ćoj komponenti stanja umesto $*$ upisuje a. Zatim se nastavlja kretanje nadesno, sve dok vrednost brojača i ne postane k. Na primer, ako je $\mathcal{M}'$ čitala svoju traku sa sadržajem

$$\cdots * \#a\hat{b}ba\#01\hat{0}10\#\hat{a}bb\# * \cdots,$$

tada će po opisanom prolazu stanje kontrolnog mehanizma biti oblika

$$(s_2, q, b, 0, a, 3)$$

za neko $s_2 \in Q_1$.

Opisani "trik" sa pamćenjem ograničene količine informacije od strane kontrolnog mehanizma TM implementiran je u svakom savremenom računaru: u pitanju su registri procesora.

Ažuriranje trake se vrši u drugom prolazu, zdesna nalevo. Sada čitanje svakog novog simbola $\#$ umanjuje brojač i za jedan, tako da taj brojač u svakom trenutku pokazuje redni broj trake izvorne mašine $\mathcal{M}$ koja se u simulaciji obrađuje. Kad god $\mathcal{M}'$ naiđe na simbol sa "kapicom", simulira akciju koju bi $\mathcal{M}$ izvršila na i-toj traci u stanju q. Dakle, ako je to pisanje simbola b preko a, tada $\mathcal{M}'$ zamenjuje $\hat{a}$ sa $\hat{b}$. Ako je to pomeranje glave nalevo ili nadesno, tada $\mathcal{M}'$ zamenjuje $\hat{a}$ sa a, dok se simbol b levo ili desno od njega zamenjuje sa $\hat{b}$. Zatim

se ovaj posao nastavlja na narednoj traci (ide se nalevo, čeka se prvi simbol $\#$, dok se i umanjuje za jedan).

Naravno, problem nastaje ukoliko se $\mathcal{M}'$, simulirajući kretanje glave $\mathcal{M}$ na nekoj traci nalevo ili nadesno, nađe na polju koje sadrži marker $\#$. Ovo se događa upravo onda kada mašina $\mathcal{M}$ na nekoj od traka po prvi put skenira neko (prazno) polje, koje do tada nije bilo korišćeno. Sada mašina $\mathcal{M}'$ mora da "otvori" prostor za novo polje između dva uzastopna $\#$ i tu (na početku ili kraju razmatrane podreči) upiše simbol $\hat{*}$. Tada glava $\mathcal{M}'$ mora da se (kontrolisana adekvatnim internim stanjem iz Q_1) vrati na jedan od rubova radnog segmenta svoje trake i zatim translira slovo po slovo sadržaj trake za jedno mesto nalevo / nadesno, sve do mesta gde je potrebno novo polje (to mesto se takođe može zapamtiti daljim "struktuiranjem" skupa Q_1). U novo polje se smešta željeni simbol $\hat{*}$ i simulacija se nastavlja na narednom segmentu određenom uzastopnim simbolima $\#$.

Kada $\mathcal{M}'$ stigne do prvog simbola $\#$ sleva, prelazi u stanje oblika

$$(s_3, q', *, \ldots, *, 0)$$

za neko $s_3 \in Q_1$, gde je q' naredno stanje mašine $\mathcal{M}$ po okončanju simuliranog koraka.

Ukoliko je pri tome $q' \in \{\top, \bot\}$, mašina $\mathcal{M}'$ će ući u svoje stanje prihvatanja / odbijanja i završiti rad. Ukoliko mašina $\mathcal{M}$ izračunava neku parcijalnu funkciju, tada ona na kraju svog rada ispisuje rezultat (ako on postoji) na neku od traka i pozicionira glavu desno od njega. U tom slučaju, mašina $\mathcal{M}'$ će kopirati taj podatak (kontrola se ponovo ostvaruje putem stanja iz Q_1) desno od poslednjeg simbola $\#$, postaviti svoju glavu desno od njega i obustaviti rad.

Preostaje da se izvrši procena vremenske složenosti simulacije $\mathcal{M}'$. Najpre, inicijalizacija trake $\mathcal{M}'$ se realizuje u vremenu $\mathcal{O}(n)$, što je istovremeno $\mathcal{O}(t(n))$, pošto funkcija $t(n)$ ne može biti (po uslovima teoreme) asimptotski slabija od linearne. Sama simulacija jednog koraka mašine $\mathcal{M}$ od strane $\mathcal{M}'$ uključuje:

- jedno "čitanje" trenutnog sadržaja trake sleva nadesno — ako je trenutna dužina reči na traci d, onda ova faza troši vreme $\mathcal{O}(d)$;
- ažuriranje sadržaja pojedinih segmenata trake (ograničenih uzastopnim simbolima $\#$), kao i pozicija virtuelnih glava — što zahteva konstantan broj koraka mašine $\mathcal{M}'$ (dakle, troši vreme $\mathcal{O}(1)$);
- u slučaju da je potrebno ranije opisano "proširenje" nekog segmenta, treba

realizovati translaciju nekog dela (prefiksa ili sufksa) reči koja čini radni deo trake — ova rutina očigledno troši vreme $\mathcal{O}(d)$.

Međutim, dužina d reči zapisane na traci mašine $\mathcal{M}'$ jednaka je

$$d_1 + \ldots + d_k + (k+1),$$

gde je d_i dužina reči na i-toj traci mašine $\mathcal{M}$ pre koraka koji se trenutno simulira. Ta dužina d_i ne može biti veća od $t(n)$, pošto svaka TM tokom t koraka može da skenira najviše isto toliko polja trake. Zaključujemo da se jedan korak mašine $\mathcal{M}$ simulira sa $\mathcal{O}(t(n))$ koraka mašine $\mathcal{M}'$. Zbog toga je

$$T_{\mathcal{M}'}(n) = \mathcal{O}(t(n))T_{\mathcal{M}}(n) = \mathcal{O}([t(n)]^2),$$

što je i trebalo dokazati. □

Dakle, simulacija VTM putem običnih TM nas "košta" kvadratnog usporenja rada. To, međutim, znači da se pri prelasku sa TM na VTM *ne menja* svojstvo vremenske polinomnosti algoritma — svi algoritmi koji rade u polinomnom vremenu na VTM radiće u polinomnom vremenu i na TM (jedino što se udvostručava stepen polinoma koji izražava vreme rada, a i koeficijenti mogu biti veći). Specijalno, klasa jezika $\mathbf{P}$ "ne oseća" ovu modifikaciju računskog modela.

Zadaci.

1. Dizajnirati višetračne TM koje odlučuju jezike iz Zadatka 3.1.1 (a) i (b). Proceniti složenost tako dobijenih mašina.

2. Dizajnirati višetračnu TM koja za dva binarna broja data da prvoj traci, razdvojena simbolom $\#$, ispisuje iza njih (dakle, takođe na prvoj traci) još jedan simbol $\#$ i njihov zbir. Analizirati složenost odgovarajućeg algoritma.

3.4 Univerzalna Tjuringova mašina

Konstrukcija koju prezentiramo u ovom odeljku predstavlja jednu od ključnih u teoriji izračunljivosti. Naime, polazeći od Čerč-Tjuringove teze, možemo reći da svaka pojedinačna Tjuringova mašina predstavlja apstraktni model odgovarajućeg algoritma koji se implementira u vidu *programa*. Kako bi izvršenje

programa na računaru bilo moguće, potrebno je odgovarajuće softversko okruženje, tj. poseban program (dakle, Tjuringova mašina) koji će za *svaki* program P napisan po datim sintaksnim pravilima "simulirati" algoritam opisan sa P prosleđivanjem računaru odgovarajućih mašinskih instrukcija i ulaznih podataka čije je učitavanje predviđeno u P. Drugim rečima, potreban nam je *interpretator*. Koncept interpretatora je formalizovan upravo kroz *univerzalnu Tjuringovu mašinu* $\mathcal{U}$.

Grubo govoreći, ideja je u tome da ulazna reč za ovu mašinu bude par oblika

$$(\mathcal{M}, w),$$

gde je $\mathcal{M}$ neka Tjuringova mašina, w reč u njenoj ulaznoj azbuci (pri čemu su svi ovi podaci kodirani u nekoj fiksnoj konačnoj azbuci $\Sigma_{\mathcal{U}}$), i da mašina $\mathcal{U}$ na određeni način simulira rad mašine $\mathcal{M}$ za ulaznu reč w, tako da je rezultat rada $\mathcal{U}$ za opisani ulaz isti kao i rezultat obrade w na $\mathcal{M}$. Prema tome, želimo da bude

$$L(\mathcal{U}) = \{(\mathcal{M}, w) : \ w \in L(\mathcal{M})\},$$

tj. da $\mathcal{U}$ prihvata kod para $(\mathcal{M}, w)$ ako i samo ako w prihvata $\mathcal{M}$.

Ispostavlja se da ovakva mašina $\mathcal{U}$ *postoji*, i da sama njena egzistencija — kao što ćemo to videti u narednom odeljku — ima više nego značajne posledice u teoriji algoritama.

Kako bismo mogli da precizno opišemo mašinu $\mathcal{U}$, moramo najpre da vidimo kako ćemo proizvoljnu mašinu $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \top, \bot)$ i $w \in \Sigma^*$ pretvoriti u podatak, tj. reč nad $\Sigma_{\mathcal{U}}$. Za ulaznu azbuku mašine $\mathcal{U}$ dovoljno će biti da uzmemo $\Sigma_{\mathcal{U}} = \{0, 1, \#\}$. Naime, možemo poći od pretpostavke (koja nije nimalo ograničavajuća) da su kod svih mašina $\mathcal{M}$ koje razmatramo skupovi stanja i azbuke početni segmenti od $\mathbb{N}$, tj. da je

$$\begin{aligned} Q &= \{0, 1, \ldots, n-1\}, \\ \Sigma &= \{0, 1, \ldots, m-1\}, \\ \Gamma &= \{0, 1, \ldots, k-1\}, \end{aligned}$$

gde je $m < k$, dok su $r, s, t < n$ i $u < k$ redom početno, prihvatno, odbijajuće stanje i blanko. Tako, kod mašine $\mathcal{M}$ može započeti prefiksom

$$0^n 1 0^m 1 0^k 1 0^r 1 0^s 1 0^t 1 0^u 1,$$

koji se završava sa sedmim pojavljivanjem simbola 1, i koji se jedinstveno dekoduje tako da $\mathcal{M}$ ima n stanja, njena azbuka k simbola, od kojih su prvih m

ulazni, dok ostatak reči definiše istaknuta stanja i redni broj simbola $*$. Dalje, svaka pojedinačna instrukcija $\delta(p, a) = (\alpha, q)$ može biti kodirana sa

$$0^p 10^a 10^{b+3} 10^q 1$$

ako je $\alpha = b \in \Gamma$, odnosno sa

$$0^p 10^a 10^z 10^q 1$$

ako je $\alpha \in \{L, R, H\}$, gde akcijama redom pridružujemo redne brojeve $z = 0, 1, 2$. Tako, niz ovakvih sekvenci sa po četiri simbola 1 jedinstveno određuje funkciju prelaza δ. Naravno, i reč w se može analogno kodirati:

$$0^{x_0} 1 \ldots 10^{x_i}$$

pokazuje da je u pitanju bila reč $w = x_0 \ldots x_i$ (za $w = \lambda$ kod je takođe prazna reč). Najzad, kod para $(\mathcal{M}, w)$ je

$$\langle \text{kod mašine } \mathcal{M} \rangle \# \langle \text{kod reči } w \rangle,$$

što ćemo u daljem (uz neznatnu zloupotrebu notacije) pisati $\mathcal{M}\#w$.

Sada prelazimo na skicu mašine $\mathcal{U}$ (koja, podsetimo se, za ulaz $\mathcal{M}\#w$ treba da prihvati / odbije / uđe u mrtvu petlju ako i samo ako $\mathcal{M}$ prihvata / odbija / ulazi u mrtvu petlju za ulaz w). $\mathcal{U}$ će imati 3 trake, od kojih je prva *read-only* i sadrži ulaznu reč. Na drugoj traci se odigrava simulacija mašine $\mathcal{M}$ (ako je ulaz oblika $\mathcal{M}\#w$), dok treća traka služi kako bi čuvala kod trenutnog stanja kontrolnog mehanizma simulirane mašine.

Pre nego što započne bilo kakva simulacija, mašina $\mathcal{U}$ mora da obavi malo pripremnog rada. Najpre, ona mora da utvrdi da li je ulazna reč oblika $\mathcal{M}\#w$ za neku mašinu $\mathcal{M}$ i reč w nad njenom ulaznom azbukom (prilično je očigledno da postoji niz komandi koji rešava ovaj potproblem). Ukoliko nije tako, $\mathcal{U}$ odmah prelazi u stanje odbijanja i zaustavlja se. Ako ulaz ima traženi oblik, kod reči w se kopira na drugu traku, a na treću traku se upisuje 0^r, gde je r redni broj početnog stanja. Glava druge trake se postavlja desno od kodirane reči w.

Sada počinje simulacija. U svakom koraku, mašina $\mathcal{U}$ poredi niz nula na trećoj traci i niz uzastopnih nula levo od položaja glave na drugoj traci sa "blokovima" (sa četiri jedinice) u onom delu koda koji reprezentuje funkciju prelaza δ. Kada se pronađe poklapanje — što znači da je sadržaj treće trake 0^p, $p \geqslant 0$, a druge $\ldots 0^a \hat{1} \ldots$ (ili $\ldots 0^a \hat{*}$), i na prvoj traci se desno od sedme jedinice pronalazi podreč $0^p 10^a 1$ — čita se ostatak "bloka" $0^c 10^q 1$ i preduzima se ažuriranje druge i treće trake koje odgovara jednom koraku mašine $\mathcal{M}$ (broj nula

na trećoj traci postaje q, a u zavisnosti od vrednosti c, mašina se zaustavlja, glava druge trake "skače" na prvu jedinicu levo/desno, ili se menja dužina niza uzastopnih nula levo od glave). Prema tome, nakon simulacije i-tog koraka mašine $\mathcal{M}$, $i \geqslant 0$, trake mašine $\mathcal{U}$ izgledaju ovako:

$\mathcal{M}\#w$
$\langle$kod sadržaja trake mašine $\mathcal{M}$ posle i-tog koraka$\rangle$
$\langle$kod stanja kontr. mehanizma mašine $\mathcal{M}$ posle i-tog koraka$\rangle$

Ukoliko se na trećoj traci pojave kodovi prihvatnog / odbijajućeg stanja (što se ustanovljava poređenjem sa odgovarajućim kodom mašine $\mathcal{M}$), mašina $\mathcal{U}$ ulazi u svoje prihvatno / odbijajuće stanje.

Sada se lako uočava da ovako skicirana mašina odgovara našim zahtevima: ishod rada mašine $\mathcal{U}$ sa ulaznom reči $\mathcal{M}\#w$ isti je kao i ishod rada mašine $\mathcal{M}$ sa ulaznom reči w.

3.5 Neodlučivi jezici i problemi

Kao što smo videli u Teoremama 3.1 i 3.2, klasa rekurzivno nabrojivih jezika se poklapa sa klasom jezika svih Tjuringovih mašina, dok su rekurzivni jezici tačno jezici totalnih Tjuringovih mašina. S druge strane, problem kome odgovara nerekurzivan jezik je *neodlučiv*, tj. nije rešiv putem algoritama.

U Odeljku 2.10 smo videli da važi inkluzija $\mathbf{R} \subseteq \mathbf{RE}$. Osim toga, trivijalno važi i $\mathbf{RE} \subseteq \mathcal{P}(\mathbb{N})$. Međutim, ono što za sada još ne znamo je: da li su ove dve inkluzije stroge? Ispostaviće se da je u oba slučaja odgovor pozitivan. U narednim tvrđenjima, koja spadaju u temeljne rezultate teorije izračunljivosti (budući da ukazuju na fundamentalna ograničenja izračunljivosti kao koncepta), upoznaćemo se sa nekim od "klasičnih" neodlučivih problema odlučivanja (tj. jezika).

Pretpostavimo najpre da nam je data reč u nad binarnom azbukom $\{0, 1\}$. Ova reč može biti upravo kod neke Tjuringove mašine. Ukoliko je to slučaj, odgovarajuću mašinu (sa konvencijama u vezi sa $Q, \Sigma, \Gamma, \ldots$ kao i u prethodnom odeljku) označavamo sa $\mathcal{M}_u$. Ako ulazna azbuka ove mašine ima bar dva simbola, tada sama reč u može biti ulazni podatak za rad mašine $\mathcal{M}_u$. Stoga možemo definisati tzv. *dijagonalni jezik*:

$$L_d = \{u \in \{0,1\}^* : \ u \text{ je kod neke mašine i } u \notin L(\mathcal{M}_u)\},$$

budući da on nastaje upravo kao rezultat jedne "kantorovske" dijagonalne konstrukcije (baš kao u dokazu da ne postoji bijekcija između $\mathbb{N}$ i $\mathbb{R}$, odnosno između bilo kog skupa X i njegovog partitivnog skupa $\mathcal{P}(X)$). U L_d možemo ubrojati i sve kodove mašina nad jednoelementnim alfabetom (slučaj $\Sigma = \{0\}$), pošto i tada možemo smatrati da automatski važi $u \notin L(\mathcal{M}_u)$.

Teorema 3.7 *Jezik L_d nije rekurzivno nabrojiv.*

Dokaz. Pretpostavimo suprotno. Po Teoremi 3.1, tada postoji Tjuringova mašina $\mathcal{M}$ nad $\Sigma = \{0, 1\}$ (sa svim ostalim parametrima u skladu sa zahtevima iz konstrukcije univerzalne mašine $\mathcal{U}$) tako da je $L_d = L(\mathcal{M})$. Neka je u kod mašine $\mathcal{M}$, tj. $\mathcal{M} = \mathcal{M}_u$.

Kao i u drugim situacijama u kojima imamo posla sa dijagonalizacijom, pitamo se da li važi $u \in L_d$. Ako bi bilo $u \in L_d$, to bi značilo da $u \notin L(\mathcal{M}_u) = L(\mathcal{M}) = L_d$. S druge strane, pretpostavka $u \notin L_d = L(\mathcal{M}_u)$ odmah vodi zaključku $u \in L_d$. Kontradikcija. □

Iz prethodne teoreme sledi da ne postoji algoritam koji za proizvoljnu mašinu $\mathcal{M}$ odlučuje da li $\mathcal{M}$ prihvata svoj sopstveni kod ili ne. Drugim rečima, ako binarna reč u predstavlja ulazni podatak, tada je pitanje $u \in L(\mathcal{M}_u)$ neodlučivo.

Još značajniji jezik u teoriji izračunljivosti je tzv. *univerzalni jezik*:

$$L_{\text{univ}} = \{u\#w : \ u \text{ je kod neke mašine i } w \in L(\mathcal{M}_u)\}.$$

Ovaj jezik odgovara opštem problemu — poznatom i kao *problem pripadnosti* (engl. *membership problem* (MP)) — koji pita, za datu mašinu $\mathcal{M}$ i reč w u njenoj azbuci, da li $w \in L(\mathcal{M})$. Lako se može uočiti da bi postojanje algoritma koji rešava ovaj problem imalo za direktnu posledicu $\mathbf{R} = \mathbf{RE}$: naime, tada bi taj pretpostavljeni "super-algoritam" za svaku fiksnu mašinu $\mathcal{M}$ odlučivao da li važi $w \in L(\mathcal{M})$, za datu reč w. Drugim rečima, svi jezici Tjuringovih mašina (tj. rekurzivno nabrojivi jezici) bi bili rekurzivni. Da to nije tako, pokazuje

Teorema 3.8 (A.M.Turing, 1936) *Jezik L_{univ} jeste rekurzivno nabrojiv, ali nije rekurzivan.*

Dokaz. Najpre, očigledno je da je L_{univ} upravo jezik univerzalne mašine $\mathcal{U}$, $L_{\text{univ}} = L(\mathcal{U})$, pa je on rekurzivno nabrojiv po Teoremi 3.1.

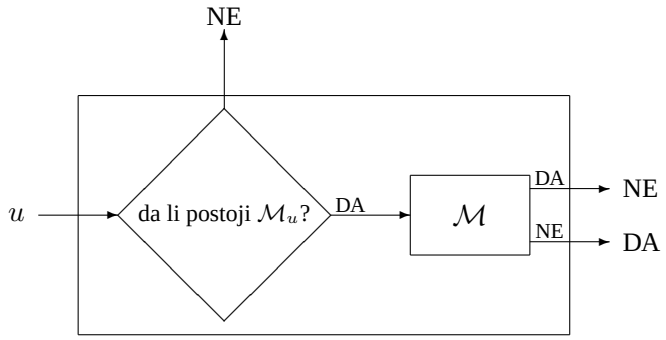
Pretpostavimo sada, suprotno tvrđenju teoreme, da je jezik L_{univ} rekurzivan. U tom slučaju, postoji *totalna* Tjuringova mašina $\mathcal{M}$ koja ga prihvata. Razmotrimo mašinu $\mathcal{M}'$ nad $\{0, 1\}$ koja je konstruisana tako da radi na sledeći način:

(1) Za dati ulaz $u \in \{0,1\}^*$, $\mathcal{M}'$ najpre proverava da li je u kod neke mašine. Ako nije, $\mathcal{M}'$ ulazi u odbijajuće stanje i zaustavlja se.

(2) Ako je rezultat testa pod (1) potvrdan, reč $u\#u$ se prosleđuje mašini $\mathcal{M}$. Pošto je mašina $\mathcal{M}$ totalna, njen rad se završava u konačno mnogo koraka u nekom od stanja $\top, \bot$.

(3) Po okončanju rada mašine $\mathcal{M}$, mašina $\mathcal{M}'$ se takođe zaustavlja, s tim da je završno stanje mašine $\mathcal{M}'$ *suprotno* od onog u kojem je $\mathcal{M}$ završila rad.

Šematski, $\mathcal{M}'$ se može prikazati ovako:



Odmah se uočava da $\mathcal{M}'$ prihvata w ako i samo ako je u kod neke mašine i $\mathcal{M}_u$ ne prihvata u. Drugim rečima, $L(\mathcal{M}') = L_d$. Međutim, po prethodnoj teoremi, L_d nije rekurzivno nabrojiv jezik, stoga ne može biti jezik ni jedne Tjuringove mašine, pa ni mašine $\mathcal{M}'$. Kontradikcija. □

Prema tome, ne postoji algoritam koji za proizvoljnu mašinu $\mathcal{M}$ odlučuje da li data reč w pripada jeziku $L(\mathcal{M})$ ili ne. Štaviše, postoji *fiksna* mašina $\mathcal{M}$ kod koje je pitanje $w \in L(\mathcal{M})$ (za datu reč w) neodlučivo: na primer, takva je univerzalna mašina $\mathcal{U}$ i njen jezik L_{univ} iz prethodne teoreme. Dakle, postoje Tjuringove mašine čiji jezici *nisu* rekurzivni. Samim tim, postoje nerekurzivni rekurzivno nabrojivi skupovi, odakle, između ostalog, sledi i naredno tvrđenje.

Posledica 3.9 *Postoji (unarna) prosto rekurzivna funkcija f čija je slika (kodomen) nerekurzivan skup.*

Može se pokazati da je neodlučiv čak i "jednostavniji" problem od problema pripadnosti — u pitanju je tzv. *problem zaustavljanja* ("halting problem" (HP)). Odgovarajući "halting" jezik je sledeći:

$$L_h = \{u\#w : \ u \text{ je kod neke mašine i } \mathcal{M}_u \text{ se za ulaz } w \text{ zaustavlja}\}.$$

Pri tome je u potpunosti irelevantno stanje u kojem se zaustavila mašina $\mathcal{M}_u$. Jasno, $L_u \subseteq L_h$.

Teorema 3.10 $L_h \in \mathbf{RE} \setminus \mathbf{R}$.

Dokaz. Malom modifikacijom univerzalne mašine $\mathcal{U}$ dobija se mašina $\mathcal{U}'$ čiji je jezik baš L_h. Naime, dovoljno je "prepraviti" $\mathcal{U}$ tako da pri simulaciji mašine $\mathcal{M}$ koja se odigrava za ulaz $\mathcal{M}\#w$ mašina $\mathcal{U}'$ prelazi u prihvatno stanje čim se $\mathcal{M}$ zaustavi, *nezavisno* od stanja u kojem $\mathcal{M}$ okončava rad. Ovo pokazuje da je jezik L_h rekurzivno nabrojiv.

S druge strane, pretpostavimo da je jezik L_h rekurzivan. Tada postoji totalna mašina $\mathcal{M}$ koja ga prihvata. Posmatrajmo sada mašinu $\mathcal{M}'$ koja radi na sledeći način (za ulaz $u\#w$, $u, w \in \{0,1\}^*$):

(1) Ulaz $u\#w$ se prvo prosleđuje mašini $\mathcal{M}$. Ako je rezultat njenog izračunavanja negativan, $\mathcal{M}'$ vraća NE i zaustavlja se.

(2) Ako je odgovor mašine $\mathcal{M}$ pozitivan (što znači da $u\#w \in L_h$, tj. $\mathcal{M}_u$ postoji i zaustavlja se za ulaz w), ulazna reč $u\#w$ se prosleđuje mašini $\mathcal{U}$, koja simulira $\mathcal{M}_u$ za ulaz w. Rezultat njenog rada je ujedno i rezultat rada mašine $\mathcal{M}'$.

Mašina $\mathcal{M}'$ je očito totalna. Osim toga, $L(\mathcal{M}') = L(\mathcal{U}) = L_{\text{univ}}$, što je kontradikcija sa nerekurzivnošću univerzalnog jezika. □

Dakle, i halting problem je algoritamski nerešiv. Štaviše, može se lako pokazati da je neodlučiva i restrikcija halting problema koja pita da li se data mašina $\mathcal{M}$ (data, naravno, svojim kodom) zaustavlja za prazan ulaz (reč λ).

Koristeći navedene ili druge poznate neodlučive jezike, može se pokazati algoritamska nerešivost širokog spektra problema u raznim oblastima matematike. Ispostavlja se da je neodlučivost zapravo učestala pojava, pre pravilo nego "egzotični" raritet. U preostalom delu ovog odeljka pomenućemo tri poznata problema za koje danas znamo da se ne mogu rešiti putem algoritama.

Deseti Hilbertov problem. Na svetskom kongresu matematičara održanom 1900. u Parizu, David Hilbert je održao predavanje koje je ušlo u istoriju matematike. U okviru svog izlaganja Hilbert je izneo 23 tada otvorena problema za koje je verovao da će predstavljati okosnicu matematičkih istraživanja u narednom stoleću. I zaista, Hilbertovi problemi su u velikoj meri odredili i usmerili razvoj matematike XX veka. Među tim problemima, *deseti* je tražio da se pronađe opšti metod, postupak (danas bismo rekli: algoritam) kojim bi se utvrđivala rešivost *diofantskih jednačina*. Preciznije, problem je sledeći:

ULAZ: Polinom $f(x_1, \ldots, x_m) \in \mathbb{Z}[x_1, \ldots, x_m]$ od više promenljivih sa celim koeficijentima,

$$f(x_1, \ldots, x_m) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_m=0}^{n_m} a_{i_1 \ldots i_n} x_1^{i_1} \ldots x_m^{i_m}.$$

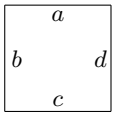
IZLAZ: Da li postoji celobrojno rešenje jednačine $f(x_1, \ldots, x_m) = 0$?

Deseti Hilbertov problem se dugo opirao rešenju. Najzad, 1970. godine, ruski matematičar *Jurij V. Matijaševič* je pokazao da je ovaj problem neodlučiv — algoritam koji je Hilbert tražio zapravo ne postoji. Iz Matijaševičevih opštih rezultata naime sledi da postoji polinom f od 57 promenljivih sa celim koeficijentima takav da za $a \in \mathbb{N}$ diofantska jednačina

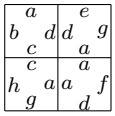
$$f(x_1, \ldots, x_{56}, a) = 0$$

ima rešenje po $x_1, \ldots, x_{56}$ u skupu $\mathbb{Z}$ ako i samo ako je $||u\#w|| = a$ za neku reč $u\#w \in L_h$. Stoga bi postojanje algoritma za utvrđivanje rešivosti diofantskih jednačina omogućavalo algoritamsko rešavanje halting problema, što je nemoguće.

Problem domina. Pod *dominom* podrazumevamo kvadrat fiksnih dimenzija čije su sve stranice označene simbolima $(a, b, c, d, \ldots)$. Pri tome je iz tehničkih razloga zgodno da pretpostavimo da je svaki od simbola zapravo oznaka za neki binarni niz. Tako, jedna domina može da izgleda ovako:



Domine se slažu u ravni jedna do druge tako da ivice kvadratića koje se dodiruju moraju imati iste oznake, npr.



Pri tome, nije dozvoljeno rotiranje domina: domina je, formalno, uređena četvorka (a, b, c, d) gde niz oznaka počinje od gornje stranice i nastavlja se u smeru kretanja kazaljke sata.

Konačan skup domina koji je na raspolaganju u pojedinačnoj instanci ovog problema može se sada kodirati kao reč u azbuci $\{0, 1, \$, \#\}$, gde se jedna domina zapisuje kao $a\$b\$c\$d$ (a, b, c, d su binarni nizovi), dok $\#$ služi za razdvajanje kodova dve različite domine.

Problem domina je sledeći:

ULAZ: Konačan skup domina.

IZLAZ: Da li postoji popločavanje ravni dominama u skladu sa opisanim pravilom slaganja (pri čemu se svaka od datih domina može koristiti neograničeno mnogo puta)?

Ovaj problem je neodlučiv. Štaviše, može se pokazati (uz pomoć Königove leme iz teorije grafova) da ako D označava jezik koji odgovara problemu domina, tada je $\overline{D}$, komplement jezika D, rekurzivno nabrojiv. Otuda sledi po Postovoj teoremi (Teorema 2.30) da problem domina nije čak ni rekurzivno nabrojiv, jer bi u suprotnom D bio rekurzivan.

Postov problem korespondencije. Ovaj neodlučiv problem potiče od Emila L. Posta. Neka je $\Sigma = \{a_0, a_1, \ldots\}$ neka fiksirana prebrojiva azbuka (slično prethodnim situacijama, njena slova se mogu kodirati binarnim rečima tako što a_j predstavimo kao 0^j1). Postov problem korepondencije (PCP) je sledeći:

ULAZ: Konačan skup P parova (u, v), gde je $u, v \in \Sigma^*$.

IZLAZ: Da li postoje (ne nužno različiti) parovi $(u_1, v_1), \ldots, (u_n, v_n) \in P$ tako da je

$$u_1 \ldots u_n = v_1 \ldots v_n\,?$$

Intuitivno, ovaj problem možemo zamisliti na sledeći način. Dat nam je konačan skup "kartica" na kojima se nalazi po jedna reč gore i dole, pri čemu nam od svake kartice na raspolaganju stoji neograničen broj primeraka. PCP pita da li je moguće napraviti izbor kartica tako da se, kada se one poređaju u niz jedna do druge, reči u gornjem i donjem redu poklapaju.

Neodlučivost problema PCP nalazi svoju veliku primenu u teorijskom računarstvu i matematici, pošto se neodlučivost mnogih algoritamskih problema pokazuje upravo zahvaljujući svođenjem PCP na njih.

3.6 RAM mašine

Model višetračne Tjuringove mašine u velikom broju slučajeva značajno olakšava implementaciju i analizu raznih algoritamskih problema. Ipak, taj model nije rešio problem zbog koga je uveden. Umesto jedne linearne (sekvencijalne)

memorije, kod VTM imamo k takvih memorija: tako, u jednom trenutku imamo neposredan pristup većem, ali ipak ograničenom broju podataka. Jedino pravo rešenje krije se u suštinski drugačijoj koncepciji memorije, koja bi omogućila neposredan pristup neograničenom broju podataka. Reč je o organizaciji memorije koja se danas primenjuje na realnim računarima, a koja poznata pod akronimom RAM (*random access memory*). Na taj način, dobijamo koncept *RAM mašine* koji su 1963. godine formalizovali *J.C.Shepherdson* i *H.E.Sturgis* [39], polazeći od osnovnih principa arhitekture računara koje je ranije postavio *John von Neumann* (rođen kao *Neumann János Lajos*, 1903–1957).

RAM mašina je računski model koji radi sa celim brojevima (standardno se uzima da su oni zapisani u binarnom sistemu, uz dodatni bit koji definiše znak). Njena osnovna memorijska jedinica jeste *registar*, koji može da uskladišti proizvoljno veliki ceo broj. RAM mašina se sastoji od tri komponente:

- memorije (RAM),
- procesora, i
- programa.

RAM sadrži beskonačno mnogo registara (to je, uz veličinu registara, jedino po čemu se suštinski razlikuje od realnog računara), i ti registri su numerisani brojevima $1, 2, \ldots$. Ovaj pridruženi broj registra je njegova *adresa*, i ona je ta koja omogućava neposredni pristup svakom registru. Na početku rada svi registri sadrže broj 0, osim registara sa adresama $1, 2, \ldots, k$, koji sadrže ulazne podatke. Broj uskladištenih u i-tom registru ćemo označavati sa r_i.

Procesor se sastoji od dva registra. Jedan se zove *akumulator*, i to je registar sa adresom 0. On ima posebnu ulogu, jer se sve operacije na RAM mašini izvode unutar njega. Drugi registar je *brojač* (koji označavamo sa $count$) i u njemu se može naći isključivo nenegativan broj. Na početku on sadrži broj 1, a inače pokazuje redni broj komande u programu koja se trenutno izvršava. Mašina završava svoj rad tako što se u brojaču pojavljuje vrednost 0.

Najzad, program je konačan niz komandi. Spisak mogućih komandi (koje delimo u tri grupe) dat je u narednoj tabeli.

Aritmetika	***Ulaz-izlaz***	***Kontrola***
ADD arg	LOAD arg	JUMP lab
SUB arg	STORE arg	JPOS lab
MULT arg		JZERO lab
DIV arg		JNEG lab
		HALT

U prve dve grupe komandi (sa izuzetkom STORE) argument arg može biti "$= i$", "i" ili "$\uparrow i$", što respektivno označava vrednosti i, r_i i r_{r_i} (dakle, doslovno vrednost i, vrednost i-tog registra, i *pokazivač*, vrednost registra čija je adresa zapisana u i-tom registru). U komandama iz treće grupe, argument lab je pozitivan prirodan broj, labela neke komande iz programa.

Komande realizuju sledeće operacije:

- ADD / SUB / MULT / DIV arg: izvođenje odgovarajuće aritmetičke operacije sa sadržajem akumulatora kao prvim i arg kao drugim argumentom (pod DIV se podrazumeva celobrojno deljenje),
- LOAD arg: $r_0 := arg$,
- STORE i: $r_i := r_0$, STORE $\uparrow i$: $r_{r_i} := r_0$,
- JUMP lab: $count := lab$,
- JPOS / JZERO / JNEG lab: ako je $r_0 > / = / < 0$, tada $count := lab$,
- HALT: $count := 0$.

Smatramo da je svaka "besmislena" komanda (npr. pozivanje registra sa negativnom adresom ili deljenje nulom) ekvivalentna komandi HALT za zaustavljanje mašine. Pri tome, izvršenje svake od komandi iz prve dve grupe uvećava vrednost brojača za 1, dok preostale kontrolne komande eksplicitno definišu narednu vrednost brojača, tj. redni broj komande koja će se sledeća izvršiti. Kao što je već naglašeno, mašina završava svoj rad kada se u brojaču nađe broj 0. Sadržaj akumulatora u tom trenutku se smatra za rezultat izračunavanja mašine.

U izvesnom smislu, komande MULT i DIV predstavljaju "luksuz". Naime, može se pokazati (vidi [33]) da ako iz mašinskog jezika RAM mašina uklonimo ove dve komande i koristimo komandu HALF koja sadržaj akumulatora r_0 zamenjuje sa $\lfloor r_0/2 \rfloor$, tada MULT arg možemo isprogramirati u nekoliko desetina redova, dok se program za DIV tada može napisati u skladu sa razmatranjima iz Primera 2.7.

Sistemi obračunavanja vremenske složenosti RAM mašina

Najjednostavniji pristup u proceni vremenske složenosti RAM mašine jeste da za vremensku jedinicu uzmemo izvršenje jedne komande programa. Na taj način, dobijamo *uniformni* sistem analize složenosti.

Uniformni sistem koristimo u analizi većine algoritama. Međutim, treba u najmanju ruku biti oprezan u njegovoj upotrebi, jer, kao što su to 1974. pokazali

Hartmanis i Simon, moguće je *zloupotrebiti* ovaj sistem tako što se užasno komplikovana izračunavanja kodiraju ogromnim celim brojevima, i zatim se sam algoritamski problem svodi na njihovo množenje (pa bi se tako esencijalno eksponencijalni algoritmi realizovali u polinomnom vremenu). Evo jednog jednostavnog primera: izračunavanje funkcije $f(n) = 3^{2^n}$. Sledeći RAM program računa ovu funkciju (na početku, broj n je u prvom registru).

```
 1  LOAD =3
 2  STORE 2
 3  LOAD 1
 4  JZERO 12
 5  LOAD 2
 6  MULT 2
 7  STORE 2
 8  LOAD 1
 9  SUB =1
10  STORE 1
11  JUMP 4
12  LOAD 2
13  HALT
```

Algoritamska ideja implementirana u ovom programu je da se do vrednosti $f(n)$ dođe uzastopnim kvadriranjem, polazeći od inicijalne vrednosti 3 (komanda 1). Registar br.1 se koristi kao brojač, i sve dok je njegova vrednost pozitivna (komanda 4) vrši se kvadriranje (komande 5 i 6) međurezultata, koji se skladišti u registru br.2. Kada vrednost brojača postane 0, učitava se vrednost funkcije u akumulator (komanda 12) i program okončava svoj rad. Kako komande 4–11 čine petlju koja se izvršava tačno n puta, složenost ovog programa je $8n + 6$, dakle $\mathcal{O}(n)$. Skrećemo pažnju da je ova složenost eksponencijalna (a ne linearna, kako se to na prvi pogled čini), budući da je ulazni podatak n, pa je njegova dužina d (u binarnom zapisu) približno $\log_2 n$. Stoga je složenost gornjeg algoritma $\mathcal{O}(2^d)$.

Međutim, u stvarnosti je situacija mnogo lošija. Kako je $3^{2^k} > 2^{2^k}$, zaključujemo da se broj 3^{2^k} zapisuje sa bar $2^k + 1$ binarnih cifara. Zbog toga, komanda 6 (kojom se kvadrira prethodni međurezultat) u m-toj iteraciji podrazumeva množenje dva broja sa ne manje od $2^{m-1}+1$ binarnih cifara, što rezultuje brojem sa bar $2^m + 1$ cifara. Primera radi, za $m = 1000$, taj broj ima više od 2^{1000} cifara, i procenjuje se da svi listovi papira ikada proizvedeni na planeti Zemlji nisu dovoljni da se taj broj zapiše. Jasno je da se na realnim računarima

izvođenje operacija sa takvim brojevima (ukoliko uopšte ima dovoljno memorije na raspolaganju) ne može ostvariti u istom vremenu kao i množenje dva bita. Šta više, takav obračun složenosti ne bi bio opravdan ni na idealizovanim RAM mašinama, budući da se samo množenje rekurzivno definiše kao uzastopno sabiranje (vidi Primer 2.1), pa bi se u slučaju velikih argumenata ta operacija teško mogla smatrati elementarnom. Isto važi i za sabiranje velikih brojeva, budući da se ono na očigledan način sastoji od niza sabiranja jednocifrenih brojeva.

Zbog toga, uvodimo tzv. *logaritamski* sistem obračunavanja vremenske složenosti. Ovaj sistem respektuje veličinu argumenata prilikom izvršavanja svih komandi iz programa[12]. Kod ulazno-izlaznih komandi, sabiranja i oduzimanja, obračunava se vremenski utrošak $\mathcal{O}(\log n)$, gde n predstavlja maksimum argumenata komande. Kada je u pitanju množenje i celobrojno deljenje, razlikujemo dve "podvrste" logaritamskog sistema. Jedan pristup se sastoji u tome da se i za ove operacije utrošak vremena računa na isti način kao i kod drugih operacija. Taj pristup — iako tehnički jednostavniji — takođe nije realan. Naime, ako uzmemo izvođenje aritmetičke operacije na dva bita kao osnovnu vremensku jedinicu, tada algoritam za sabiranje zaista radi u linearnom vremenu u odnosu na logaritme sabiraka: to je upravo algoritam sa potpisivanjem i sabiranjem "cifru-po-cifru" (sa prenosom) koji smo svi učili u školi. Međutim, lako se može videti da "klasičan" algoritam za množenje dva k-tocifrena broja uključuje *kvadratno* mnogo (u odnosu na k) osnovnih operacija na bitovima (a isto važi i za deljenje, vidi Odeljak 4.1). Danas nije poznat nijedan algoritam za množenje brojeva koji se sastoji iz $\mathcal{O}(k)$ takvih elementarnih operacija, i pitanje egzistencije takvog algoritma je otvoren problem[13]. Stoga je najrealniji — ali istovremeno najteži za primenu — "strogi" logaritamski sistem u kome je osnovna vremenska jedinica izvođenje neke operacije na nivou bitova, pri čemu se izvršenje komandi MULT i DIV obračunava sa $\mathcal{O}(\log^2 n)$, gde je n veći od dva argumenta.

Izbor sistema kojeg koristimo za proračun vremenske složenosti nekog algoritma jeste pre svega pitanje mere, "dobrog ukusa" i prilagođavanja karakteru razmatranog problema. Preterano insistiranje na logaritamskom sistemu bi u najmanju ruku zamaglilo matematičku analizu većine problema. Zbog toga, *pravilna* upotreba unformnog sistema na podacima umerene veličine čini matematička razmatranja transparentnijim, i jasnije otkriva suštiniski stepen složenosti algoritamskog zadatka. Uniformni sistem ćemo bez izuzetka koristiti

[12] Ovo, naravno, nema nikakvog uticaja na kontrolne komande, budući da najveća moguća vrednost njihovog argumenta lab zavisi od dužine programa — koji je fiksan broj, nezavisan od ulaznih podataka.

[13] Asimptotski najbolji rezultat u tom pravcu dali su A.Schönhage i V.Strassen, koji su našli algoritam za množenje k-tocifrenih brojeva koji sadrži $\mathcal{O}(k \log k \log\log k)$ operacija na bitovima.

kod analize svih grafovskih algoritama, problema u vezi sa iskaznim formulama, itd.

Kada je u pitanju ekvivalentnost računskih moći TM i RAM mašina, kao i odnos njihovih mera vremenske složenosti, imamo sledeće tvrđenje. Skice odgovarajućih simulacija se mogu naći npr. u [33, 37].

Teorema 3.11 *Neka je $\mathcal{R}$ RAM mašina. Tada postoji (višetračna) TM $\mathcal{M}$ koja simulira $\mathcal{R}$ (u istom smislu kao i u Teoremi 3.6). Pri tome, ako je $T_{\mathcal{R}}(n) = \mathcal{O}(t(n))$ za neku funkciju $t : \mathbb{N} \to \mathbb{N}$, tada se simulacija $\mathcal{M}$ može konstruisati tako da je njena vremenska složenost $T_{\mathcal{M}}(n)$ jednaka:*

- *$\mathcal{O}([t(n)]^2)$, ako se na $\mathcal{R}$ primenjuje osnovni logaritamski sistem obračunavanja, ili ako njen program ne sadrži komande MULT i DIV,*
- *$\mathcal{O}([t(n)]^3)$, ako se na $\mathcal{R}$ primenjuje striktni logaritamski sistem obračunavanja.*

Prema tome, strogo logaritamski obračunate RAM mašine su polinomno ekvivalentne osnovnom modelu TM (uz ušestostručenje stepena polinoma koji izražava složenost). Kao neposredna posledica ovog zaključka, sledi da za svaki jezik L čiji se problem odlučivanja može rešiti na RAM mašini (uz odgovarajuće binarno kodiranje azbuke) u polinomnom vremenu, važi $L \in \mathbf{P}$.

U narednim glavama, naš osnovni računski model biće upravo RAM mašina, imajući u vidu njenu ekvivalentnost sa TM u smislu Teoreme 3.11. To otvara mogućnost da algoritme zapisujemo u pseudo-kodu, koristeći imperativni programski stil, budući da RAM mašine nisu ništa drugo do idealizacija realnih računara.

Zadaci.

1. Bez korišćenja komande DIV, napisati RAM program koji izračunava celobrojni količnik dva broja. Oceniti složenost ovog programa.

2. Oceniti broj sabiranja i množenja binarnih cifara neophodnih za množenje dva n-tocifrena binarna broja.

4

Algoritmi polinomne vremenske složenosti

U ovoj glavi ćemo razmotriti nekoliko ”klasičnih” algoritama koji rade u polinomnom vremenu u odnosu na veličinu ulaznih podataka. Najpre ćemo prikazati jedan od najstarijih i najpoznatijih algoritama: Euklidov algoritam za izračunavanje NZD dva broja, a kroz taj prikaz ćemo ”provući” i analizu algoritma za celobrojno deljenje. Efekte primene poznate algoritamske strategije ”podeli i vladaj” (eng. *divide and conquer*) uočićemo kod množenja (celobrojnih) matrica. Takođe, razmotrićemo i jedan problem odlučivanja iz iskazne logike, problem zadovoljivosti Hornovih iskaznih formula, kao oslabljenje opšteg problema SAT (kojim ćemo se više pozabaviti u Glavi 6). Najzad, prikazaćemo i tri klasična grafovska algoritma: za pretraživanje grafova, za izračunavanje dužine najkraćeg puta iz datog izvora u težinskom grafu (Dajkstrin algoritam), kao i za nalaženje minimalnog razapinjućeg stabla u težinskom grafu.

4.1 Euklidov algoritam

Euklidov algoritam, jedan od istorijski najznačajnijih algoritama, izračunava najveći zajednički delitelj dva data prirodna broja $x, y > 0$. Kao što je poznato, on se sastoji u iterativnom nalaženju celobrojnih količnika i ostataka pri deljenju. Naime, u prvom koraku izražavamo (pretpostavljajući da je $x \geqslant y$)

$$x = q_0 y + r_0,$$

gde je $0 \leqslant r_0 < y$. Nakon toga, par (y, r_0) preuzima ulogu (x, y):

$$y = q_1 r_0 + r_1,$$

$0 \leqslant r_1 < r_0$, a zatim uzastopno vršimo deljenja

$$r_i = q_{i+2} r_{i+1} + r_{i+2},$$

pri čemu je $i \geqslant 0$ i $0 \leqslant r_{i+2} < r_{i+1}$. Naravno, tu je

$$q_{i+2} = \left\lfloor \frac{r_i}{r_{i+1}} \right\rfloor.$$

Kako na ovaj način dobijamo opadajući niz

$$x \geqslant y > r_0 > r_1 > \ldots > r_i > r_{i+1} > r_{i+2} > \ldots,$$

ovaj postupak se mora okončati nakon konačno mnogo iteracija, tj. za neko $k \geqslant 0$ imamo $r_k = 0$. Tada se lako pokazuje da je $\mathrm{NZD}(x, y) = r_{k-1}$, poslednji nenula član gornjeg niza (smatramo da je $r_{-1} = y$), što je upravo željeni rezultat izračunavanja.

Euklidov algoritam.

1. Proveri da li je $x, y > 0$. Ako jeste, $a := x$, $b := y$, $c := y$. Ako nije, vrati poruku greške.
2. Dok je $c \neq 0$, radi:
   - $c := \mathrm{rest}(a, b)$;
   - $a := b$;
   - $b := c$.
3. $\mathrm{NZD} := a$.

Kako bismo analizirali složenost Euklidovog algoritma, treba da učinimo dve stvari:

- da ocenimo (u zavisnosti od x, y) broj iteracija u koraku 2,
- da ocenimo složenost jedne iteracije.

Naravno, pošto se jedna iteracija sastoji iz jednog nalaženja ostatka pri deljenju (što se postiže jednim celobrojnim deljenjem, jednim množenjem i jednim oduzimanjem) i dve dodele vrednosti, u uniformnom sistemu bi ona trošila konstantno vreme. Međutim, Euklidov algoritam je dovoljno elementaran, pa ćemo na njega primeniti strogi logaritamski sistem. Zbog toga, najpre moramo na neki način "opravdati" obračunavanje složenosti celobrojnog deljenja.

Kako delimo dva broja? Pretpostavimo da su brojevi $a \geqslant b$ zapisani u nekom (na primer, decimalnom) sistemu. Najpre tražimo najkraći prefiks od a koji (posmatran kao zapis broja) nije manji od b. Ova operacija zahteva vreme $\mathcal{O}(\log^2 a)$, jer jedno poređenje posmatranog prefiksa i b podrazumeva jedno oduzimanje i poziv RAM komande JPOS. Kada smo pronašli traženi prefiks (ako takvog nema, celobrojni količnik je 0), prelazimo na formiranje prve cifre količnika. Da bismo to uradili, potrebno nam je da poznajemo jednocifrene umnoške od b: $0, b, 2b, \ldots, 9b$. Njih možemo dobiti uzastopnim sabiranjem (devet sabiranja), za šta nam je potrebno vreme $\mathcal{O}(\log b)$. Ako je m najmanja cifra sa osobinom da je $(m+1)b$ strogo veće od odabranog prefiksa, tada je m željena cifra, pa do nje dolazimo nakon najviše devet poređenja, dakle, $\mathcal{O}(\log a)$ elementarnih operacija. Postupak se iterativno nastavlja oduzimanjem mb od uočenog prefiksa i već poznatim "spuštanjem" dodatne cifre i dopisivanjem uz formiranu razliku. Na kraju postupka nam preostaje upravo $\mathrm{rest}(a, b)$.

Iz samog opisa postupka deljenja, jasno je da je broj iteracija $\mathcal{O}(\log a)$, pri čemu svaka iteracija troši vreme $\mathcal{O}(\log a)$. Stoga je realno da se za celobrojno deljenje a sa b, odnosno za nalaženje ostatka pri tom deljenju, obračuna vremenski utrošak od $\mathcal{O}(\log^2 a)$.

Prema tome, broj osnovnih operacija koji se izvode na ciframa u jednoj operaciji Euklidovog algoritma možemo proceniti sa $\mathcal{O}(\log^2 x)$. Preostaje da ograničimo sam broj iteracija. Za to je dovoljno je dokazati sledeće tvrđenje.

Lema 4.1 *Za sve $i \geqslant 0$, u Euklidovom algoritmu važi*

$$r_{i+2} < \frac{r_i}{2}.$$

Dokaz. Podsetimo se da je jednakost koja povezuje r_i i r_{i+2} sledeća:

$$r_i = q_{i+2} r_{i+1} + r_{i+2}.$$

Razmatramo dva slučaja. Ako je $r_{i+1} \leqslant \frac{r_i}{2}$, tada iz $r_{i+2} < r_{i+1}$ neposredno sledi tvrđenje leme. Zato pretpostavimo da je $r_{i+1} > \frac{r_i}{2}$. U tom slučaju važe nejednakosti

$$1 < \frac{r_i}{r_{i+1}} < 2,$$

zbog čega je $q_{i+2} = 1$ (jer je u pitanju zapravo ceo deo gornjeg količnika). Tako, sledi da je

$$r_i = r_{i+1} + r_{i+2},$$

odakle je $r_{i+2} = r_i - r_{i+1} < r_i - \frac{r_i}{2} = \frac{r_i}{2}$, što je i trebalo pokazati. □

Na osnovu gornje leme sledi zaključak da se u dve iteracije Euklidovog algoritma (pri čemu se sa (x, y) prelazi na (r_0, r_1), odnosno sa (r_i, r_{i+1}) na (r_{i+2}, r_{i+3})), par brojeva koje delimo jedan drugim *bar prepolovi*. Stoga je ukupan broj iteracija $\leqslant \log_2 x$.

Zaključujemo da je vremenska složenost Euklidovog algoritma $\mathcal{O}(\log x)$ u uniformnom sistemu, $\mathcal{O}(\log^2 x)$ u osnovnom, a $\mathcal{O}(\log^3 x)$ u striktnom logaritamskom sistemu obračunavanja.

Zadaci.

1. Napisati RAM program koji realizuje Euklidov algoritam.

4.2 Množenje matrica

Neka su date dve kvadratne matrice $A = [a_{ij}]$, $B = [b_{ij}]$, formata $n \times n$, čiji su elementi celi brojevi. Po samoj definiciji proizvoda matrica $C = AB = [c_{ij}]$, imamo da je

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

Prema tome, za formiranje jednog elementa proizvoda potrebno je n množenja i $n-1$ sabiranja. U uniformnom sistemu obračuna, to podrazumeva vreme $\mathcal{O}(n)$, dok je utrošak vremena u strogom logaritamskom sistemu $\mathcal{O}(n \log^2 M_{A,B})$, gde je

$$M_{A,B} = \max\{|a_{ij}|, |b_{ij}| : \ 1 \leqslant i, j \leqslant n\}$$

najveći po apsolutnoj vrednosti element matrica A i B. Kako elemenata ima n^2, to u ovim sistemima izračunavanje proizvoda dve matrice po definiciji traje respektivno $\mathcal{O}(n^3)$, odnosno $\mathcal{O}(n^3 \log^2 M_{A,B})$.

Ipak, postoji način da se dve matrice pomnože u asimptotski boljem vremenu, koristeći tzv. tehniku "podeli i vladaj"[14]. Ideja se sastoji u tome da najpre nađemo način da pomnožimo dve matrice konkretnog formata koristeći pri

[14] U daljem ćemo koristiti isključivo uniformni sistem obračuna, uz napomenu da prelazak na strogi logaritamski sistem podrazumeva množenje svih dobijenih vremenskih složenosti faktorom $\log^2 M_{A,B}$.

tom *manje množenja* nego u klasičnom algoritmu. To pruža mogućnost da naše proizvoljne matrice podelimo na blokove i množimo ih po uočenoj šemi, pa da tako, primenjujući rekurzivnu strategiju izračunavanja proizvoda, dobijemo bolje rezultate u pogledu vremenske složenosti.

Upravo to je 1969. uradio nemački matematičar *Volker Strassen* (Folker Štrasen, 1936) [41]. Naime, klasičnim putem, množenje dve matrice 2×2 zahteva 8 množenja i 4 sabiranja. Štrasen je, međutim, pronašao način da isti posao obavi sa samo 7 množenja i, doduše, sa 18 sabiranja. On je uočio da važi

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} s_1 + s_2 - s_4 + s_6 & s_4 + s_5 \\ s_6 + s_7 & s_2 - s_3 + s_5 - s_7 \end{bmatrix},$$

gde je

$$\begin{aligned} s_1 &= (b-d)(g+h), \\ s_2 &= (a+d)(e+h), \\ s_3 &= (a-c)(e+f), \\ s_4 &= h(a+b), \\ s_5 &= a(f-h), \\ s_6 &= d(g-e), \\ s_7 &= e(c+d). \end{aligned}$$

Sada je (radi jednostavnosti) dovoljno pretpostaviti da je n stepen dvojke (u praksi zaokružujemo n na prvi veći stepen dvojke). Ako, dakle, treba da pomnožimo dve $n \times n$ matrice, možemo da ih razbijemo na po 4 bloka $\frac{n}{2} \times \frac{n}{2}$, na koje rekurzivno primenjujemo gornji postupak:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} S_1 + S_2 - S_4 + S_6 & S_4 + S_5 \\ S_6 + S_7 & S_2 - S_3 + S_5 - S_7 \end{bmatrix},$$

gde je

$$\begin{aligned} S_1 &= (B-D)(G+H), \\ S_2 &= (A+D)(E+H), \\ S_3 &= (A-C)(E+F), \\ S_4 &= H(A+B), \\ S_5 &= A(F-H), \\ S_6 &= D(G-E), \\ S_7 &= E(C+D). \end{aligned}$$

Ako sada sa $T(n)$ označimo vreme potrebno za množenje $n \times n$ matrica po opisanom algoritmu, sledi da je

$$T(n) = 7T\left(\frac{n}{2}\right) + \mathcal{O}(n^2)$$

(kvadratni član potiče od matričnih sabiranja). Nakon smena $k = \log_2 n$ i $t_k = T(2^k)$, ova rekurentna relacija se lako rešava teleskopiranjem, pa sledi da je

$$T(n) = \mathcal{O}(n^{\log_2 7}) = \mathcal{O}(n^{2,81\cdots}).$$

Opisano rezonovanje se, naravno, može uopštiti: kad god nađemo način da dve matrice $c \times c$ pomnožimo koristeći d skalarnih množenja, to daje osnovu za rekurzivni algoritam čija je složenost $\mathcal{O}(n^{\log_c d})$. Na primer, za $c = 3$, kako bismo poboljšali Štrasenov algoritam, bilo bi potrebno da bude $d \leqslant 21$. Međutim, danas najbolji poznat algoritam za množenje 3×3 matrica koristi 23 množenja. S druge strane, *Viktor Pan* je 1978. godine [32] pronašao način da pomnoži dve matrice 70×70 koristeći 143640 množenja (!!!), što daje asimptotsku složenost $\mathcal{O}(n^{2,795\cdots})$.

4.3 **Problem** HORNSAT

SAT je problem odlučivanja iz oblasti iskazne logike. Sa istorijskog, teorijskog i praktičnog stanovišta, to je jedan od najznačajnijih algoritmaskih problema. Njegova formulacija je sledeća:

ULAZ: Iskazna formula ϕ data u konjuktivnoj normalnoj formi (KNF).

IZLAZ: Da li je ϕ zadovoljiva?

Podsetimo, za ϕ kažemo da je zadovoljiva ako postoji valuacija

$$\tau : X_\phi \to \{\top, \bot\}$$

u odnosu na koju je vrednost te formule tačna, $v_\tau(\phi) = \top$ (gde je X_ϕ skup svih iskaznih slova koja se pojavljuju u ϕ). Takođe, za formulu ϕ kažemo da je u KNF ako je u pitanju konjunkcija

$$\bigwedge_{i=1}^{k} C_i,$$

gde su podformule C_i — zvane *konjunkti* — disjunkcije literala (pod *literalom* podrazumevamo iskazno slovo ili njegovu negaciju). Pri tome, često koristimo

notaciju

$$x^{\alpha} = \begin{cases} x & \alpha = \top, \\ \neg x & \alpha = \bot. \end{cases}$$

Nije teško videti da KNF ϕ ima vrednost $\top$ u nekoj valuaciji ako i samo ako svaki konjunkt sadrži bar jedan literal koji je tačan u odnosu na tu valuaciju.

Ovaj problem je poseban stoga što za njega (zasad) *nije poznat* efikasan, polinomni algoritam, i što je to zapravo najstariji problem za koji pokušaji da se takav algoritam nađe nisu urodili plodom (naravno, "pravolinijsko" *brute force* rešenje koje podrazumeva isprobavanje svih mogućih valuacija u opštem slučaju troši vreme koje je eksponencijalna funkcija u odnosu na dužinu date formule). Nalaženje odgovarajućeg polinomnog algoritma bi automatski rešilo problem $\mathbf{P} = \mathbf{NP}$ (vidi Glavu 6), jedan od od 7 najznačajnijih otvorenih matematičkih problema po izboru *Fondacije Clay* iz Bostona (za rešenja svakog od njih, ta fondacija je raspisala nagradu od po 1.000.000 US$). S druge strane, dokaz da *ne postoji* polinomni algoritam za rešenje ovog problema bi za posledicu imao $\mathbf{P} \neq \mathbf{NP}$. Jednom rečju, pitanje egistencije polinomnog algoritma za SAT identifikovano je kao jedan od najrelevantnijih nerešenih problema savremene matematike.

Međutim, često se dešava da specijalni slučajevi računski složenih problema ipak imaju polinomno rešenje. To znači da algoritamski problem ostaje nepromenjen, već se samo sužava skup *instanci* — svih potencijalnih ulaznih podataka. Takav slučaj možemo prikazati upravo u slučaju problema SAT: ispostavlja se da je za KNF specijalnog oblika lakše ustanoviti zadovoljivost formule nego u opštem slučaju. Tako dolazimo do problema $\mathrm{HORNSAT}$.

Za KNF ϕ kažemo da je *Hornova formula* ako se u svakom konjunktu pojavljuje najviše jedan literal bez negacije. Na taj način, sve konjunkte možemo podeliti na dve grupe:

- *negativni* konjunkti, koji sadrže sve negirane literale,
- *pozitivni* konjunkti, koji sadrže tačno jedan ne-negiran literal, dakle:
  - konjukti koji se poklapaju sa nekim iskaznim slovom,
  - konjunkti oblika $\neg x_1 \vee \ldots \vee \neg x_m \vee y$ koje zovemo *implikacije* (pošto su ekvivalentni formulama oblika $(x_1 \wedge \ldots \wedge x_m) \Rightarrow y$, kako ćemo ove konjunkte od sada i pisati).

Prema tome, kada razmatramo Hornovu KNF ϕ, uvek je možemo pisati kao $\phi_- \wedge \phi_+$, gde smo sve negativne konjunkte grupisali u ϕ_-, a pozitivne u ϕ_+.

Problem HORNSAT sada definišemo kao problem SAT čiji je skup instanci sužen na Hornove formule. U narednom algoritmu za rešavanje ovog problema, formiraćemo jednu specifičnu valuaciju putem koje se odlučuje zadovoljivost date formule. Pri tome ćemo valuaciju, umesto preslikavanja $\tau : X_\phi \to \{\top, \bot\}$, posmatrati kao skup $T = \{x \in X_\phi : \ \tau(x) = \top\}$[15].

Algoritam za HORNSAT**.**

1. $T := \varnothing$.
2. Potraži u ϕ_+ prvi konjunkt sleva koji u odnosu na trenutnu valuaciju T ima vrednost $\bot$. Ako takvog konjunkta nema, pređi na korak 5.
3. Ako je taj konjunkt oblika y ili $(x_1 \wedge \ldots \wedge x_m) \Rightarrow y$,
$$T := T \cup \{y\}.$$
4. Vrati se na korak 2.
5. Na kraju je dobijena valuacija T_0. Ako ϕ ima vrednost $\top$ u odnosu na T_0, tada je ϕ zadovoljiva; u suprotnom — nije.

Najpre moramo dokazati korektnost ovog algoritma, koja nije očita. Ona će biti posledica sledećeg tvrđenja.

Teorema 4.2 *Hornova KNF ϕ je zadovoljiva ako i samo ako valuacija T_0 iz gornjeg algoritma zadovoljava ϕ.*

Najpre nam je neophodan sledeći pomoćni rezultat.

Lema 4.3 *Ako je T' valuacija koja zadovoljava ϕ_+, onda je $T_0 \subseteq T'$.*

Dokaz. Pretpostavimo, suprotno tvrđenju leme, da postoji valuacija T' koja zadovoljava ϕ_+, ali pri tome $T_0 \not\subseteq T'$. Neka je
$$\varnothing = T_1, T_2, \ldots, T_\ell = T_0$$
niz svih valuacija formiranih uzastopnom primenom koraka 3 iz gornjeg algoritma. Kako je $T_1 \subseteq T'$, to postoji najmanji indeks r za koji je $T_r \subseteq T'$, ali $T_{r+1} \not\subseteq T'$. Međutim, T_r i T_{r+1} se razlikuju samo za jedno iskazno slovo, $T_{r+1} = T_r \cup \{z\}$, pa zaključujemo da

[15] Razlog za ovo je čisto tehničke prirode, radi lakšeg zapisivanja dokaza korektnosti sledećeg algoritma, a u praksi valuaciju svakako implementiramo kao binarni niz dužine d, gde je d broj različitih iskaznih slova u datoj formuli.

- $z \notin T'$,
- prilikom r-te primene koraka 3, konjunkt oblika $(x_1 \wedge \ldots \wedge x_m) \Rightarrow z$ imao je vrednost $\bot$, pa je zato $x_1, \ldots, x_m \in T_r \subseteq T'$.

Ova dva tvrđenja povlače da konjunkt $(x_1 \wedge \ldots \wedge x_m) \Rightarrow z$ ima vrednost $\bot$ u odnosu na T', što je u kontradikciji sa pretpostavkom da T' zadovoljava ϕ_+. Zbog toga, tvrđenje leme mora biti tačno. □

Dokaz Teoreme 4.2. Implikacija ($\Leftarrow$) je trivijalna. Zato dokazujemo sledeće tvrđenje: ako T_0 ne zadovoljava Hornovu KNF ϕ, tada formula ϕ nije zadovoljiva.

Kako je valuacija T_0 konstruisana tako da ona sigurno zadovoljava ϕ_+ (jer je to jedini način da u koraku 2 izađemo iz petlje i okončamo rad algoritma, a to će se desiti nakon konačno mnogo iteracija, pošto $T = X_\phi$ sigurno zadovoljava ϕ_+), naša pretpostavka implicira da T_0 ne zadovoljava ϕ_-. Dakle, postoji negativni konjunkt, npr. $\neg x_1 \vee \ldots \vee \neg x_s$, koji ima vrednost $\bot$ u valuaciji T_0. To znači da $x_1, \ldots, x_s \in T_0$. Ako bi posotjala valuacija T' koja zadovoljava ϕ, tada bi po prethodnoj lemi bilo $T_0 \subseteq T'$, pa bi važilo $x_1, \ldots, x_s \in T'$. Ali, tada bi već uočeni negativni konjunkt imao vrednost $\bot$ i u T'. Kontradikcija. □

Preostaje još da procenimo složenost našeg algoritma. Za to je dovoljno uočiti da izračunavanje istinitosne vrednosti iskazne formule u odnosu na datu valuaciju (pa tako i bilo koje njene podformule) zahteva linearno vreme u odnosu na n, dužinu date formule. Tako, korak 1 zahteva konstantno vreme, a korak 5 vreme $\mathcal{O}(n)$, što isto važi i za jednu pojedinačnu iteraciju definisanu koracima 2–4. Jasno, iteracija može biti najviše onoliko koliko ϕ_+ ima konjunkta, jer svaki (netačan) konjunkt koji se obrađuje u koraku 2, nakon transformacije u koraku 3 postaje tačan, i *ostaje* takav do kraja algoritma. Prema tome, broj iteracija je $\mathcal{O}(n)$, stoga ovaj algoritam radi u vremenu $\mathcal{O}(n^2)$.

4.4 Reprezentacije grafova

Orijentisani graf je struktura $\mathcal{G} = (V, E)$, gde je V neprazan skup i $E \subseteq V \times V$ binarna relacija na V. Elementi skupa V se nazivaju *čvorovi*, dok skup E čine *grane*. Grafovi se najčešće vizuelno prikazuju tako što su čvorovi predstavljeni tačkama (u ravni, prostoru, itd.), dok za dva čvora $u, v \in V$ činjenicu da je $(u, v) \in E$ reprezentujemo neprekidnom linijom koja spaja u i v, i ona je pri tome orijentisana strelicom od u ka v. Mi ćemo se u ovoj knjizi baviti isključivo

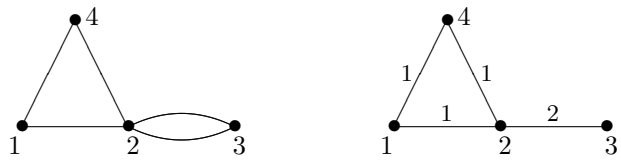
konačnim grafovima, pa će se u daljem podrazumevati da su skupovi V i E konačni.

Grafovi predstavljaju veoma značajan objekat istraživanja u teorijskoj matematici, ali su nezaobilazni u primenjenoj matematici, informatici i mnogim prirodnim naukama i tehničkim disciplinama. Razlog za to je vrlo jednostavan: prvenstveni model za razne odnose među podacima i drugim entitetima jeste apstraktni pojam *relacije*. Većina relacija koje razmatramo u praksi su binarne relacije. Grafovi stoga prestavljaju najjednostavniji matematički model interakcije kao fenomena kojeg svakodnevno opažamo.

Kod nekih pojava i odnosa koje modeliramo grafovima orijentacija grane nije bitna. Zbog takvih situacija uvodimo koncept *neorijentisnog grafa*. Imamo dva načina da definišemo neorijentisan graf. Jedan je da pod neorijentisanim grafom podrazumevamo orijentisan graf kod koga je E simetrična relacija: ako je $(u, v) \in E$, tada je i $(v, u) \in E$. Drugi način je da definišemo granu kao *neuređen par* $\{u, v\}$, pa je tada E podskup od $\mathcal{P}(V)$ sa osobinom da svaki element od E ima jedan ili dva elementa.

U teoriji grafova je uobičajeno da se pod terminom *graf* podrazumeva neorijentisan graf, dok se njegova orijentisanost posebno naglašava. Orijentisani grafovi se zovu još i *digrafovi*. Grane se često kraće označavaju i sa uv, umesto sa (u, v), odnosno $\{u, v\}$ (pri čemu je redosled bitan samo kod orijentisanih grafova). Graf je *prost* ako nema petlji, tj. grana oblika uu.

U primenama su takođe izuzetno značajni *težinski grafovi*. U pitanju su strukture oblika $\mathcal{G} = (V, E, w)$, gde je (V, E) prost graf, a $w : E \to X$ težinska funkcija. Njene vrednosti su u *skupu težina* X, koji je — iako može u načelu biti proizvoljan — najčešće uređen, a ne retko je na X definisana i neka algebarska struktura koja omogućava izvođenje osnovnih operacija na težinama. U velikom broju slučajeva, X ima strukturu (uređenog) poluprstena. Tipični primeri u tom smislu su $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, Bulov poluprsten $\mathbb{B}_2$ i drugi. Težine grana se uobičajeno ucrtavaju neposredno do krivih koje reprezentuju grane. Primetimo da se težinski grafovi sa težinama iz $\mathbb{N}$ mogu identifikovati sa *grafovima sa višestrukim granama*, kod kojih je dozvoljeno da neka grana "više puta" bude element multiskupa E.



Slika 4.4.1. Neorijentisani graf sa višestrukim granama i odgovarajući težinski graf

Kako bismo mogli da posmatramo algoritme na grafovima i njihove implementacije na RAM mašinama, veoma je pogodno da grafove reprezentujemo strukturama podataka koje će omogućiti laku manipulaciju. Tako, grafove možemo prikazati preko *matrica susedstva* i *lista susedstva*.

Za dati orijentisani graf $\mathcal{G} = (V, E)$, $|V| = n$, definišemo matricu susedstva $A_{\mathcal{G}} = [a_{ij}]$ formata $n \times n$, tako da je

$$a_{ij} = \begin{cases} 1 & (i,j) \in E, \\ 0 & \text{inače.} \end{cases}$$

Pri tome smo (bez umanjenja opštosti) identifikovali V sa skupom $\{1, \ldots, n\}$. Za neorijentisane grafove će — u skladu sa našim definicijama — matrica susedstva biti simetrična.

"Elegantnost" ove reprezentacije leži, između ostalog, i u sledećem poznatom tvrđenju iz teorije grafova.

Lema 4.4 *Ako je A matrica susedstva (orijentisanog) grafa $\mathcal{G}$ i $d > 0$ prirodan broj, tada je element na (i,j)-tom polju matrice A^d (pri čemu se sva sabiranja i množenja vrše u $\mathbb{N}$) jednak broju različitih puteva dužine d u $\mathcal{G}$ koji vode iz čvora i u čvor j.*

Na isti način (uz tvrđenje analogno gornjoj lemi) možemo definisati i matricu susedstva grafova sa višestrukim granama, pri čemu a_{ij} označava višestrukost grane (i,j).

Reprezentacija preko matrice susedstva se može primeniti i na težinske grafove, ali uz malu modifikaciju. Naime, ako je X uređeni skup težina grafa $\mathcal{G}$ (sa najmanjim elementom 0) i $\infty \notin X$, tada definišmo da je (i,j)-ti element matrice $A_{\mathcal{G}}$ jednak

$$a_{ij} = \begin{cases} 0 & i = j, \\ w(i,j) & (i,j) \in E, \\ \infty & \text{inače.} \end{cases}$$

Ovim želimo da izrazimo da je težina nepostojeće grane "beskonačna": uobičajeno je da relaciju poretka na X proširujemo sa $x < \infty$ za sve $x \in X$. Iz tog razloga, za svaki težinski graf možemo u suštini pretpostaviti da je u pitanju kompletan prost graf sa totalnom težinskom funkcijom $w : V^2 \to X \cup \{\infty\}$.

Glavni nedostatak matrične reprezentacije grafova je relativna prostorna (memorijska) neekonomičnost. Naime, kada su u pitanju grafovski algoritmi, standardno se za meru veličine grafa $\mathcal{G} = (V, E)$ kao ulaznog podatka uzimaju

dva broja: $n = |V|$ i $m = |E|$. Naravno, pri tome je $m = \mathcal{O}(n^2)$, ali u opštem slučaju (a i najčešće u praksi) broj grana u grafu može biti i znatno manji od reda veličine n^2. Kod težinskih grafova imamo i treću meru: to je ukupan broj cifara težina (ukoliko su one celobrojne), ali je ta veličina u uniformnom sistemu obračuna vremena retko kad relevantna. Sada je očito da skladištenje informacije o grafu u formi matrice zahteva n^2 memorijskih jedinica, tj. nezavisno je od broja grana. U praksi, to znači čuvanje ogromnog broja nula u memoriji i poneke jedinice u "moru" nula. Zbog toga je za velike i "retke" grafove mnogo pogodnija reprezentacija koja koristi *liste susedstva*.

Za svaki čvor $u \in V$ grafa $\mathcal{G}$, $Adj(u)$ je lista čiji su elementi svi susedi od u (ovde razmatramo neorijentisani slučaj, dok za orijentisane grafove treba da za svaki čvor imamo po dve liste susedstva: ulaznu i izlaznu). Preciznije, $Adj(u)$ je pokazivač koji pokazuje na prvi element liste, a element liste je slog koji sadrži informaciju o susedu v, eventualno težinu grane uv (ako je graf težinski) pokazivač na sledeći element liste i, po potrebi, "povratni" pokazivač (eng. *back-pointer*) na $Adj(u)$. U ovom pristupu, ukupan broj slogova i pokazivača je $n + m$ ($n + 2m$ u orijentisanom slučaju), tako da se graf pamti uz utrošak memorije $\mathcal{O}(n + m)$.

Prema tome, ako je $m \ll n^2$, ova reprezentacija grafova je memorijski mnogo ekonomičnija, a osim toga omogućava da se brzo pristupi svim susedima nekog čvora i, u krajnjoj liniji, isključivo baratanje pokazivačima kao najefikasnijim oblikom programiranja grafovskih problema. S druge strane, u matričnoj reprezentaciji se trenutno (u jednom koraku) odlučuje da li za dati par čvorova (i, j) postoji grana ij, dok isti posao u reprezentaciji preko lista zahteva linearno vreme $\mathcal{O}(n)$. Izbor reprezentacije za dati problem predstavlja, naravno, stvar ukusa, i treba da u načelu odgovori praktičnim potrebama proizašlim iz prirode i strukture samog problema.

Zadaci.

1. U poznatom filmu *Good Will Hunting*, Matt Damon tumači problematičnog mladog marginalca Vila koji radi kao čistač na MIT-u, a inače je vanserijski talenat za matematiku. Na početku filma (4:57 na DVD snimku koji posedujem), Vil (čisteći hodnik ispred amfiteatra) opaža domaći zadatak zapisan na maloj tabli u hodniku. U tom zadatku, dat je graf sa Slike 4.4.1 (levo) i četiri pitanja vezana za njega. Prva dva su:

1. Odrediti matricu susedstva datog grafa.
2. Odrediti matricu koja predstavlja broj svih puteva dužine 3 u tom grafu.

(Druga dva zadatka su u vezi generativnih funkcija za puteve u ovom grafu.) Vil rešava ove zadatke i rešenja se "misteriozno" pojavljuju na tabli sledećeg jutra, na opšte zaprepašćenje studenata i profesora(?!). Izračunajte i vi (poput Vila) ove dve matrice! (Rešenje se može nakratko videti u filmu negde oko 8:30.)

2. Nakon uklanjanja težina iz grafa na desnoj strani slike 4.4.1, odrediti odgovarajuće liste susedstva.

4.5 Dostiživost u grafovima

Problem dostiživosti (eng. *reachability*) jedan je od najznačajnijih algoritamskih problema u teoriji grafova:

ULAZ: (Orijentisani) graf $\mathcal{G} = (V, E)$ i dva čvora $s, t \in V$.

IZLAZ: Da li u $\mathcal{G}$ postoji put $s \rightsquigarrow t$?

(Ovde oznake s i t potiču od engleskih termina *source* (izvor) i *target* (meta, cilj).) Nešto "zahtevnije" verzije ovog problema imaju kao ulaz graf $\mathcal{G}$ i izvor s, a izlazni podatak može biti skup svih čvorova u za koje postoji put $s \rightsquigarrow u$ (što je u neorijentisanom slučaju upravo komponenta povezanosti čvora s), ili čak funkcija rastojanja $d_s : V \to \mathbb{N} \cup \{\infty\}$, gde $d_s(u) = d(s, u)$ označava dužinu najkraćeg puta od s do u (ta dužina je ∞ ukoliko traženi put ne postoji). Konačno, voleli bismo da (kao "nusproizvod") odgovarajući algoritam za svaki čvor u identifikuje bar jedan najkraći put $s \rightsquigarrow u$.

Kao što smo to već naglasili, strukture podataka koje se najčešće koriste u grafovskim algoritmima su *liste*. Mi ovde nećemo detaljno ulaziti u implementaciju lista (naposletku, to i nije zadatak analize algoritama), već ćemo koristiti jednu uopštenu sintaksu za manipulaciju listama. Pošto je sama lista u principu pokazivač koji pokazuje njen prvi slog, *glava* liste je jedini podatak kojem u datom trenutku možemo neposredno da pristupimo. Ako je L posmatrana lista, čvor koji je zapisan u njenom prvom slogu dobijamo kao rezultat funkcije **head**(L). Osim toga, na listu možemo smeštati podatke (komandom **push**(L, x)), a takođe ih i uklanjati sa nje, tj. obrisati trenutnu glavu liste (komandom **pull**(L)), čime drugi po redu podatak (ako postoji) postaje nova glava.

Naravno, pri tome se postavlja pitanje hijerarhije novih elemenata u listi: nije svejedno da li ih smeštamo na početak ili kraj liste. U prvom slučaju, kažemo da je lista ustrojena po principu LIFO (*last in, first out*), i ona se tada naziva *stek* (eng. *stack*). U suprotnom, reč je o listi tipa FIFO (*first in, first out*), poznatijoj kao *red za opsluživanje* (eng. *queue*).

U najgrubljim crtama, ideja za polinomni algoritam koji rešava problem dostiživosti je sledeća. Na početku, lista L sadrži samo čvor s, i on je jedini "označen" čvor (ovo se realizuje jednom Bulovom funkcijom na V). Zatim, iterativno odabiramo jedan čvor sa liste L i uklanjamo ga sa nje. Inspekcijom liste susedstva tog čvora, pronalazimo sve njegove do tada neoznačene susede. Te susede označavamo, i stavljamo ih na L. Algoritam radi sve dok se L ne isprazni, kada označeni čvorovi predstavljaju upravo skup svih čvorova dostiživih iz s.

U zavisnosti od tipa liste koju koristimo, opisana pretraga može poprimiti sasvim različit tok. Ako za L koristimo stek, tada će ovaj algoritam najpre pronaći susede od s, opredeliti se za jednog od njih, zatim pronaći njegove susede, ponovo se opredeliti za jednog od njih, i tako redom, sve dok više nije moguće naći nijedan neoznačen čvor koji je susedan sa trenutno obrađivanim. Tada primenjujemo tehniku "vraćanja", *back-tracking*, vraćamo se na prethodno razmatrani čvor i posmatramo nekog drugog njegovog suseda (koji čeka na steku), itd. Na taj način, naizmeničnim otkrivanjima "u dubinu" i vraćanjima otkrivamo sve čvorove dostižive iz s. Stoga i nije čudno što se ovakva tehnika pretraživanja grafa zove *pretraživanje u dubinu*, tj. DFS (akronim od eng. *depth-first search*).

Sasvim je druga situacija ako se opredelimo da L bude red za opsluživanje, kada dobijamo *pretraživanje u širinu*, BFS (od *breadth-first search*). Ovde se ponovo polazi od s i otkrivaju se njegovi susedi. Zatim se za svaki od tih suseda ponaosob pronalaze njihovi do tada neotkriveni susedi, i tako redom. Važna primedba je da se ovom tehnikom rešava i proširen problem dostiživosti, pošto ona nalazi baš najkraće puteve iz s, i stoga omogućava algoritamsko izračunavanje funkcije rastojanja u grafu $\mathcal{G}$ (nije teško uveriti se da to nije slučaj sa DFS). Preciznije, imamo sledeći algoritam. U njemu, osim liste L, figurišu i funkcije $d : V \to \mathbb{N} \cup \{\infty\}$ i $new : V \to \{\top, \bot\}$, koje redom izračunavaju rastojanje od s i vode računa o označenosti čvorova ($new(u) = \bot$ će značiti da je u označen).

BFS algoritam za pretraživanje grafa $\mathcal{G} = (V, E)$.

1. $d(s) := 0$; $new(s) := \bot$; $L := [s]$.
2. Za sve $u \in V \setminus \{s\}$ radi:
   - $d(u) := \infty$;
   - $new(u) := \top$.

3. Dok je $L \neq \varnothing$ radi:

   a) u :=**head**(L);

   b) **pull**(L);

   c) za sve $v \in Adj(u)$:

      ako $new(v)$ onda:

      - $d(v) := d(u) + 1$;
      - $new(v) := \perp$;
      - **push**(L, v).

Kako bismo dokazali korektnost ovog algoritma, najpre nam treba jedno pomoćno tvrđenje.

Lema 4.5 *Ako je u gornjem algoritmu u nekom trenutku*

$$L = [u_1, \ldots, u_j],$$

tada je

$$d(u_1) \leqslant \ldots \leqslant d(u_j) \leqslant d(u_1) + 1.$$

Dokaz. Dovoljno je proveriti da se opisana "situacija" očuvava tokom celog algoritma, tj. da je invarijantna na sve promene sadržaja liste L. Na početku, L ima samo jedan element, pa su tražene nejednakosti trivijalno ispunjene. Zato pretpostavimo da je u nekom trenutku $L = [u_1, u_2, \ldots, u_j]$ i razmatrajmo dve mogućnosti. U koracima 3.a i 3.b uklanja se glava liste u_1. Drugim rečima, novi sadržaj liste L je $[u_2, \ldots, u_j]$. Sada imamo

$$d(u_2) \leqslant \ldots \leqslant d(u_j) \leqslant d(u_1) + 1 \leqslant d(u_2) + 1,$$

što smo i želeli da imamo.

S druge strane, drugi tip promene koju lista može da doživi jeste da joj se na kraj dopiše novi čvor (korak 3.c), tj. da umesto $[u_1, \ldots, u_j]$ ona postane $[u_1, \ldots, u_j, v]$. Ovo se dešava zbog toga što je sa liste upravo uklonjena njena malopređašnja glava u_0, a v je njen sused. Pri tome je $d(v) = d(u_0) + 1$. Stoga važe nejednakosti

$$d(u_0) \leqslant d(u_1) \leqslant \ldots \leqslant d(u_j) \leqslant d(u_0) + 1 = d(v) \leqslant d(u_1) + 1,$$

pa je dokaz leme okončan. □

Teorema 4.6 (Korektnost BFS) *Na kraju rada gornjeg algoritma, za sve $u \in V$ važi $d(u) = d_s(u)$, pri čemu $d(u) = \infty$ označava nepostojanje puta od s do u.*

Dokaz. Najpre dokazujemo indukcijom po i da ako je $d(u) = i$, tada postoji put $s \rightsquigarrow u$ dužine i. Za $i = 0$ ovo je jasno, pošto je s jedini čvor koji dobija vrednost 0 tokom gornjeg algoritma (vidi korake 2 i 3.c). S druge strane, ako je $i > 0$, tada je čvor u mogao biti označen ovim brojem samo tokom koraka 3.c, i to tako što je u tom trenutku bio aktivan čvor u' čiji je u sused, pri čemu je bilo $d(u') = i - 1$. Po induktivnoj pretpostavci, to znači da postoji put $s \rightsquigarrow u'$ dužine $i - 1$, pa je $s \rightsquigarrow u' \rightsquigarrow u$ put dužine i.

Sada preostaje da pokažemo da $d(u) = i > 0$ implicira da ne postoji put $s \rightsquigarrow u$ dužine manje od i. Pretpostavićemo suprotno: da postoji čvor $v \in V$ takav da je $d_s(v) = k < i = d(v)$, i da je pri tom k minimalno sa opisanom osobinom. Neka je u "pretposlednji" čvor na jednom uočenom najkraćem putu $s \rightsquigarrow v$. Međutim, svaki početni segment najkraćeg puta od s do nekog čvora je ujedno i najkraći put od s do "usputnih" čvorova, pa zaključujemo da je $d_s(u) = k - 1$. Po gornjem uslovu minimalnosti, mora biti $d(u) = k - 1$. Ako je u uklonjen sa L pre nego što je v otkriven, tada bi v bio otkriven kao sused čvora u, i bilo bi $d(v) = d(u) + 1 = k$, što nije slučaj. Zaključujemo da je v bio otkriven pre nego što je u postao "aktivan". Međutim, otkrivanje u nije moglo da se desi nakon "aktiviranja" v, jer bi u suprotnom u bio otkriven kao sused od v, i imali bismo

$$d(u) = d(v) + 1 = i + 1 > k - 1,$$

što je nemoguće. Dakle, jedina mogućnost je da su u jednom trenutku u i v bili istovremeno na listi L. Kako je

$$d(v) - d(u) = i - (k - 1) = i - k + 1 \geqslant 2,$$

imamo kontradikciju sa prethodnom lemom.

Kako smo iscrpli sve mogućnosti, zaključujemo da opisani čvor v ne postoji, tj. da važi $d_s(v) = d(v)$ za sve $v \in V$. □

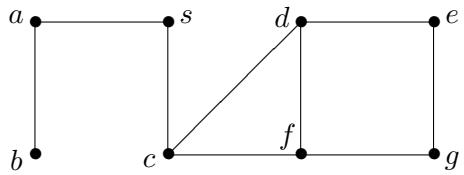
Na osnovu dokaza gornje teoreme je veoma lako opisati postupak kojim se za svaki čvor $t \in V$ koji je dostiživ iz s konstruiše bar jedan najkraći put $s \rightsquigarrow t$. Naime, ako je razmatrani BFS algoritam izračunao da je $d(t) = i > 0$, tada t mora da ima bar jednog suseda u takvog da je $d(u) = i - 1$. Odaberimo jedan takav čvor u_1. Sada u_1 ima suseda u_2 takvog da je $d(u_2) = i - 2$, itd. Na ovaj način se formira niz čvorova $u_1, u_2, \ldots$ tako da je $d(u_j) = i - j$ za sve

$1 \leqslant j \leqslant i$. Naravno, to uključuje i $d(u_i) = 0$; ali, kako je $d_s(x) = 0$ ako i samo ako je $s = x$, sledi $u_i = s$. Stoga je $s, u_{i-1}, \ldots, u_1, t$ put dužine i.

Preostaje da analizaramo vremensku složenost razmatranog BFS algoritma. Neka je (kao i u prethodnom odeljku) $|V| = n$ i $|E| = m$. Očigledno je da faza 1 našeg algoritma troši konstantno vreme. Takođe, korak 2 se realizuje u vremenu $\mathcal{O}(n)$. Što se tiče iteracije 3, njenu vremensku složenost bismo "lagodno" mogli da procenimo na $\mathcal{O}(n^2)$ (broj iteracija je ograničen brojem čvorova, dok svaka iteracija uključuje $a|Adj(u)| + b$ koraka, gde su a, b konstante, a u je "aktivan" čvor, pa je u pitanju $\mathcal{O}(n)$). Međutim, moguća je i nešto bolja procena. Naime faze 3.a i 3.b će biti realizovane onoliko puta koliko ima i iteracija. Međutim, primetimo da je broj elementarnih koraka koji će se sprovesti u tački 3.c *u svim iteracijama kumulativno* proporcionalan broju grana grafa $\mathcal{G}$: uvek razmatramo jedan "aktivan" čvor i njegove susede i u tom smislu će svaka grana biti obrađena samo jednom — naime, ako je otkriven čvor v iz čvora u, tada će u biti uklonjen sa L i nikada više neće tamo dospeti, pošto je $new(u) = \perp$ do kraja algoritma. Zbog toga je složenost koraka 3, a tako i celog algoritma, $\mathcal{O}(m + n)$ — linearna funkcija po m i n.

Zadaci.

1. "Ručno" primeniti algoritam za BFS pretragu sledećeg grafa (izvor je s):



Pri tome su sve liste susedstva su abecedno soritrane.

2. Za graf iz prethodnog zadatka, "ručno" primeniti DFS strategiju pretrage (koja se dobija kada se lista L implementira kao stek). Uporediti redosled otkrivanja čvorova tokom BFS i DFS pretraga.

4.6 Dajkstrin algoritam

Kao uopštenje problema dostiživosti u grafovima, možemo posmatrati *problem najkraćih puteva* u težinskom grafu $\mathcal{G} = (V, E, w)$. Pri tome radi jednostavnosti pretpostavljamo da su težine prirodni brojevi. Podsetimo se da w možemo posmatrati kao totalnu funkciju $V \times V \to \mathbb{N} \cup \{\infty\}$ tako da je $w(x, x) = 0$ za

sve $x \in V$, dok je $w(x, y) = \infty$ ako i samo ako $x \neq y$ i $\{x, y\} \notin E$ (odnosno $(x, y) \notin E$).

Problem najkraćih puteva *iz jednog izvora* je sledeći:

ULAZ: Težinski graf $\mathcal{G}$ i $s \in V$.

IZLAZ: Funkcija $d : V \to \mathbb{N} \cup \{\infty\}$ takva da je za sve $u \in V$,

$$d(u) = d(s, u),$$

dužina "najlakšeg" puta $s \rightsquigarrow u$.

Naime, za proizvoljnu kolekciju grana $X \subseteq E$ definišemo njenu težinu kao

$$w(X) = \sum_{e \in X} w(e),$$

pa se ova definicija odnosi i na puteve između dva čvora. U tom smislu se u gornjem problemu traži minimalna težina puta $s \rightsquigarrow u$. Primetimo da je (prošireni) problem dostiživosti u grafovima specijalan slučaj gornjeg problema, kada su težine svih grana istovetne (npr. sve su jednake 1).

Polinomni algoritam za rešavanje problema najkraćih puteva iz jednog izvora dao je *Edsger W. Dijkstra* (Edsher Dajkstra, 1930–2002) 1959. godine [8], a odlikuju ga elegancija i frapantna jednostavnost. Osim funkcije d, u njemu figuriše još i skup obrađenih čvorova X (nije teško videti da je ova struktura ekvivalentna sa Bulovom funkcijom new iz prethodnog odeljka, ali je zapis petlje iz koraka 3 donjeg algoritma ovako kompaktniji).

Dajkstrin algoritam.

1. $d(s) := 0$; $X := \{s\}$.
2. Za sve $u \in V \setminus \{s\}$ radi $d(u) := w(s, u)$.
3. Dok je $X \neq V$ radi:
   a) nađi $u \in V \setminus X$ tako da je $d(u)$ minimalno;
   b) $X := X \cup \{u\}$;
   c) za sve $v \in V \setminus X$ radi:

$$d(v) := \min(d(v), d(u) + w(u, v)).$$

Opisani iterativni način formiranja funkcije $d(u)$ primenjen u koraku 3.c (kojim se ta funkcija postepeno "poboljšava", tj. približava optimalnom rešenju) se u literaturi često naziva *relaksacija*.

Korektnost ovog algoritma biće direktna posledica naredna dva tvrđenja.

Lema 4.7 *U svakom koraku Dajkstrinog algoritma za sve $u \in X$, $v \notin X$ važi $d(u) \leqslant d(v)$.*

Dokaz. Lema će biti dokazana ako pokažemo da se tražena nejednakost očuvava pri svakoj promeni skupa X i svakoj promeni vrednosti funkcije d tokom Dajkstrinog algoritma.

Naravno, opisana nejednakost važi na početku algoritma, jer je $d(s) = 0$, dok je $d(u) > 0$ za sve $u \neq s$. Pretpostavimo sada da pre razmatrane promene (skupa X ili relaksacije funkcije d) imamo da je $d(u) \leqslant d(v)$ za sve $u \in X$, $v \notin X$. Nakon te promene, imaćemo skup X' i funkciju d'.

Skup X se menja u koraku 3.b, kada se u njega uključuje novi čvor. Označimo taj čvor sa u_0. Prema našoj pretpostavci, važi $d(u) \leqslant d(v)$ sa sve $u \in X$ i $v \in V \setminus X'$. Osim toga, zbog načina na koji je čvor u_0 izabran u koraku 3.a, imamo $d(u_0) \leqslant d(v)$ za sve $v \notin X$, $v \neq u_0$. Prema tome, posmatrana nejednakost neće biti narušena u koraku 3.b.

Do promene funkcije d dolazi u koraku 3.c: njena vrednost se modifikuje za (neke) susede v čvora u_0 koji je upravo uključen u X, tako što se za suseda $v \in Adj(u_0)$ koji ne leži u X takvog da je $d(u_0) + w(u_0, v) < d(v)$ za $d'(v)$ uzme baš $d(u_0) + w(u_0, v)$. Međutim, pošto je čvor u_0 upravo uključen u X', važi $d(u) \leqslant d(u_0)$ za sve $u \in X$. S druge strane, zbog izbora načinjenog u 3.a, važi $d(u_0) \leqslant d(v)$ za sve $v \in V \setminus X'$. Opisanom relaksacijom se tačnost ove poslednje nejednakosti očuvava, jer u slučaju $d'(v) \neq d(v)$ važi $d(u_0) \leqslant d(u_0) + w(u_0, v) = d'(v)$. Time je dokaz leme okončan. □

Lema 4.8 *U svakom koraku Dajkstrinog algoritma za sve $u \in X$ važi da je trenutna vrednost $d(u)$ dužina najkraćeg puta iz s do u koji prolazi isključivo kroz čvorove iz skupa X.*

Dokaz. Pošto je na početku $X = \{s\}$, tvrđenje leme je tada trivijalno zadovoljeno. Slično kao i u prethodnoj lemi, dokazaćemo da je tačnost njenog tvrđenja invarijatna na promene skupa X i funkcije d tokom realizacije algoritma. Zapravo, dovoljno je ograničiti se samo na promene skupa X, jer se u koraku 3.c menjaju vrednosti $d(v)$ za neke čvorove v koji u tom trenutku *ne* pripadaju skupu X, dok se nakon uključenja nekog čvora u u X vrednost $d(u)$ više ne menja.

Zbog toga, pretpostavljamo da u nekom trenutku važi tvrđenje leme i dokazujemo da ono važi i nakon sledećeg koraka u algoritmu u kojem je došlo do uključenja novog čvora u_0 u skup X.

Najpre, dokazujemo da je u tom trenutku vrednost $d(u_0)$ upravo dužina najkraćeg puta $s \rightsquigarrow u_0$ koji prolazi kroz čvorove iz X. Početna vrednost $d(u_0)$ je bila $w(s, u_0)$. Ukoliko je tokom algoritma ona bila bar jednom relaksirana, to znači da je u trenutku uključenja u_0 u X vrednost $d(u_0)$ bila jednaka $d(x) + w(x, u_0)$ za neko $x \in X$. Međutim, tada je $d(x)$ bilo jednako dužini najkraćeg puta $s \rightsquigarrow x$ koji prolazi kroz čvorove iz X. Stoga je pri uključenju u_0 u X, vrednost $d(u_0)$ jednaka dužini *nekog* puta $s \rightsquigarrow u_0$ koji prolazi kroz čvorove iz X.

Neka je sada (u tom istom trenutku)

$$s = x_0, x_1, \ldots, x_{n-1}, x_n = u_0$$

jedan od najkraćih puteva $s \rightsquigarrow u_0$, pri čemu je $x_0, x_1, \ldots, x_{n-1} \in X$ (primetimo da je tada $s = x_0, x_1, \ldots, x_{n-1}$ jedan od najkraćih puteva $s \rightsquigarrow x_{n-1}$ koji prolazi kroz čvorove iz X, čija je dužina, po pretpostavci, jednaka $d(x_{n-1})$). Prilikom uključenja čvora x_{n-1} u X (koje se ranije već moralo desiti), došlo je do relaksacije vrednosti d za njegove susede koji tada nisu pripadali X. Jedan od takvih suseda je bio baš u_0. Tada je za novu vrednost $d(u_0)$ definisano $\min(d(u_0), d(x_{n-1}) + w(x_{n-1}, u_0))$ (gde se $d(u_0)$ odnosi na prethodnu vrednost). Prema tome, vrednost $d(u_0)$ pri uključenju u_0 u X nije veća od $d(x_{n-1}) + w(x_{n-1}, u_0)$, a što je upravo dužina puta $s, x_1, \ldots, x_{n-1}, u_0$.

Rezimirajući, imamo da je $d(u_0)$ (u trenutku koji posmatramo) dužina nekog puta $s \rightsquigarrow u_0$ sa međučvorovima iz X, ali da ta dužina nije veća od dužine najkraćeg takvog puta. Dakle, $d(u_0)$ je baš dužina najkraćeg puta $s \rightsquigarrow u_0$ sa međučvorovima iz X.

Najzad, preostaje da pokažemo da se uključenjem u_0 u X tačnost tvrđenja leme nije promenila u odnosu na ostale čvorove $u \in X$. Naime, pre uključenja u_0, $d(u)$ je bila dužina najkraćeg puta $s \rightsquigarrow u$ sa međučvorovima iz X. Uključenjem u_0 u X mogu da nastanu novi putevi $s \rightsquigarrow u$. Međutim, takvi putevi imaju u_0 kao međučvor, pa je njihova dužina bar $d(u_0)$. U prethodnoj lemi smo videli da mora biti $d(u) \leqslant d(u_0)$. To znači da novonastali putevi $s \rightsquigarrow u$ ne mogu biti kraći od "starog" najkraćeg puta između ovih čvorova, pa je lema dokazana. □

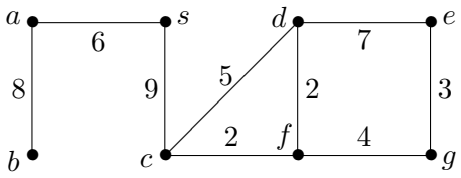
Iz gornje leme odmah dobijamo da su na kraju posmatranog algoritma izračunate vrednosti $d(u)$, $u \in V$, jednake dužinama najkraćih puteva iz s do u.

Kao i kod algoritma za BFS, koraci 1 i 2 troše ukupno vreme $\mathcal{O}(n)$. Broj iteracija 3 je takođe ograničen sa n, pošto se u svakoj iteraciji dodaje po jedan

novi čvor u skup X. Korak 3.b se realizuje u konstantnom vremenu, a koraci 3.a i 3.c, kao i provera uslova iteracije, u vremenu $\mathcal{O}(n)$. Prema tome, složenost Dajkstrinog algoritma[16] je $\mathcal{O}(n^2)$.

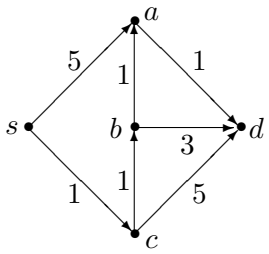
Zadaci.

1. "Ručno" primeniti Dajkstrin algoritam za sledeći neorijentisan težinski graf (izvor je s):



Rad algoritma prikazati kao matricu u kojoj su kolone označne čvorovima, dok se u i-toj vrsti nalazi trenutne vrednosti $d(u)$ nakon i-te iteracije, $u \in V$.

2. Analogno kao i u prethodnom zadatku, "ručno" primeniti Dajsktrin algoritam na sledeći orijentisan težinski graf:



4.7 Minimalna razapinjuća stabla u težinskim grafovima

Povezan graf koji ne sadrži cikluse zovemo *stablo*. Kao što je poznato iz teorije grafova, bilo koji od uslova povezanosti, odnosno acikličnosti, može se u ovoj definiciji ekvivalentno zameniti uslovom da posmatrani graf ima n čvorova i $n-1$ granu. Takođe, stabla možemo definisati kao u izvesnom smislu "maksimalne" aciklične grafove: dodavanje grane između bilo koja dva nesusedna

[16] Korišćenjem reprezentacije težinskih grafova preko lista susedstva, kao i nešto složenijih struktura podataka, moguće je postići složenost $\mathcal{O}(m + n \log n)$.

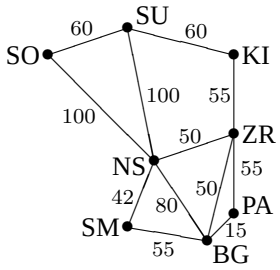
čvora rezultuje pojavom ciklusa. S druge strane, u pitanju su "minimalni" povezani grafovi: uklanjanje bilo koje od grana daje nepovezan graf (tj. svaka grana je *most*).

Neka je $\mathcal{G} = (V, E)$ proizvoljan (neorijentisan) graf. Njegov podgraf oblika $\mathcal{T} = (V, E')$ koji je stablo zovemo *razapinjuće stablo* od $\mathcal{G}$. Ako je pri tome $\mathcal{G}$ težinski graf, tada možemo posmatrati (kao i u prethodnom odeljku) težinu nekog njegovog razapinjućeg stabla kao sumu težina grana koje čine to stablo. Tako se nameće izuzetno značajan problem pronalaženja razapinjućeg stabla minimalne težine (MST, od eng. *minimal spanning tree*):

ULAZ: Povezan težinski graf $\mathcal{G} = (V, E, w)$.

IZLAZ: Težina minimalnog razapinjućeg stabla grafa $\mathcal{G}$.

Naravno, bilo bi od velike koristi da se tokom odgovarajućeg algoritma identifikuje bar jedno minimalno razapinjuće stablo (koje ne mora biti jedinstveno). Problem je korektno formulisan, pošto se indukcijom lako pokazuje da graf ima razapinjuće stablo ako i samo ako je povezan. S druge strane, ne bi predstavljao ozbiljan problem ni da je kao skup instanci navedena klasa svih težinskih grafova, pošto se povezanost može testirati (npr. pomoću BFS algoritma sa proizvoljnim izvorom) u linearnom vremenu.



Slika 4.7.1. Graf koji reprezentuje približna rastojanja Beograda i sedišta okruga u Vojvodini

Problem MST ima veliku praktičnu primenu u dizajniranju komunikacijskih i logističkih sistema uz minimalni utrošak komunikacione opreme, odnosno transportnih resursa. Na primer, mrežu većih gradova u nekom regionu ili državi sa odgovarajućom mrežom puteva možemo posmatrati kao težinski graf, budući da troškovi transporta neke robe između gradova A i B očito zavise (između ostalog) od njihovog rastojanja. Isto se može reći i ukoliko želimo da između posmatranih gradova položimo optičke kablove. Minimalno razapinjuće stablo razmatranog grafa tada definiše transportne koridore, odnosno trasu polaganja optičkog kabla, tako da se uz minimalni utrošak sredstava obezbedi prevoz robe (ili protok podataka) na datoj teritoriji.

Verovatno prvi algoritam za rešavanje problema MST dao je 1926. profesor Univerziteta u Brnu, *Otakar Borůvka* — motiv je bio elektrifikacija zapadne Moravske. 1930. godine, takođe češki matematičar *Vojtěch Jarník* (1897–1970) našao je znatno jednostavniji algoritam. *Joseph Kruskal* (Džozef Kraskal, 1929) [23] je 1956. dao je jedan takođe veoma prost algoritam za MST, a godinu dana kasnije *Robert C. Prim* (1921) je ponovo otkrio Jarníkov algoritam[17]. U teoriji grafova su najpoznatija ova poslednja dva algoritma, pa ćemo ih i mi ovde razmotriti. Oni, naravno, daju isti rezultat, ali to postižu različitim algoritamskim strategijama. Ne retko se dešava da oni nalaze dva *različita* minimalna razapinjuća stabla datog težinskog grafa.

Kraskalov algoritam za MST**.**

1. Sortiraj grane po težini u neopadajućem poretku.
2. Za svaku granu e sa sortirane liste, uključi e u skup grana F (koji će na kraju formirati minimalno razapinjuće stablo) ukoliko ona ne čini ciklus sa već ranije odabranim granama. U suprotnom, odbaci e.
3. Ukoliko još nije odabrano $n-1$ grana, pređi na narednu granu sa sortirane liste i ponovi korak 2, a u suprotnom stani.

Jarník-Primov algoritam za MST**.**

1. Izaberi proizvoljan čvor $v_1 \in V$.
2. Ako su do sada odabrani čvorovi $\{v_1, \ldots, v_{i+1}\}$, odaberi granu e minimalne težine sa osobinom da je incidentna sa čvorovima $v_j \in \{v_1, \ldots, v_{i+1}\}$ i $x \notin \{v_1, \ldots, v_{i+1}\}$. Uključi granu e u skup F i dodaj $v_{i+2} := x$ spisku odabranih čvorova.
3. Ukoliko još nije odabrano $n-1$ grana, ponovi korak 2, a u suprotnom stani.

Oba ova algoritma pripadaju tipu "gramzivih" (eng. *greedy*) algoritama, gde se željeni objekat formira iterativno, dodavanjem novih (u tom trenutku najpovoljnijih) elemenata u svakom koraku, bez "povratnih" koraka (tj. bez relaksacije, kao što je to bio slučaj u Dajkstrinom algoritmu). U oba ova algoritma, nakon svake pojedinačne iteracije 2–3 (a i pre nje) imamo razapinjuću *šumu*

[17] Naravno, Prim je ovaj algoritam otkrio ne znajući za rezultate Jarníka. Treba imati u vidu da su 1957. mogućnosti komunikacije i pretraživanja naučne literature bile drastično manje nego danas.

$\mathcal{F} = (V, F)$ od $\mathcal{G}$, gde je $F \subseteq E$. Podsetimo, šuma je acikličan graf, bez uslova povezanosti (reč je zapravo o uniji stabala).

Na početku, u oba algoritma, šuma od koje polazimo je $(V, \varnothing)$. Ona u tom trenutku ima $|V|$ komponenti: svaki čvor čini komponentu za sebe. Nakon svakog odabiranja nove grane, broj komponenti šume se smanjuje za jedan, pošto se odabirom te grane dve komponente spajaju u jednu. Tu nastupa "strateška" razlika između Kraskalovog i Jarník-Primovog algoritma. Naime, geometrijski posmatrano, kod Kraskalovog algoritma stabla-komponente koja čine razmatranu razapinjuću šumu rastu "haotično", budući da kao "reper" imamo uređenu listu svih grana težinskog grafa $\mathcal{G}$, i zato je svaka grana u svakom trenutku potencijalni element razapinjućeg stabla kojeg gradimo. S druge strane, situacija je mnogo uređenija u slučaju Jarník-Primovog algoritma: ovde je rast šume takoreći "centralizovan" u odnosu na prvi čvor v_1. U svakom trenutku algoritma, jedino komponenta kojoj pripada v_1 može biti netrivijalno stablo, dok su sve ostale komponente trivijalne (sastoje se od po jednog čvora). Kasnije se jedan po jedan izolovani čvor pridodaje toj jedinoj netrivijalnoj komponenti, koja na kraju prerasta u MST.

1983. godine, *Robert E. Tarjan* (Robert Tardžan, 1948) [43] je primetio da su kako dva posmatrana algoritma, tako i niz drugih algoritama koji nalaze MST za dati težinski graf specijalni slučajevi jedne opštije šeme, koju je on nazvao *plavo-crveni algoritam*. Naime, on posmatra proces odabira grana i njihovog uključivanja u razapinjuće stablo kao bojenje grana grafa u dve boje: plavu i crvenu (pri tome će plave grane činiti traženo minimalno razapinjuće stablo, dok će prostale grane biti crvene). Tokom bojenja, primenjuju se dva pravila.

> ***Plavo pravilo:*** Nađi u $\mathcal{G}$ *rez*, neprazan skup čvorova $X \subset V$ tako da nijedna grana koja spaja čvorove iz X i $V \setminus X$ nije plava. Među svim neobojenim granama koje spajaju čvorove iz X i $V \setminus X$ odaberi onu najmanje težine i oboji je u plavo.
>
> ***Crveno pravilo:*** Nađi u $\mathcal{G}$ ciklus u kome nijedna grana nije crvena. Među svim neobojenim granama tog ciklusa odaberi onu najveće težine i oboji je u crveno.

Na početku algoritma, nijedna grana posmatranog težinskog grafa nema boju. Zatim primenjujemo plavo i crveno pravilo *u proizvoljnom redosledu* dogod je to moguće. Tardžan je dokazao sledeći naoko iznenađujući rezultat.

Teorema 4.9 (R.E.Tarjan, 1983) *Kako god primenjivali plavo i crveno pravilo, plavo-crveni postupak će uvek rezultovati bojenjem svih grana datog težin-*

skog grafa $\mathcal{G}$ i pri tome podgraf određen plavim granama čini minimalno razapinjuće stablo od $\mathcal{G}$ (štaviše, ovo potonje tvrđenje je tačno već nakon bojenja $(|V|-1)$-ve grane u plavo, kada neke grane možda još nisu obojene).

Dokaz. Za delimično (parcijalno) bojenje grana grafa $\mathcal{G}$ (tako da grane mogu biti plave, crvene, ili bezbojne) kažemo da je *ekonomično* ako $\mathcal{G}$ ima minimalno razapinjuće stablo koje sadrži sve plave i nijednu crvenu granu u uočenom bojenju. Ideja dokaza je da indukcijom pokažemo da je svako parcijalno bojenje koje nastaje tokom plavo-crvenog algoritma ekonomično. Jasno, ovo je trivijalno tačno na početku (nijedna grana nije obojena), pa je dovoljno dokazati da primena bilo plavog, bilo crvenog pravila čuva ekonomičnost. Pretpostavimo, dakle, da se u nekom koraku algoritma boji grana $e = uv$ i da je prethodno bojenje ekonomično, tako da je T skup grana jednog minimalnog razapinjućeg stabla od $\mathcal{G}$ koji sadrži sve u posmatranom trenutku plave grane i nijednu crvenu granu.

(1) *e se boji po plavom pravilu.* Ako je $e \in T$, tada upravo T "svedoči" da je novo bojenje takođe ekonomično. Zbog toga, razmotrimo slučaj $e \notin T$. Neka je $X \subset V$ rez na koji smo upravo primenili plavo pravilo. Pošto grane iz skupa T obrazuju razapinjuće stablo od $\mathcal{G}$, T sadrži grane koje formiraju put $u \rightsquigarrow v$. Kako po plavom pravilu jedan od čvorova u, v pripada X, a drugi ne, uočeni put sadrži granu $e' = xy$ tako da je jedan od čvorova x, y iz X, a drugi nije. Po plavom pravilu, grana e' ne može biti plava. Po načinu na koji smo izabrali T, e' ne može biti ni crvena, pa je u pitanju neobojena grana. Pri tome je $w(e) \leqslant w(e')$, imajući u vidu plavo pravilo. Neka je sada

$$T' = (T \setminus \{e'\}) \cup \{e\}.$$

Očigledno, $w(T') \leqslant w(T)$. Osim toga, grane skupa T' formiraju razapinjuće stablo od $\mathcal{G}$, jer je $|T'| = |T| = |V| - 1$ i T' ne sadrži cikluse (dodavanjem grane e se stvara tačno jedan ciklus, koji nestaje uklanjanjem grane e'). Stoga grane iz T' takođe formiraju minimalno razapinjuće stablo od $\mathcal{G}$ (i, specijalno, mora biti $w(T') = w(T)$ i $w(e') = w(e)$). Ovo stablo pokazuje da je bojenje koje u posmatranom slučaju nastaje posle primene plavog pravila ekonomično.

(2) *e se boji po crvenom pravilu.* Ako $e \notin T$, tada T pokazuje ekonomičnost novonastalog bojenja. Stoga pretpostavljamo da je $e \in T$. Primetimo da se brisanjem grane e iz stabla određenog sa T to stablo "raspada" na dve komponente (svaka grana u stablu je most). Posmatrajmo ciklus u $\mathcal{G}$ u odnosu na koji je upravo primenjeno crveno pravilo. Ovaj ciklus sadrži granu $e'' \neq e$ koja je incidentna sa čvorovima iz različitih komponenti šume nastale brisanjem e iz T.

Po crvenom pravilu, ova grana nije crvena, ali nije ni plava, jer bi u suprotnom bilo $e'' \in T$, što protivreči činjenici da grane iz skupa $T \setminus \{e\}$ formiraju dve komponente povezanosti. Stoga, e'' je neobojena grana i po crvenom pravilu mora biti $w(e) \geqslant w(e'')$. Ako sada definišemo

$$T'' = (T \setminus \{e\}) \cup \{e''\},$$

dobijamo skup od $|V| - 1$ grana koji ne sadrži cikluse (jer je e'' most u tom skupu), tj. razapinjuće stablo. Slično kao i u prethodnom slučaju, to razapinjuće stablo je takođe minimalno i pokazuje ekonomičnost novog bojenja.

Sada pokazujemo da se proces bojenja grana neće ”zaglaviti” sve dok sve grane nisu obojene. Pretpostavimo da u nekom trenutku imamo parcijalno bojenje u kome postoji neobojena grana $e = uv$. Gornji argumenti pokazuju da je to parcijalno bojenje ekonomično: zbog toga, plave grane formiraju acikličan graf, tj. šumu. Nazovimo komponente te šume *plavim stablima*. Ako se čvorovi u, v nalaze u istom plavom stablu, tada se može primeniti crveno pravilo na ciklus određen sa e i granama (jedinstvenog) plavog puta $u \rightsquigarrow v$. U suprotnom, za rez X uzmimo čvorove plavog stabla kojem pripada čvor u. Tada se može primeniti plavo pravilo u odnosu na X (s tim da nije sigurno da će grana koju ćemo obojiti u plavo biti baš e). Dakle, u svakom slučaju se proces bojenja može nastaviti dogod postoje neobojene grane.

Najzad, razmotrimo ”završnu” situaciju u kojoj su sve grane obojene. Po dokazanom, ovo bojenje takođe mora biti ekonomično: postoji minimalno razapinjuće stablo $\mathcal{T}$ koje sadrži sve plave grane i nijednu crvenu. Ovo je moguće samo ako plave grane formiraju povezan graf, tj. stablo, i ako se ono poklapa sa $\mathcal{T}$. Prema tome, plavo pravilo će biti primenjeno $|V| - 1$ puta, nakon čega imamo jedno minimalno razpinjuće stablo od $\mathcal{G}$. □

Primetimo da je Jarník-Primov algoritam veoma specijalan slučaj plavo-crvenog algoritma: on primenjuje isključivo plavo pravilo, dogod je to moguće (za rez uzimamo upravo skup do tada odabranih čvorova). Posle $|V| - 1$ koraka u kojima smo primenili plavo pravilo, preostaje da se uzastopno primenjuje crveno pravilo, ali to naravno nije neophodno.

Isto se može reći i za Kraskalov algoritam, s tim da on primenjuje jednu za nijansu složeniju strategiju. Naime, ako je e grana koja se trenutno obrađuje (redosled obrade grana definisan je njihovom po težini soritranom listom) i ako je ona incidentna sa čvorovima u i v takvim da već postoji plavi put $u \rightsquigarrow v$, tada ćemo primeniti crveno pravilo u odnosu na ciklus određen opisanim putem i granom e. U suprotnom, u i v pripadaju različitim komponentama V_1 i V_2

"plave šume" određene skupom F, pa primenjujemo plavo pravilo npr. u odnosu na $X = V_1$ (jer je e u datom trenutku najlakša neobojena grana u celom grafu, pa tako i među onim granama čiji je jedan "kraj" u X).

Prema tome, gornja teorema implicira korektnost Kraskalovog i Jarník-Primovog algoritma.

Analiza Jarník-Primovog algoritma

Označimo sa X skup odabranih čvorova. Na početku je $X = \{v_1\}$. U svakoj iteraciji, bira se grana najmanje težine koja povezuje po jedan čvor iz X i $V \setminus X$. Naravno, ispitivanje svih grana bi oduzelo mnogo vremena, pa ćemo algoritam implementirati putem pogodnih struktura podataka koje će omogućiti brzo pronalaženje tražene grane. Za to će nam biti potrebna dva niza indeksirana skupom čvorova V (koje ćemo, naravno, ažurirati tokom algoritma): **close**$[v]$ će sadržati jedan od "najbližih" čvorova iz skupa X u odnosu na v (za $v \in X$ definisaćemo **close**$[v] = \infty$), dok će element niza **wmin**$[v]$ biti jednak $w(v, x)$ ako je **close**$[v] = x \neq \infty$, dok je **wmin**$[v] = \infty$ za $v \in X$. Tako će naredna odabrana grana biti najlakša od $\{v, \textbf{close}[v]\}$, $v \in V \setminus X$.

Na početku je **close**$[v_1] = \infty$ i **wmin**$[v_1] = \infty$, dok je za sve druge čvorove **close**$[v] = v_1$ i **wmin**$[v] = w(v, v_1)$. Sada svaka iteracija Jarník-Primovog algoritma podrazumeva (na nivou implementacije) sledeće dve faze:

- *Izbor sledeće (plave) grane.* Najpre nalazimo minimum niza **wmin** i identifikujemo jedan čvor $v \in V$ u kome se taj minimum realizuje. Ovaj posao zahteva vreme $\mathcal{O}(n)$, gde je $n = |V|$. Odabrana grana će biti $\{v, \textbf{close}[v]\}$, čime v postaje element skupa X ($X := X \cup \{v\}$), pa stavljamo **close**$[v]$ =**wmin**$[v] = \infty$. Ove dodele se realizuju u konstantnom vremenu.

- *Ažuriranje nizova* **close** *i* **wmin** (koje nastaje kao posledica uključenja v u X). Naime, za sve $u \in V \setminus X$, u slučaju da je **close**$[u] \neq \infty$ i $w(v, u) <$ **wmin**$[u]$, potrebno je redefinisati:

  - **close**$[u] := v$,
  - **wmin**$[u] := w(v, u)$.

  Ovo ažuriranje se obavlja u vremenu $\mathcal{O}(n)$.

Kako je broj iteracija $n - 1$, opisana implementacija radi u vremenu $\mathcal{O}(n^2)$.

U prethodnom, koristili smo matričnu reprezentaciju težinskih grafova. Ako ih prikažemo preko lista i koristimo neke složenije strukture podataka, moguće je postići složenost $\mathcal{O}(m \log n)$ ($m = |E|$).

Analiza Kraskalovog algoritma

Pre svega, u koraku 1 Kraskalovog algoritma se sortiraju grane po težini. Ovo se realizuje u vremenu $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$ (pošto je $m = \mathcal{O}(n^2)$), koristeći neki od standardnih algoritama sortiranja.

Radi implementacije iteracije sadržane u koracima 2–3, uvodi se struktura podataka poznata pod imenom *union-find*. Ova struktura se sastoji iz particije $\{A_1, \ldots, A_k\}$ datog skupa S (dakle, važi $i \neq j \Rightarrow A_i \cap A_j = \varnothing$ i $A_1 \cup \ldots \cup A_k = S$), na kojoj je moguće izvoditi dve operacije:

- $\mathrm{UNION}(A_i, A_j)$, kojom se postojeća particija zamenjuje particijom

$$\Big(\{A_1, \ldots, A_k\} \setminus \{A_i, A_j\}\Big) \cup \{A_i \cup A_j\}$$

  (klase A_i i A_j se "spajaju" u jednu klasu).

- $\mathrm{FIND}(x)$, koja za $x \in S$ vraća "ime", labelu klase kojoj x pripada.

Upravo nam je ovakva struktura podataka potrebna u Kraskalovom algoritmu: u njemu radimo sa razapinjućom šumom određenom skupom F, i komponente te šume čine particiju skupa čvorova V. Dalje, uključivanje grane $e = uv$ u F ima za posledicu da se komponente kojima u i v pripadaju (nazovimo ih V_1 i V_2) spajaju u jednu, što je upravo primena operacije $\mathrm{UNION}(V_1, V_2)$. Grana e se odabira ukoliko ona ne zatvara ciklus sa ranije odabranim granama — ovaj uslov se lako proverava, jer e zatvara ciklus ako i samo ako u i v pripadaju istoj komponenti u trenutnoj particiji. Prema tome, u Kraskalovom algoritmu se obrađivana grana e uključuje u F ukoliko je $\mathrm{FIND}(u) \neq \mathrm{FIND}(v)$.

Jedan od najprostijih načina za realizaciju opisane strukture jeste da posmatramo niz indeksiran elementima skupa S (odnosno V u Kraskalovom algoritmu), čiji su elementi iz skupa $\{1, \ldots, |S|\}$ (labele komponenti). Tako se $\mathrm{FIND}(x)$ izračunava u konstantnom vremenu, jer je u pitanju očitavanje određenog člana niza. Međutim, primena $\mathrm{UNION}(A_i, A_j)$ zahteva linearno vreme u odnosu na $|S|$, pošto je potrebno proći kroz ceo niz i izjednačiti labele koje odgovaraju komponentama A_i i A_j (preuzima se jedna od labela starih komponenti). Budući da se u implementaciji Kraskalovog algoritma operacija UNION primenjuje isključivo pri uključivanju grane u F, ona će biti primenjena tačno $n - 1$ puta. Provera uslova iteracije troši konstantno vreme, kao i sve iteracije u kojima se razmatrana grana odbacuje. Broj iteracija, jasno, može biti najviše m. Prema tome, ukupna vremenska složenost je $\mathcal{O}(m \log n + n^2 + m) = \mathcal{O}(n^2)$.

Nešto sofisticiranijim pristupom u realizaciji strukture *union-find* moguće je postići asimptotski bolji rezultat. Ako se ova struktura implementira preko drveta u kojima se u korenu nalazi prazan pokazivač (labela komponente) i prirodan broj koji označava broj čvorova u drvetu — pri čemu su grane u drvetu orijentisane "nagore", ka korenu, a spajanje komponenti se sprovodi tako što koren manjeg drveta postaje naslednik korena većeg, dok se brojevi koji označavaju veličinu drveta sabiraju — tada se može postići složenost $\mathcal{O}(m \log n)$. Ukoliko se prilikom poziva operacije $\mathrm{FIND}(x)$ (koja podrazumeva praćenje orijentisanog puta u odgovarajućem drvetu od x do korena) primenjuje tehnika *kompresije puteva*, što znači da se samo drvo preuređuje tako da x nakon otkrivanja korena drveta kome pripada postaje naslednik tog korena, za iteraciju 2–3 se može postići i složenost $\mathcal{O}((m+n)\alpha(n))$, gde je α tzv. *inverzna Akermanova funkcija*. Ova funkcija nije ograničena, ali raste izuzetno sporo: npr. poznato je da je $\alpha(x) \leqslant 4$ za sve $x < 2^{65536}$. Zbog toga se u svim realnim primenama može smatrati da je $\alpha(n)$ konstanta (čak, vrlo mala, $\leqslant 4$), čime se dobija da ovaj deo algoritma radi praktično u linearnom ("pseudolinearnom") vremenu. Takođe, u specijalnim slučajevima (koji se u praksi najčešće i susreću), moguće je izvršiti sortiranje iz koraka 1 u vremenu boljem od $\mathcal{O}(m \log m)$: ukoliko su težine po apsolutnoj vrednosti mali celi brojevi, poznate su metode sortiranja koje rade u linearnom vremenu $\mathcal{O}(m)$, tako da implementacija celog Kraskalovog algoritma postaje pseudolinearna.

Zadaci.

1. "Ručno" primeniti Jarník-Primov i Kraskalov algoritam na težinske grafove date u Zadatku 4.6.1 , odnosno na Slici 4.7.1 (uz prethodno fiksiranje neke hijerarhije među granama iste težine). Uporediti tok ova dva algoritma na datim grafovima.

2. U grafovima datim u prethodnom zadatku proveriti zavisnost razapinjućih stabala koja daje Jarník-Primov algoritam od početnog čvora v_1.

5

Nedeterminizam

Nedeterminizam je jedan od najzanimljivijih fenomena u teoriji računskih mašina. U svim računskim modelima koje smo do sada susretali, naredno stanje mašine je bilo *jednoznačno* (deterministički) određeno postojećom konfiguracijom, tako da je izračunavanje bilo ”linearan” objekat: niz konfiguracija koje nastaju tokom rada mašine. Koncept nedeterminizma je upravo suprotan ovome: mašina u svakom trenutku ima na raspolaganju nekoliko mogućnosti među kojima se bira naredno stanje. Trenutna konfiguracija ne određuje naredni korak mašine (tj. algoritma) jednoznačno, već taj izbor može zavisiti od niza faktora koji su van okvira našeg razmatranja.

Strogo uzev, apstraktni koncept nedeterminističke računske mašine nije realan u smislu da ne modelira rad nijednog fizički postojećeg uređaja za računanje[18]. Ipak, nedeterminizam se u izvesnom smislu pojavljuje u računarskoj praksi, kada spoljašnji činioci utiču na tok izračunavanja: na primer, neki proces u distribuiranom sistemu može zavisiti od poruka koje dobija od drugih procesa, pri čemu nije moguće predvideti ni sadržaj tih poruka, a ni trenutak u kome će one biti upućene.

U teoriji izračunljivosti je od posebnog interesa izučavanje odnosa između odgovarajućih determinističkih i nedeterminističkih modela. Primera radi, poznat je rezultat o ekvivalenciji determinističkih i nedeterminističkih konačnih

[18] Validnost ove rečenice se, međutim, ozbiljno dovodi u pitanje pojavom tzv. *kvantnih računara*.

automata: i jedni i drugi prihvataju jezike predstavljene regularnim izrazima, tzv. *regularne jezike*. S druge strane, kod pushdown automata za kontekstno-slobodne jezike ova simetrija ne važi: postoji kontekstno-slobodni jezik koji nije jezik nijednog determinističkog pushdown automata (vidi [15, 26]). U ovoj i narednoj glavi naš prvenstveni cilj je da razmotrimo koncept nedeterminističkog algoritma definisan preko nedeterminističke TM i da proučimo odnos običnih TM i njihovih nedeterminističkih "rođaka". Taj koncept će nam omogućiti da procenjujemo složenosti čitavog niza interesantnih algoritamskih problema.

5.1 Nedeterminističke Tjuringove mašine

Definicija *nedeterminističke Tjuringove mašine* (NTM) razlikuje se od determinističke varijante po tome što je funkcija prelaza sada definisana kao

$$\delta : Q \times \Gamma \to \mathcal{P}((\Gamma \cup \{L, R, H\}) \times Q).$$

(Alternativno, možemo posmatrati δ kao *relaciju prelaza*,

$$\delta \subseteq Q \times \Gamma \times (\Gamma \cup \{L, R, H\}) \times Q.)$$

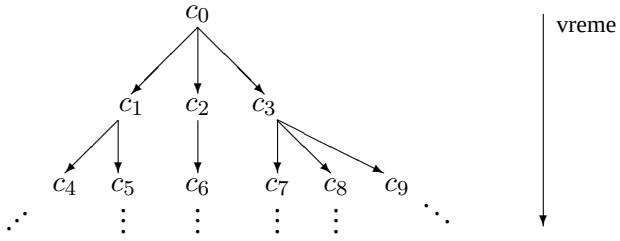
Jednu komandu NTM $\mathcal{M}$ možemo zapisati ovako:

$$q\ a \begin{cases} \alpha_1\ q_1 \\ \alpha_2\ q_2 \\ \vdots \\ \alpha_m\ q_m \end{cases}$$

(ukoliko je $\delta(q, a) = \{(\alpha_1, q_1), (\alpha_2, q_2), \ldots, (\alpha_m, q_m)\}$). Kao što smo već istakli, dok je kod TM stanjem kontrolnog mehanizma i simbolom koji se čita na traci jednoznačno određena sledeća akcija i sledeće stanje mašine, kod NTM to nije slučaj. Naprotiv, postoji neuređenost u smislu da naredni korak i stanje mašine može biti bilo koja od ponuđenih m opcija. Ovo, naravno, uključuje i mogućnost da za neko $q \in Q$ i $a \in \Gamma$ važi $m = 1$ (u kom slučaju imamo upravo determinističku "situaciju"), kao i $m = 0$ (kada se mašina zaglavljuje, jer nema narednog koraka, tj. konfiguracije u koju $\mathcal{M}$ može preći).

Naravno, prelaskom na računski model NTM, menja se i pojam toka izračunavanja. Ako je data konfiguracija c u kojoj je mašina u stanju q i čita simbol a, tada u slučaju $|\delta(q, a)| = m$ na raspolaganju stoji m različitih konfiguracija u koje ta mašina može da pređe. Ovakav pristup nam sugeriše da zamislimo rad NTM kao odvijanje "paralelnih stvarnosti", jednu vrstu paralelnog

računanja u kome se na istoj TM istovremeno izvršava više programa. Zbog toga, izračunavanje NTM prikazujemo kao *drvo*.



Slika 5.1.1. Drvo izračunavanja NTM

Svaki put $c_0 \rightsquigarrow c$ (tj. put od korena drveta do nekog čvora kome odgovara konfiguracija c) predstavlja rad jedne determinističke TM koja radi isto kao i data NTM, uz nametanje određenog niza izbora iz skupova $\delta(q, a)$.

Slično kao i kod TM, za NTM $\mathcal{M}$ kažemo da je *totalna* ako je drvo izračunavanja te mašine konačno (tj. konačne dubine) za svaku ulaznu reč w. U suprotnom, drvo izračunavanja sadrži beskonačan put: postoji način da nedeterministički biramo korake tako da $\mathcal{M}$ uđe u mrtvu petlju.

Jednostavnosti radi, mi ćemo posmatrati NTM isključivo u režimu za rešavanje problema odlučivanja (iako se može definisati i nedeterminističko izračunavanje parcijalne funkcije). Stoga treba da definišemo šta znači da NTM prihvata datu ulaznu reč. Kažemo da NTM $\mathcal{M}$ *prihvata* $w \in \Sigma^*$ ako u odgovarajućem drvetu izračunavanja postoji bar jedna konfiguracija oblika $(w_1, \top, w_2)$. Drugim rečima, ulazni podatak je prihvaćen ako postoji bar jedan način da se izvrši niz nedeterminističkih izbora tako da $\mathcal{M}$ dođe u prihvatno stanje. Obratno, ulaz je odbijen ako svako moguće izračunavanje rezultuje stanjem različitim od $\top$. Kao i ranije, $L(\mathcal{M})$, *jezik* NTM $\mathcal{M}$, sastoji se iz svih prihvaćenih reči nad ulaznom azbukom Σ.

Očigledno, svaka deterministička TM je specijalni slučaj NTM, pa zato klasa jezika svih NTM sadrži klasu rekurzivno nabrojivih jezika. Naredna teorema pokazuje da su u smislu računske moći NTM i obične TM ekvivalentne (tj. da jezik svake NTM pripada klasi **RE**), pošto je rad svake NTM moguće simulirati determinističkim modelom.

Teorema 5.1 *Za svaku nedeterminističku Tjuringovu mašinu $\mathcal{M}$ postoji deterministička 3-tračna Tjuringova mašina $\mathcal{D}$ tako da je $L(\mathcal{D}) = L(\mathcal{M})$.*

Skica dokaza. Pretpostavimo da je $b \in \mathbb{N}$ tako da u $\mathcal{M}$ važi $|\delta(q, a)| \leqslant b$ za sve $q \in Q$, $a \in \Gamma$. Trake mašine $\mathcal{D}$ zvaćemo redom A,B,C. Na početku, na traci A

je ulazna reč w, i ona je *off-line*, što znači da se ona nikada ne koristi za pisanje, dok su B i C prazne. Traka B je *radna* i na njoj će se vršiti izračunavanja, a C je *adresna traka* i na njoj će se tokom narednog algoritma generisati nizovi elemenata skupa $\{1, \ldots, b\}$ u istom poretku u kojem funkcija $|| \cdot ||$ enumeriše reči nad b-elementnim alfabetom (vidi Odeljak 1.4)[19].

Algoritam koji opisuje rad mašine $\mathcal{D}$ je sledeći:

1. Kopiraj ulaznu reč w sa trake A na traku B i postavi glavu trake B iza kopirane reči.

2. Na traci B simuliraj rad mašine $\mathcal{M}$ koristeći se trakom C kao sredstvom za vršenje nedeterminističkih izbora. Dakle, ako je u k-tom koraku simulacije mašina u stanju q i čita slovo a, tako da je

   $$\delta(q, a) = \{(\alpha_1, q_1), \ldots, (\alpha_c, q_c)\}$$

   za neko $c \leqslant b$, a na k-tom polju trake C stoji broj $m_k \leqslant c$, primeni komandu

   $$q\ a\ \alpha_{m_k}\ q_{m_k}.$$

   Kada na traci C nema više brojeva, ili je $m_k > c$, obustavi simulaciju $\mathcal{M}$.

3. Ako se (radom na traci B) došlo do prihvatnog stanja, PRIHVATI ulaznu reč i stani.

4. U suprotnom, generiši u naredni niz na traci C, obriši sadržaj trake B i vrati se na korak 1.

Očigledno, ovo je upravo deterministički algoritam koji vrši pretragu kompletnog stabla izračunavanja $\mathcal{M}$ za ulaznu reč w. On prihvata tu reč ako i samo ako je u smislu prethodnih definicija prihvata NTM $\mathcal{M}$. □

5.2 Nedeterminističke klase složenosti

Funkcija $T_{\mathcal{M}} : \mathbb{N} \to \mathbb{N}$ *ocenjuje vremensku složenost* totalne NTM $\mathcal{M}$ ako nijedno drvo izračunavanja za $\mathcal{M}$ koje odgovara ulaznoj reči dužine $\leqslant n$ nije dublje od $T_{\mathcal{M}}(n)$. *Ocena prostorne složenosti* $S_{\mathcal{M}} : \mathbb{N} \to \mathbb{N}$ mašine $\mathcal{M}$ se definiše analogno: u svakom nizu koraka koje $\mathcal{M}$ prevode iz početne konfiguracije u neku konfiguraciju iz drveta izračunavanja za ulaznu reč dužine $\leqslant n$, $\mathcal{M}$ izvršava komandu pisanja na ne više od $S_{\mathcal{M}}(n)$ polja trake.

[19] Taj poredak je definisan na sledeći način: za dva niza x, y nad $\{1, \ldots, b\}$ važi $x < y$ ako je x kraći od y, ili su nizovi x i y iste dužine i za najmanje i takvo da je $x_i \neq y_i$ važi $x_i < y_i$.

Na ove definicije se sada prirodno nadovezuje uvođenje klasa složenosti

$$NTIME(f(n)) = \{L : \text{ postoji NTM } \mathcal{M} \text{ tako da } L(\mathcal{M}) = L,\ T_{\mathcal{M}}(n) = \mathcal{O}(f(n))\},$$

i

$$NSPACE(f(n)) = \{L : \text{ postoji NTM } \mathcal{M} \text{ tako da } L(\mathcal{M}) = L,\ S_{\mathcal{M}}(n) = \mathcal{O}(f(n))\}.$$

Analogno kao i u deterministčkom slučaju, dalje formiramo vremensku klasu

$$\mathbf{NP} = \bigcup_{k \geqslant 0} NTIME(n^k),$$

kao i prostorne klase

$$\mathbf{NL} = NSPACE(\log n)$$

i

$$\mathbf{NPSPACE} = \bigcup_{k \geqslant 0} NSPACE(n^k).$$

U Odeljku 4.3 smo već spomenuli problem $\mathbf{P} = \mathbf{NP}$ (formulacija problema je, naravno, sadržana u njegovom imenu) i naglasili da je posredi jedno od najznačajnijih i najintrigantnijih otvorenih pitanja moderne matematike i teorijskog računarstva. Pri tome je inkluzija $\mathbf{P} \subseteq \mathbf{NP}$ trivijalna[20], pa se ovaj problem svodi na sledeće pitanje: da li se svaki nedeterministički algoritam (tj. NTM) može "determinizovati" tako da pri tom ostane sačuvana polinomnost utrošenog vremena (sa neograničenim mogućnostima "kvarenja" stepena polinoma)? Da li je, u načelu, za svaki problem iz $\mathbf{NP}$ moguća lokalizacija konfiguracije sa prihvatnim stanjem (ukoliko ona uopšte postoji) na srazmerno mali deo drveta izračunavanja? Da li se to drvo uvek može "pametno" pretražiti?

Problem leži u tome da je simulacija iz Teoreme 5.1 suviše "naivna", te da ne daje željeni rezultat. Naravno, pri tome mašina $\mathcal{D}$ nije totalna, pa je moramo prepraviti tako da se ona zaustavi kad god simulira NTM $\mathcal{M}$ konačne vremenske složenosti.

Teorema 5.2 *Neka je $\mathcal{M}$ NTM vremenske složenosti $\mathcal{O}(f(n))$. Tada postoji deterministička TM $\mathcal{D}'$ koja simulira $\mathcal{M}$ takva da je $T_{\mathcal{D}'}(n) = 2^{\mathcal{O}(f(n))}$.*

[20] Generalno, za sve funkcije $f : \mathbb{N} \to \mathbb{N}$ važe inkluzije $TIME(f(n)) \subseteq NTIME(f(n))$ i $SPACE(f(n)) \subseteq NSPACE(f(n))$.

Dokaz. Mašinu $\mathcal{D}'$ dobijamo malom modifikacijom mašine $\mathcal{D}$ iz Teoreme 5.1. Naime, dovoljno je da uvedemo jednu Bulovu promenljivu koja služi kao indikator da li se u pretraživanju drveta izračunavanja $\mathcal{M}$ za ulaz w došlo do poslednjeg sprata tog drveta (ovo možemo realizovati preko dva simbola trake — nazovimo ih 0 i 1 — koje upisujemo na novu, četvrtu traku mašine, ili ćemo dozvoliti pisanje na jednom polju trake A, desno od ulazne reči). Na početku, vrednost ove promenljive je 0. Kad god se u koraku 2 (vidi opis mašine $\mathcal{D}$) naiđe na konfiguraciju u kojoj je $\delta(q, a) \neq \varnothing$, postavlja se vrednost 1. Svaki put kada se na traci C (korak 4) dođe do adrese oblika $11\ldots1$ (leksikografski prve adrese date dužine), proverava se vrednost posmatrane promenljive. Ako je ona 0, mašina $\mathcal{D}'$ okončava svoj rad (jer nijedan čvor na prethodno obrađenom spratu analiziranog stabla izračunavanja nema naslednika). U suprotnom, promenljiva se ponovo postavlja na 0 i mašina nastavlja svoj rad (istraživanjem narednog sprata).

Na ovaj način, mašina $\mathcal{D}'$ će završiti svoj rad nakon što generiše sve adrese dužine $\leqslant T_{\mathcal{M}}(n)$. Svaka adresa dužine d određuje niz od d koraka jedne determinističke TM, pa zato deo simulacije vezan za tu adresu troši vreme $\mathcal{O}(d)$. Dakle, složenost mašine $\mathcal{D}'$ je asimptotski ograničena sa

$$b + 2b^2 + \ldots + tb^t \leqslant t^2 b^t \leqslant b^{2t} = 2^{\mathcal{O}(t)},$$

gde je $t = T_{\mathcal{M}}(n)$. U Teoremi 3.6 smo videli da prelazak na jednotračnu TM kvadrira složenost (na šta je red veličine $2^{\mathcal{O}(t)}$ invarijantan), pa sledi tvrđenje teoreme. □

Čak i kada bismo optimizovali gornju pretragu (npr. BFS tehnikom) tako da se u njoj svaki čvor drveta izračunavanja obilazi samo jednom (i da se za svaki čvor vezuje konstantan broj koraka mašine $\mathcal{D}$), broj samih čvorova u posmatranom drvetu možemo proceniti sa

$$1 + b + \ldots + b^t = \frac{b^{t+1} - 1}{b - 1} = \mathcal{O}(b^t) = 2^{\mathcal{O}(t)},$$

$t = T_{\mathcal{M}}(n)$, što ne menja gornji rezultat. Prema tome, za razliku od simulacija nekih drugih računskih modela (VTM, RAM mašine) običnim TM (gde se složenost simulacije izražava kao polinom u odnosu na složenost izvorne mašine), gornja simulacija NTM determinističkim modelom podrazumeva eksponencijalno usporenje. Čitav problem $\mathbf{P} = \mathbf{NP}$ je zapravo u tome što do sada nije poznata nijedna suštinski efikasnija simulacija od navedene, niti je pokazano da takva simulacija ne postoji.

S druge strane, upravo nam razmatrana simulacija omogućava da dobijemo sledeću vezu između nedeterminističkih vremenskih i determinističkih prostornih klasa.

Teorema 5.3 *Za svaku funkciju* $f : \mathbb{N} \to \mathbb{N}$ *takvu da je* $f(n) \geqslant n$ *važi* $NTIME(f(n)) \subseteq SPACE(f(n))$.

Skica dokaza. Neka je $\mathcal{M}$ nedeterministička mašina čija je vremenska složenost $t(n) = \mathcal{O}(f(n))$. Posmatrajmo mašinu $\mathcal{D}'$ iz dokaza Teoreme 5.2 koja je simulira (odnosno, mašinu $\mathcal{D}$ iz dokaza Teoreme 5.1, pošto je njena prostorna složenost asimptotski ista) i pratimo potrošnju prostora u pojedinim fazama njenog rada. Pri tome n označava dužinu ulazne reči w.

Korak 1 očigledno troši prostor n. Nakon toga se u koraku 2 simulira rad mašine $\mathcal{M}$ za jedan put u drvetu izračunavanja (adresna traka sadrži odgovarajući niz nedeterminističkih izbora). Pošto su svi takvi putevi dužine $\leqslant t(n)$, mašina $\mathcal{D}$ može tokom ove simulacije da piše na ne više od $t(n)$ polja. Do kraja jednog ciklusa u simulaciji na traci B neće biti potrošeno više nijedno polje (samo će ranije konzumirana polja biti "obrisana", tj. ispunjena sa $*$). Naravno, u skladu sa uslovom sadržanim u koraku 4, na adresnoj traci C može biti pisano na najviše $t(n) + 1$ polju. Prema tome, $S_{\mathcal{D}}(n) = \mathcal{O}(t(n)) = \mathcal{O}(f(n))$.

Preostaje da se primeti da prelazak na jednotračni model TM ne menja prostornu složenost (vidi Teoremu 3.6). □

Direktna posledica prethodne teoreme je inkluzija $\mathbf{NP} \subseteq \mathbf{PSPACE}$.

U Glavi 6 ćemo videti da klasa $\mathbf{NP}$ sadrži u izvesnom smislu reprezentativne, tzv. $\mathbf{NP}$*-kompletne probleme*. Njihov značaj je prvenstveno u tome što se na njima "lomi" pitanje $\mathbf{P} = \mathbf{NP}$: za ma koji $\mathbf{NP}$-kompletan problem odlučivanja $\mathcal{A}$ važi da je jednakost klasa $\mathbf{P}$ i $\mathbf{NP}$ *ekvivalentna* egzistenciji polinomnog (determinističkog) algoritma za rešavanje $\mathcal{A}$. Zbog toga ćemo u daljem razmotriti neke od najpoznatijih $\mathbf{NP}$-kompletnih problema.

Kada su u pitanju nedeterminističke prostorne klase, jedan od prvih i najznačajnijih rezultata jeste *Savičeva teorema* [38].

Teorema 5.4 (Walter J. Savitch, 1970) *Za svaku funkciju* $f : \mathbb{N} \to \mathbb{N}$ *takvu da je* $f(n) \geqslant \log n$ *važi* $NSPACE(f(n)) \subseteq SPACE([f(n)]^2)$.

Dokaz gornje teoreme zasniva se na konstrukciji algoritma za rešavanje problema dostižívosti u grafovima koji troši prostor $\mathcal{O}(\log^2 n)$. Kao specijalan slučaj Savičeve teoreme, važi $\mathbf{NL} \subseteq SPACE(\log^2 n)$. Pitanje da li je

$\mathbf{L} = \mathbf{NL}$ nešto je manje poznato od problema $\mathbf{P} = \mathbf{NP}$, ali (čini se) ništa manje teško. Kao posledicu prethodne teoreme imamo i jednakost $\mathbf{PSPACE} = \mathbf{NPSPACE}$. Najzad, može se pokazati da važi i inkluzija $\mathbf{NL} \subseteq \mathbf{P}$ (u dokazu se ponovo na pogodan način koristi problem dostiživosti).

Ako je $\mathbf{C}$ neka klasa složenosti (vremenska ili prostorna), definišemo $\mathbf{coC}$, *komplement* klase $\mathbf{C}$, kao $\mathbf{coC} = \{\overline{L} : L \in \mathbf{C}\}$, pri čemu se komplement jezika uzima u odnosu na domen problema koji taj jezik modelira. Na primer, $\overline{\mathrm{SAT}}$ predstavlja jezik koji se sastoji iz svih iskaznih formula koje *nisu* zadovoljive, tj. formula čija je negacija tautologija. Takođe, možemo posmatrati problem da li dati graf *nema* Hamiltonovu konturu, što predstavlja komplement problema Hamiltonovih kontura. Komplement problema da li je data reč nad Σ palindrom je pak predstavljen jezikom $\Sigma^* \setminus L_p$, gde je L_p skup svih palindroma nad Σ.

Veoma je lako videti da za svaku determinističku klasu $\mathbf{C}$ važi $\mathbf{C} = \mathbf{coC}$, jer ako totalna TM $\mathcal{M}$ prihvata jezik L sa složenošću $\mathcal{O}(f(n))$, tada je jezik mašine $\mathcal{M}'$, koja se dobija od $\mathcal{M}$ tako što se zamene uloge stanja $\top$ i $\bot$, upravo $\overline{L}$. Jasno, složenosti mašina $\mathcal{M}$ i $\mathcal{M}'$ su identične. Zbog toga, razmatranje komplementarnih klasa $\mathbf{coC}$ ima smisla samo u nedeterminističkom slučaju. Treći značajan otvoren problem teorije računske složenosti koji pominjemo u ovom odeljku jeste pitanje da li je $\mathbf{NP} = \mathbf{coNP}$ (očigledno, važi $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$). S druge strane, 1987/88. godine *R.Szelepcsényi* [42] i *N.Immerman* [16] su (nezavisno jedan od drugog) pokazali da za svaku funkciju $f : \mathbb{N} \to \mathbb{N}$ takvu da je $f(n) \geqslant \log n$ važi $\mathbf{NSPACE}(f(n)) = \mathbf{coNSPACE}(f(n))$, pa specijalno i $\mathbf{NL} = \mathbf{coNL}$.

5.3 Klasa NP i polinomna verifikacija

U ovom odeljku je naš cilj da damo jednu operativniju karakterizaciju problema koji se nalaze u klasi $\mathbf{NP}$.

Neka je $\mathcal{A} = (\Gamma, A)$ problem odlučivanja. Za problem $\mathcal{B}$ kažemo da *proverava* (ili *verifikuje*) problem $\mathcal{A}$ ako je $\mathcal{B}$ oblika

$$\mathcal{B} = (\Gamma \times \Delta, B),$$

gde je Δ najviše prebrojiv skup, tako da za sve $x \in \Gamma$ važi

$$x \in A \iff (\exists c \in \Delta)\,(x, c) \in B.$$

Za $x \in A$, objekat c za koji važi $(x, c) \in B$ nazivamo *sertifikat* (ili *dokaz*) za x.

Na primer, u problemu SAT, Δ može biti skup svih valuacija nad prebrojivim skupom iskaznih slova; sertifikat za zadovoljivu KNF ϕ jeste bilo koja

valuacija τ takva da je $v_\tau(\phi) = \top$. Tako je problem-verifikator za SAT sledeći: za datu formulu ϕ u KNF i datu valuaciju, da li je ϕ tačna u odnosu na tu valuaciju?

Slično, za problem Hamiltonovih kontura u grafu, sertifikat za Hamiltonov graf je permutacija njegovih čvorova koja obrazuje traženu konturu. Zato je verifikujući problem sledeći: za dati graf i permutaciju njegovih čvorova, da li su svaka dva uzastopna čvora u permutaciji (uključujući poslednji i prvi) susedni u grafu?

U oba primera, možemo primetiti da, dok za izvorni problem nije poznat efikasan algoritam, verifikujući problemi se veoma lako mogu odlučiti u polinomnom vremenu: izračunavanje istinitosne vrednosti KNF u odnosu na datu valuaciju zahteva linearno vreme (u odnosu na dužinu formule), dok se u drugom slučaju proveravajući problem svodi na inspekciju svih lista susednosti, dakle troši vreme $\mathcal{O}(m)$ (gde je m broj grana grafa). Da to nije slučajno, pokazuje sledeće tvrđenje, kojeg možemo uzeti i kao drugu definiciju klase $\mathbf{NP}$.

Teorema 5.5 *Neka je $\mathcal{A} = (\Gamma, A)$ problem odlučivanja. $\mathcal{A} \in \mathbf{NP}$ ako i samo ako $\mathcal{A}$ ima verifikator $\mathcal{B}$ tako da je $\mathcal{B} \in \mathbf{P}$.*

Skica dokaza. ($\Rightarrow$) Neka je $\mathcal{M}$ NTM polinomne vremenske složenosti koja rešava $\mathcal{A}$. Neka je u njoj uvek $|\delta(q, a)| \leqslant b$. U proveravajućem problemu koji konstruišemo, sertifikat će biti niz c koji se sastoji od brojeva iz skupa $\{1, \ldots, b\}$, dužine $\leqslant T_{\mathcal{M}}(n)$, gde je n dužina ulazne reči (Δ je skup svih nizova nad ovim skupom). Za konstrukciju verifikatora $\mathcal{B}$ ponovo koristimo mašinu $\mathcal{D}$ iz dokaza Teoreme 5.1: dovoljno je definisati B da bude skup svih parova (w, c) za koje mašina $\mathcal{D}$ vraća rezultat PRIHVATI kada se w postavi (kao ulazna reč) na traku A, a c na na traku C (pretpostavljamo da se ne vrše dalja generisanja nizova na C, već se simulacija mašine $\mathcal{M}$ sprovodi samo za dati, konkretni niz c). Time smo konstruisali TM koja rešava $\mathcal{B}$ i radi u vremenu $n + 2T_{\mathcal{M}}(n)$ (kopiranje w sa A na B, simulacija $\mathcal{M}$ na traci B i usputno čitanje sadržaja trake C radi selekcije nedeterminističkih izbora).

($\Leftarrow$) Pod datim pretpostavkama, konstruišemo NTM $\mathcal{M}$ koja rešava $\mathcal{A}$. Ona treba da radi na sledeći način:

1. Nedeterministički ispiši neki element $c \in \Delta$ desno od ulazne reči w, tako da je dužina reči koja reprezentuje c ne veća od $T_{\mathcal{M}'}(|w|)$ (duže reči ne mogu biti sertifikati za w, s obzirom na vremensku složenost mašine $\mathcal{M}'$).

2. Simuliraj rad TM $\mathcal{M}'$ koja rešava $\mathcal{B}$ i koja radi u polinomnom vremenu.

3. Uđi u prihvatno ili odbijajuće stanje u zavisnosti od rezultata simulacije $\mathcal{M}'$.

Iz gornjeg opisa mašine $\mathcal{M}$ imamo $T_{\mathcal{M}}(n) = \mathcal{O}(T_{\mathcal{M}'}(n))$ ($n = |w|$), pa traženi zaključak sledi. □

Tako, na osnovu gornje teoreme i prethodnih razmatranja zaključujemo da problem SAT i problem Hamiltonovih kontura pripadaju klasi $\mathbf{NP}$.

6

NP-kompletnost

6.1 Redukcije i kompletnost. NP-kompletnost

Neka su $\mathcal{A} = (\Gamma_1, A)$ i $\mathcal{B} = (\Gamma_2, B)$ dva problema odlučivanja. Kažemo da se $\mathcal{A}$ *(polinomno) redukuje* na $\mathcal{B}$, u oznaci $\mathcal{A} \leqslant_P \mathcal{B}$, ako postoji *redukciona funkcija* $\sigma : \Gamma_1 \to \Gamma_2$ koja ima sledeće dve osobine:

- σ je izračunljiva u determinističkom polinomnom vremenu,
- za sve $x \in \Gamma_1$ važi
$$x \in A \iff \sigma(x) \in B.$$

Problemi $\mathcal{A}$ i $\mathcal{B}$ su *ekvivalentni*, $\mathcal{A} \equiv_P \mathcal{B}$, ako istovremeno važi $\mathcal{A} \leqslant_P \mathcal{B}$ i $\mathcal{B} \leqslant_P \mathcal{A}$.

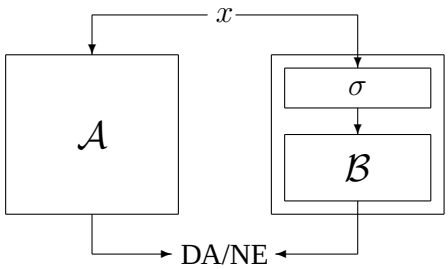
U oznakama $\leqslant_P$ i $\equiv_P$ koristimo indekse kako bismo razlikovali upravo definisanu redukciju (koja se u literaturi zove još i *Karpova* [18]) od drugih redukcija, kao što su Tjuringova $\leqslant_T$, logaritamska $\leqslant_L$ (koju primenjujemo u dokazu **NL**-kompletnosti problema dostiživosti), itd. Međutim, ovde ćemo koristiti isključivo navedenu polinomnu redukciju, pa ćemo u daljem izostavljati indekse.

Intuitivno, $\mathcal{A} \leqslant \mathcal{B}$ znači da problem $\mathcal{A}$ nije računski "teži" od $\mathcal{B}$. Specijalno, važi sledeće tvrđenje.

Lema 6.1 *Ako* $\mathcal{A} \leqslant \mathcal{B}$ *i* $\mathcal{B} \in \mathbf{P}$, *tada* $\mathcal{A} \in \mathbf{P}$.

Dokaz. Neka je $\mathcal{M}_\sigma$ TM koja izračunava funkciju σ u polinomnom vremenu, i neka je $\mathcal{M}$ mašina koja rešava problem $\mathcal{B}$. Kompozicija ove dve mašine rešava problem $\mathcal{A}$. Jasno, za datu instancu $x \in \Gamma_1$, $|x| = n$, mašina $\mathcal{M}_\sigma$ troši vreme koje je polinom po n. Takođe, $\mathcal{M}$ radi u vremenu koje je ograničeno polinomom po veličini njenog ulaza u ovom algoritmu, $|\sigma(x)|$. Preostaje da primetimo da ova poslednja veličina mora biti polinom po n, budući da je ona rezultat rada vremenski polinomno ograničene TM. □

Ako je $\mathcal{A} \leqslant \mathcal{B}$, tada odnos između ovih problema (a i kompoziciju mašina opisanu u gornjem dokazu) možemo slikovito prikazati sledećim dijagramom.



Neka je $\mathbf{C}$ neka klasa složenosti. Za problem $\mathcal{A}$ kažemo da je $\mathbf{C}$-*težak* ako za sve probleme $\mathcal{B} \in \mathbf{C}$ važi $\mathcal{B} \leqslant \mathcal{A}$. Ako je problem $\mathcal{A}$ $\mathbf{C}$-težak i još pri tom $\mathcal{A} \in \mathbf{C}$, onda je on $\mathbf{C}$-*kompletan*. Kompletni problemi jesu pravi "reprezentativni predstavnici" složenosti klase $\mathbf{C}$: u pitanju su računski najteži problemi te klase.

Naravno, ova definicija nam je najinteresantnija u slučaju da je $\mathbf{C} = \mathbf{NP}$. Značaj $\mathbf{NP}$-kompletnih problema je u potpunosti obrazložen sledećim tvrđenjem.

Propozicija 6.2 *Neka je $\mathcal{A}$ $\mathbf{NP}$-kompletan problem. Tada važi:*

$$\mathcal{A} \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}.$$

Dokaz. Implikacija ($\Leftarrow$) je trivijalna. S druge strane, neka je $\mathcal{B} \in \mathbf{NP}$ proizvoljan problem. Kako zbog kompletnosti važi $\mathcal{B} \leqslant \mathcal{A}$, to $\mathcal{A} \in \mathbf{P}$ po Lemi 6.1 povlači $\mathcal{B} \in \mathbf{P}$, pa je stoga $\mathbf{NP} \subseteq \mathbf{P}$ (obratna inkluzija je, kao što je već pomenuto, trivijalna). □

Prema tome, kako bi se potvrdila jednakost $\mathbf{P} = \mathbf{NP}$, dovoljno je za bilo koji od $\mathbf{NP}$-kompletnih problema pronaći polinomni algoritam. Obratno, ako bi bilo moguće za ma koji od tih problema pokazati nepostojanje odgovarajućeg polinomnog algoritma, odmah bi sledilo $\mathbf{P} \neq \mathbf{NP}$.

Ali, da li $\mathbf{NP}$-kompletni problemi uopšte postoje? Odgovor na ovo pitanje dobićemo već u narednom odeljku.

6.2 Kuk-Levinova teorema

Prvi otkriven $\mathbf{NP}$-kompletan problem bio je SAT, problem zadovoljivosti iskaznih formula. $\mathbf{NP}$-kompletnost ovog problema dokazali su (nezavisno jedan od drugog) *Stiven Kuk* (Stephen A. Cook, 1939) [6] i *Leonid A. Levin* [24].

Teorema 6.3 (S.A.Cook, L.A.Levin) SAT *je* $\mathbf{NP}$*-kompletan problem.*

Skica dokaza. Neka je L proizvoljan jezik iz klase $\mathbf{NP}$. To znači da postoji NTM

$$\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \surd, \times)$$

polinomne vremenske složenosti tako da je $L(\mathcal{M}) = L$. Naš cilj je da za svaku reč $w \in \Sigma^*$ konstruišemo iskaznu formulu $\phi(\mathcal{M}, w)$ koja ima sledeću izuzetnu osobinu:

$$w \in L(\mathcal{M}) \text{ ako i samo ako je } \phi(\mathcal{M}, w) \text{ zadovoljiva.}$$

Pri tome, mora postojati algoritam za konstrukciju ove formule koji radi u polinomnom vremenu u odnosu na $|w|$. Osnovna ideja koja nas vodi ovakvom kodiranju problema odlučivanja za jezik L iskaznom formulom jeste da (putem većeg broja iskaznih slova) $\phi(\mathcal{M}, w)$ sadrži kompletnu informaciju o svim relevantnim elementima mašine, te da jedna valuacija zapravo reprezentuje jednu granu u drvetu izračunavanja mašine $\mathcal{M}$ za ulaz w. Pri tome, ta valuacija će zadovoljiti $\phi(\mathcal{M}, w)$ ako i samo ako krajnji čvor te grane odgovara prihvatajućoj konfiguraciji.

Bez umanjenja opštosti, možemo pretpostaviti da smo odabrali dovoljno veliko $k \in \mathbb{N}$ (koje zavisi isključivo od mašine $\mathcal{M}$) tako da funkcija $T_{\mathcal{M}}(n) = n^k$ ocenjuje vremensku složenost od $\mathcal{M}$. Drugim rečima, $\mathcal{M}$ obustavlja svoj rad nakon $\leqslant n^k$ koraka. Ako pri tome numerišemo polja trake celim brojevima tako da je polje koje glava skenira u početnoj konfiguraciji označeno sa 0, to takođe znači da će mašina tokom obrade ulaza w (nezavisno od nedeterminističkih izbora) biti ograničena na deo trake označen brojevima iz intervala $[-n^k, n^k]$.

Iskazna slova koja koristimo u $\phi(\mathcal{M}, w)$ delimo u tri grupe:

- x_i^q ($0 \leqslant i \leqslant n^k$, $q \in Q$) — postavljanje vrednosti ove promenljive na $\top$ će intuitivno značiti:

  "Nakon i-tog koraka, stanje mašine $\mathcal{M}$ je q."

- y_{ij} ($0 \leqslant i \leqslant n^k$, $-n^k \leqslant j \leqslant n^k$) — postavljanje vrednosti ove promenljive na $\top$ će intuitivno značiti:

"Nakon i-tog koraka, glava mašine $\mathcal{M}$ skenira polje trake br. j."

- z_{ij}^a ($0 \leqslant i \leqslant n^k$, $-n^k \leqslant j \leqslant n^k$, $a \in \Gamma$) — činjenica da je u nekoj valuaciji vrednost ove promenljive $\top$ kodira iskaz:

  "Nakon i-tog koraka, u polju trake br. j nalazi se simbol a."

Formula $\phi(\mathcal{M}, w)$ biće sastavljena od nekoliko delova:

$$\phi(\mathcal{M}, w) = \phi_{\text{state}} \wedge \phi_{\text{head}} \wedge \phi_{\text{tape}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}.$$

Zadatak prva tri dela jeste da obezbede jedinstvenost konfiguracije mašine $\mathcal{M}$ u svakom pojedinačnom trenutku. Na primer, formula ϕ_{state} treba da bude konstruisana tako da kodira iskaz "Nakon svakog koraka, mašina $\mathcal{M}$ je tačno u jednom stanju.", tj. da jedine valuacije τ koje zadovoljavaju ovu formulu budu tačno one u kojima za svako i važi $\tau(x_i^q) = \top$ za tačno jedno $q \in Q$. Lako se proverava da sledeća KNF ima traženu osobinu:

$$\left[\bigwedge_{0 \leqslant i \leqslant n^k} \left(\bigvee_{q \in Q} x_i^q\right)\right] \wedge \left[\bigwedge_{0 \leqslant i \leqslant n^k} \bigwedge_{\substack{p,\, q \,\in\, Q \\ p \neq q}} (\neg x_i^p \vee \neg x_i^q)\right].$$

Analogno, ϕ_{head} obezbeđuje jedinstvenost položaja glave:

$$\left[\bigwedge_{0 \leqslant i \leqslant n^k} \left(\bigvee_{-n^k \leqslant j \leqslant n^k} y_{ij}\right)\right] \wedge \left[\bigwedge_{0 \leqslant i \leqslant n^k} \bigwedge_{-n^k \leqslant j < \ell \leqslant n^k} (\neg y_{ij} \vee \neg y_{i\ell})\right],$$

dok je uloga ϕ_{tape} da obezbedi jedinstvenost sadržaja svakog (u toku obrade w na $\mathcal{M}$ dostupnog) polja trake u svakom trenutku:

$$\left[\bigwedge_{0 \leqslant i \leqslant n^k} \bigwedge_{-n^k \leqslant j \leqslant n^k} \left(\bigvee_{a \in \Gamma} z_{ij}^a\right)\right] \wedge \left[\bigwedge_{0 \leqslant i \leqslant n^k} \bigwedge_{-n^k \leqslant j \leqslant n^k} \bigwedge_{\substack{a,\, b \,\in\, \Gamma \\ a \neq b}} (\neg z_{ij}^a \vee \neg z_{ij}^b)\right].$$

Podformule ϕ_{start} i ϕ_{accept} treba da redom opišu početnu i prihvatajuće konfiguracije. Naravno, glava je na početku u stanju q_0, čita (po dogovoru) nulto polje trake koje sadrži blanko (to je prvo polje desno od ulazne reči), i ako je

$w = a_1 \dots a_n$ tada se simbol a_j ($1 \leqslant j \leqslant n$) nalazi na polju br. $j - (n+1)$, dok su ostala polja prazna. Ovo stanje opisuje upravo sledeća formula:

$$x_0^{q_0} \wedge y_{00} \wedge \bigwedge_{j=1}^{n} z_{0,j-(n+1)}^{a_j} \wedge \bigwedge_{-n^k \leqslant j \leqslant -n-1} z_{0j}^{*} \wedge \bigwedge_{0 \leqslant j \leqslant n^k} z_{0j}^{*}.$$

Reč w je prihvaćena od strane $\mathcal{M}$ ako se u drvetu izračunavanja pojavi kofiguracija sa prihvatajućim stanjem $\surd$ (zamenili smo oznake prihvatajućeg i odbijajućeg stanja, radi izbegavanja zabune sa istinitosnim vrednostima), pa zato ϕ_{accept} glasi:

$$\bigvee_{0 \leqslant i \leqslant n^k} x_i^{\surd}.$$

Najzad, podformula ϕ_{move} treba da sadrži informaciju o svim mogućim prelazima mašine $\mathcal{M}$, tj. da bude zadovoljiva samo onim valuacijama koje opisuju izračunavanja koja su u saglasnosti sa programom δ. U zapisu ove podformule koristićemo se i iskaznim veznikom $\Rightarrow$, imajući u vidu da je $\alpha \Rightarrow \beta$ skraćeni zapis za $\neg\alpha \vee \beta$ i koristeći (po potrebi) De Morganove zakone. ϕ_{move} se prirodno sastoji četiri dela. Najpre, ako nakon i-tog koraka glava *ne* skenira j-to polje, njegov sadržaj će ostati nepromenjen i nakon $(i+1)$-vog koraka. Za ovu svrhu uvodimo formulu ψ_1:

$$\bigwedge_{\substack{0 \leqslant i \leqslant n^k \\ -n^k \leqslant j \leqslant n^k \\ a \in \Gamma}} \left((\neg y_{ij} \wedge z_{ij}^{a}) \Rightarrow z_{i+1,j}^{a} \right).$$

S druge strane, ψ_2 opisuje efekat primene neke komande oblika $q\ a\ b\ q'$ iz δ na trenutno skenirano polje:

$$\bigwedge_{\substack{0 \leqslant i \leqslant n^k \\ -n^k \leqslant j \leqslant n^k \\ a \in \Gamma}} \left((x_i^q \wedge y_{ij} \wedge z_{ij}^a) \Rightarrow \bigvee_{(q,a,b,q') \in \delta} (x_{i+1}^{q'} \wedge y_{i+1,j} \wedge z_{i+1,j}^b) \right).$$

Analogno, ψ_3 opisuje pokrete glave nalevo:

$$\bigwedge_{\substack{0 \leqslant i \leqslant n^k \\ -n^k \leqslant j \leqslant n^k \\ a \in \Gamma}} \left((x_i^q \wedge y_{ij} \wedge z_{ij}^a) \Rightarrow \bigvee_{(q,a,L,q') \in \delta} (x_{i+1}^{q'} \wedge y_{i+1,j-1} \wedge z_{i+1,j}^a) \right),$$

a ψ_4 nadesno:

$$\bigwedge_{\substack{0 \leqslant i \leqslant n^k \\ -n^k \leqslant j \leqslant n^k \\ a \in \Gamma}} \left((x_i^q \wedge y_{ij} \wedge z_{ij}^a) \Rightarrow \bigvee_{(q,a,R,q') \in \delta} (x_{i+1}^{q'} \wedge y_{i+1,j+1} \wedge z_{i+1,j}^a) \right).$$

Definišemo $\phi_{\text{move}} = \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4$.

Pretpostavimo sada da je $w \in L(\mathcal{M})$. Tada u odgovarajućem drvetu izračunavanja postoji čvor sa prihvatajućom konfiguracijom. Posmatrajmo put u drvetu koja sadrži taj čvor. Konstruišemo valuaciju τ tako da ako je u i-tom čvoru tog puta ($0 \leqslant i \leqslant n^k$) mašina $\mathcal{M}$ u stanju q i čita j-to polje trake, tada je $\tau(x_i^q) = \tau(y_{ij}) = \top$ dok sva druga x- i y-slova imaju vrednost $\bot$. Vrednosti slova z_{ij}^a biramo tako da kodiraju sadržaj trake mašine $\mathcal{M}$ u i-tom čvoru uočenog puta. Rutinski se proverava da je sada $v_\tau(\phi(\mathcal{M}, w)) = \top$.

Obratno, neka valuacija τ zadovoljava $\phi(\mathcal{M}, w)$. Budući da τ zadovoljava $\phi_{\text{state}} \wedge \phi_{\text{head}} \wedge \phi_{\text{tape}}$, za svako i, $0 \leqslant i \leqslant n^k$, postoje jedinstveni $q \in Q$ $j \in [-n^k, n^k]$ i $a \in \Gamma$ tako da je $\tau(x_i^q) = \tau(y_{ij}) = \tau(z_{ij}^a) = \top$. Na ovaj način prirodno dobijamo niz konfiguracija $c_0, c_1, \ldots, c_{n^k}$. Pošto τ zadovoljava ϕ_{start}, c_0 je baš početna konfiguracija mašine $\mathcal{M}$ za ulaz w. Pošto τ zadovoljava i ϕ_{accept}, ovaj niz sadrži prihvatajuću konfiguraciju. Najzad, kako τ zadovoljava ϕ_{move}, za $\mathcal{M}$ važi $c_0 \vdash c_1 \vdash \ldots \vdash c_{n^k}$. Prema tome, $\mathcal{M}$ prihvata w.

Preostaje da se dužina formule $\phi(\mathcal{M}, w)$ oceni polinomnom funkcijom po n. Kako su veličine poput $|Q|$, $|\Gamma|$ i $|\delta|$ konstantne (ne zavise od w), dužina formula ϕ_{state}, ϕ_{head}, ϕ_{tape} i ϕ_{move} je $\mathcal{O}(n^{2k})$, dok je dužina formula ϕ_{start} i ϕ_{accept} reda $\mathcal{O}(n^k)$. Takođe, nije teško pokazati da postoji algoritam koji na osnovu reči w i parametara mašine $\mathcal{M}$ konstruiše $\phi(\mathcal{M}, w)$, pri čemu se vreme potrebno za ispisivanje svakog simbola može ograničiti polinomom po n. Stoga je prezentirana redukcija polinomna, pa je teorema dokazana. □

6.3 Problem klika u grafovima

Polazeći od Kuk-Levinove teoreme i koncepta polinomne redukcije, moguće je pronaći čitavo "more" drugih **NP**-kompletnih problema. Naime, važi sledeće.

Propozicija 6.4 *Neka je $\mathcal{A}$ **NP**-kompletan problem. Ako $\mathcal{A} \leqslant \mathcal{B}$ i $\mathcal{B} \in \mathbf{NP}$, tada je i problem $\mathcal{B}$ takođe **NP**-kompletan.*

Dokaz. Kako je $\mathcal{A}$ **NP**-težak, za svaki problem $\mathcal{A}' \in \mathbf{NP}$ važi $\mathcal{A}' \leqslant \mathcal{A}$. Neka je σ' odgovarajuća redukciona funkcija, dok σ realizuje redukciju $\mathcal{A} \leqslant \mathcal{B}$. Tada

kompozicija funkcija $\sigma' \circ \sigma$ redukuje $\mathcal{A}'$ na $\mathcal{B}$. Kako se σ' izračunava u vremenu $\mathcal{O}(n^k)$ za neko $k > 0$, za svaki element x iz domena problema $\mathcal{A}'$ važi $|\sigma'(x)| = \mathcal{O}(|x|^k)$. Ako složenost izračunavanja funkcije σ iznosi $\mathcal{O}(n^\ell)$, tada se $\sigma(\sigma'(x))$ izračunava u vremenu $\mathcal{O}(|x|^k + (|x|^k)^\ell) = \mathcal{O}(|x|^{k\ell})$. Dakle, redukcija $\sigma' \circ \sigma$ je polinomna, pa važi $\mathcal{A}' \leqslant \mathcal{B}$. Stoga je problem $\mathcal{B}$ **NP**-težak, a time i **NP**-kompletan. □

Kao prvu ilustraciju ovog tvrđenja, redukovaćemo SAT na *problem klika u grafovima*. Ovaj problem je definisan na sledeći način:

ULAZ: Neorijentisan graf $\mathcal{G} = (V, E)$ i prirodan broj k.

IZLAZ: Da li $\mathcal{G}$ ima kompletan podgraf (kliku) od k čvorova?

Ovaj problem (koji označavamo sa KLIKE) pripada klasi **NP**, jer podskup $C \subseteq V$, $|C| = k$, predstavlja odgovarajući sertifikat, a verifikacija da li je C zaista klika u $\mathcal{G}$ zahteva $k(k-1)/2$ provera susednosti parova čvorova, što iznosi $\mathcal{O}(n^2)$ (pošto mora biti $k \leqslant n = |V|$, u suprotnom je odgovor očigledno negativan)[21].

Prema prethodnoj propoziciji, kako bismo pokazali **NP**-kompletnost problema klika dovoljno je konstruisati polinomnu redukciju $\mathrm{SAT} \leqslant \mathrm{KLIKE}$. Ova redukcija zahteva da se za proizvoljnu KNF ϕ konstruiše graf $\mathcal{G}_\phi = (V_\phi, E_\phi)$ i prirodan broj k_ϕ tako da važi sledeća ekvivalencija:

ϕ je zadovoljiva ako i samo ako $\mathcal{G}_\phi$ ima kliku od k_ϕ čvorova.

Najpre, za k_ϕ uzimamo broj konjunkta u KNF ϕ. Skup čvorova V_ϕ će biti skup svih *pojavljivanja* literala u ϕ (dakle, višestrukim pojavljivanjima istog literala odgovaraju različiti čvorovi). Dva čvora u, v ovog grafa spajamo granom ukoliko su ispunjena sledeća dva uslova:

- u i v ne predstavljaju literale iz istog konjunkta,
- literali koji odgovaraju čvorovima u, v nisu komplementarni (tj. nije reč o $x, \neg x$ za neko slovo x).

Zbog prvog uslova, ovaj graf će biti k_ϕ-partitan: čvorovi koji reprezentuju literale iz istog konjunkta čine jednu klasu particije. Stoga se svaka njegova eventualna klika mora sastojati iz čvorova iz različitih klasa (tj. literala iz različitih

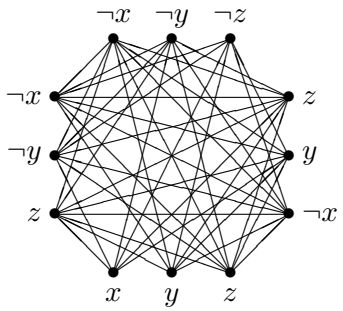
[21] Pri tome, neposredna provera svih k-elementnih podskupova od V *nije* polinomni algoritam za KLIKE, jer je broj takvih podskupova $\binom{n}{k}$. Budući da je ovde k deo ulaznog podatka, a *ne* fiksan broj, ovo je u opštem slučaju eksponencijalna veličina po n, pošto je najveći binomni koeficijent n-te klase $\geqslant 2^n/(n+1)$.

konjunkta); tako svaka njegova k_ϕ-klika (ako takva uopšte postoji) sadrži tačno po jednog predstavnika svake klase.

Primer 6.5 Posmatrajmo KNF

$$\phi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z).$$

Za ovu formulu je $k_\phi = 4$, dok odgovarajući graf $\mathcal{G}_\phi$ izgleda ovako:



□

(⇒) *ϕ je zadovoljiva.* Neka je $\tau : X_\phi \to \{\top, \bot\}$ valuacija koja zadovoljava ϕ. Tada svaki konjunkt sadrži bar jedan literal koji je tačan u odnosu na ovu valuaciju. Odaberimo po jedan takav literal iz svakog konjunkta; neka oni redom odgovaraju čvorovima $u_1, \ldots, u_{k_\phi}$ konstruisanog grafa.

Tvrdimo da ovi čvorovi formiraju kliku. Zaista, njima odgovarajući literali svi dolaze iz različitih konjunkta, po konstrukciji. Štaviše, nikoja dva od njih ne mogu odgovarati komplementarnim literalima, jer bi u suprotnom bilo nemoguće da je njihova vrednost istovremeno $\top$ u odnosu na τ. Sledi da svaka dva od navedenih čvorova moraju biti susedni u $\mathcal{G}_\phi$.

(⇐) *$\mathcal{G}_\phi$ ima k_ϕ-kliku.* Neka se ta klika sastoji od čvorova $v_1, \ldots, v_{k_\phi}$. Kao što je ranije napomenuto, ovi čvorovi moraju odgovarati literalima iz različitih konjunkta. Neka za $v \in V$, $\ell(v)$ označava literal koji odgovara čvoru v. Sada definišemo valuaciju τ na sledeći način ($x \in X_\phi$):

$$\tau(x) = \begin{cases} \top & x \in \{\ell(v_1), \ldots, \ell(v_{k_\phi})\}, \\ \bot & \text{inače.} \end{cases}$$

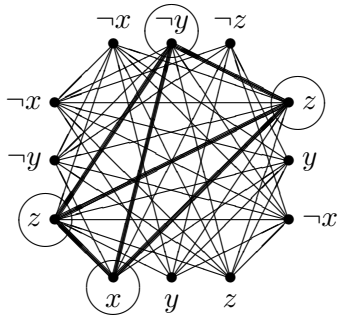
Tvrdimo da je $v_\tau(\phi) = \top$, tj. da ϕ u svakom svom konjunktu ima bar jedan literal koji je tačan u odnosu na τ. Neka je $\ell(v_j) = x_{i_j}^{\alpha_{i_j}}$, $1 \leqslant j \leqslant k_\phi$. Ako je $\alpha_{i_j} = \top$, tada $x_{i_j} \in \{\ell(v_1), \ldots, \ell(v_{k_\phi})\}$ (pošto je $x_{i_j} = \ell(v_j)$), pa je $v_\tau(\ell(v_j)) = \tau(x_{i_j}) = \top$. U suprotnom slučaju ($\alpha_{i_j} = \bot$), imamo $\ell(v_j) =$

$\neg x_{i_j}$, pa zato $x_{i_j} \notin \{\ell(v_1), \ldots, \ell(v_{k_\phi})\}$, jer bi inače skup $\{\ell(v_1), \ldots, \ell(v_{k_\phi})\}$ sadržao par komplementarnih literala, što je nemoguće po konstrukciji. Otuda je $\tau(x_{i_j}) = \bot$, odnosno $v_\tau(\ell(v_j)) = \top$. Dakle, u svakom slučaju, svi literali $\ell(v_j)$ imaju vrednost $\top$ u odnosu na τ, pa ova valuacija zadovoljava ϕ.

Preostaje da se primeti da je sama redukciona funkcija (tj. graf $\mathcal{G}_\phi$ i prirodan broj k_ϕ) izračunljiva u polinomnom vremenu u odnosu na n, dužinu formule ϕ. Zaista, $|V_\phi| = \mathcal{O}(n)$, dok je za svaki par čvorova (kojih ima $\mathcal{O}(n^2)$) za utvrđivanje njihove susednosti potrebno najviše linearno vreme (dovoljan je jedan "prolaz" kroz formulu ϕ da bi se utvrdilo da li njima odgovarajući literali leže u istom konjunktu, a za utvrđivanje komplementarnosti literala se troši konstantno vreme). Stoga se ceo graf $\mathcal{G}_\phi$ može konstruisati u vremenu $\mathcal{O}(n^3)$. Time je dokazana

Teorema 6.6 *Problem* KLIKE *je* **NP***-kompletan.*

Primer 6.7 Graf dat u Primeru 6.5 ima nekoliko 4-klika. Jedna od njih je istaknuta na sledećoj slici:



Ovoj kliki odgovara valuacija

$$\begin{pmatrix} x & y & z \\ \top & \bot & \top \end{pmatrix},$$

koja očigledno zadovoljava formulu ϕ iz prethodnog primera. □

Sa problemom KLIKE su blisko povezana i sledeća dva problema:

- INDEP. Za skup čvorova $X \subseteq V$ grafa $\mathcal{G} = (V, E)$ kažemo da je *nezavisan* ako u $\mathcal{G}$ ne postoji grana koja je incidentna sa dva čvora iz X (drugim rečima, X formira anti-kliku). Problem INDEP pita da li u datom grafu $\mathcal{G}$ postoji nezavisan skup od k čvorova, gde je $k \in \mathbb{N}$ dati broj. Ovaj problem je **NP**-kompletan zbog redukcije KLIKE $\leqslant$ INDEP koja datom grafu $\mathcal{G}$ i broju k pridružuje par $(\overline{\mathcal{G}}, k)$, gde je $\overline{\mathcal{G}} = (V, V^2 \setminus E)$ *komplement* grafa $\mathcal{G}$.

- V-COVER. Za skup čvorova $X \subseteq V$ grafa $\mathcal{G} = (V, E)$ kažemo da je *čvorni pokrivač* ako je svaka grana u $\mathcal{G}$ incidentna sa bar jednim čvorom iz X. Problem V-COVER pita da li u datom grafu $\mathcal{G}$ i za dato $k \in \mathbb{N}$ postoji čvorni pokrivač od k čvorova. Pošto je X čvorni pokrivač sa k čvorova ako i samo ako je $V \setminus X$ nezavisan skup od $|V| - k$ čvorova, važi INDEP $\leqslant$ V-COVER, pa je V-COVER **NP**-kompletan problem.

Zadaci.

1. Konstruisati graf $\mathcal{G}_\varphi$ za formulu

$$\varphi = (x \vee y) \wedge (\neg x \vee \neg y) \wedge (x \vee \neg y)$$

i na osnovu njega odrediti sve valuacije koje zadovoljavaju φ.

6.4 Problem k-SAT

Neka je ϕ iskazna formula u KNF. Za prirodan broj $k > 0$ kažmo da je ϕ u *k-konjunktivnoj normalnoj formi (k-KNF*) ako svaki konjunkt u ϕ sadrži ne više od k literala.

Problem k-SAT je (slično problemu HORNSAT) restrikcija problema SAT za k-KNF. Znači, algoritamsko pitanje ostaje isto (zadovoljivost formule ϕ), jedino je skup instanci sužen.

Odmah se nameće pitanje računske složenosti problema k-SAT za različite vrednosti k. Ispostavlja se da je vrednost parametra k od suštinskog značaja; naime, pokazaćemo da ovaj problem

- za $k \leqslant 2$ pripada **P**, dok je
- za $k \geqslant 3$ on **NP**-kompletan.

Najpre ćemo obrazložiti potonje tvrđenje tako što ćemo dokazati da važi

Teorema 6.8 *Problem* 3-SAT *je* **NP**-*kompletan.*

Naime, moguća je redukcija SAT $\leqslant$ 3-SAT, tj. u ovom slučaju se *opšti problem redukuje na svoj specijalan slučaj* (tada zbog trivijalne redukcije 3-SAT $\leqslant k$-SAT, $k \geqslant 4$, imamo **NP**-kompletnost problema k-SAT i za sve veće vrednosti k). Nakon dokaza gornje teoreme, prikazaćemo polinomni algoritam za 2-SAT. Problem 1-SAT ima očigledno rešenje, budući da je 1-KNF prosto

konjunkcija literala, a ona je zadovoljiva ako i samo ako ne sadrži komplementarne literale (što se neposredno proverava u linearnom vremenu).

Redukcija $\mathrm{SAT} \leqslant 3\text{-SAT}$ zasniva se na sledećem pomoćnom tvrđenju, poznatom kao *pravilo rezolucije*.

Lema 6.9 *Za sve iskazne formule ψ, θ, ξ u kojima se ne javlja iskazno slovo x važi da je formula*

$$(x \vee \psi) \wedge (\neg x \vee \theta) \wedge \xi$$

zadovoljiva ako i samo ako je zadovoljiva formula

$$(\psi \vee \theta) \wedge \xi.$$

Dokaz. Neka je $(x \vee \psi) \wedge (\neg x \vee \theta) \wedge \xi$ zadovoljiva formula: pretpostavimo da je zadovoljava valuacija τ. Tada je $\tau(x) = \top$ ili $\tau(x) = \bot$ (kada je $v_\tau(\neg x) = \top$), pa mora biti $v_\tau(\psi) = \top$ ili $v_\tau(\theta) = \top$. Osim toga, $v_\tau(\xi) = \top$, pa sledi $v_\tau((\psi \vee \theta) \wedge \xi) = \top$.

Obratno, ako je $v_{\tau'}((\psi \vee \theta) \wedge \xi) = \top$ za neku valuaciju τ' iskaznih slova koje se pojavljuju u formulama ψ, θ, ξ, tada je $v_{\tau'}(\xi) = \top$, i bar jedna od formula ψ, θ mora imati vrednost $\top$ u odnosu na τ'. Proširićemo τ' do valuacije τ čiji domen uključuje i x. Ako je formula ψ tačna u odnosu na τ', definišimo $\tau(x) = \bot$, a inače $\tau(x) = \top$. Sada je očigledno da je $v_\tau((x \vee \psi) \wedge (\neg x \vee \theta) \wedge \xi) = \top$, što se i tražilo. □

Dokaz Teoreme 6.8. Neka je ϕ proizvoljna KNF. Za svaki njen konjunkt

$$(\ell_1 \vee \ell_2 \vee \ldots \vee \ell_m),$$

gde su ℓ_i ($1 \leqslant i \leqslant m$) literali i $m \geqslant 4$, uvedimo nova iskazna slova $x_1, \ldots,$ x_{m-3} i zamenimo gornji konjunkt podformulom

$$(\ell_1 \vee \ell_2 \vee x_1) \wedge (\neg x_1 \vee \ell_3 \vee x_2) \wedge (\neg x_2 \vee \ell_4 \vee x_3) \wedge \ldots$$

$$\ldots \wedge (\neg x_{m-4} \vee \ell_{m-2} \vee x_{m-3}) \wedge (\neg x_{m-3} \vee \ell_{m-1} \vee \ell_m).$$

Na taj način se dobija 3-KNF ϕ'. Višestrukom primenom gornje leme sledi da je ϕ zadovoljiva ako i samo ako je ϕ' zadovoljiva. Osim toga, primetimo da se na opisani način od konjunkta koji se sastoji od $2m + r + 1$ simbola (gde je r broj negativnih literala) konstruiše podformula od ϕ' dužine $7 + 8(m - 3) + r + m - 3 = 9m + r - 20$. Prema tome, ϕ' se može izračunati na osnovu ϕ u linearnom vremenu, stoga je opisana redukcija zaista polinomna. To kompletira dokaz Teoreme 6.8, pošto 3-SAT očito pripada $\mathbf{NP}$. □

Naš drugi zadatak je da nađemo polinomni algoritam koji rešava problem 2-SAT, dakle, problem zadovoljivosti za KNF sa najviše dva literala po konjunktu. Bez ograničenja opštosti, možemo pretpostaviti da svaki konjunkt sadrži tačno dva literala, pošto svaki konjunkt oblika (ℓ) možemo pisati i kao $(\ell \vee \ell)$.

Neka je $\{x_1, \ldots, x_m\}$ skup svih iskaznih slova koja se pojavljuju u 2-KNF ϕ. Konstruisaćemo digraf $\mathcal{H}_\phi$ nad skupom čvorova

$$V = \{x_1, \neg x_1, \ldots, x_m, \neg x_m\}$$

tako što svakom konjunktu $(\ell \vee \ell')$ formule ϕ odgovara par orijetisanih grana $(\neg\ell, \ell')$ i $(\neg\ell', \ell)$. Primetimo da je digraf $\mathcal{H}_\phi$ u određenom smislu "simetričan": ako su α, β literali i (α, β) je grana u $\mathcal{H}_\phi$, tada je i $(\neg\beta, \neg\alpha)$ takođe grana. Zbog toga, u $\mathcal{H}_\phi$ postoji put $\alpha \rightsquigarrow \beta$ ako i samo ako postoji put $\neg\beta \rightsquigarrow \neg\alpha$.

Podsetimo, za dva čvora u, v nekog digrafa kažemo da su *jako povezani* ako postoje putevi $u \rightsquigarrow v$ i $v \rightsquigarrow u$, tj. ako u, v istovremeno leže na nekoj orijentisanoj šetnji (lako se pokazuje da je jaka povezanost relacija ekvivalencije na skupu čvorova). Traženi algoritam se sada zasniva na sledećem tvrđenju.

Lema 6.10 *Neka je ϕ 2-KNF. Formula ϕ je zadovoljiva ako i samo ako nikoja dva komplementarna literala nisu jako povezana u $\mathcal{H}_\phi$.*

Dokaz. ($\Rightarrow$) Pretpostavimo da su za neko iskazno slovo x, čvorovi x i $\neg x$ jako povezani u $\mathcal{H}_\phi$, ali da je formula ϕ zadovoljiva. Neka je τ valuacija za koju je $v_\tau(\phi) = \top$. Bez umanjenja opštosti, pretpostavimo da je $\tau(x) = \top$. Kako postoji put $x \rightsquigarrow \neg x$, na tom putu postoji grana (ℓ, ℓ') tako da je $v_\tau(\ell) = \top$ i $v_\tau(\ell') = \bot$. Međutim, egzistencija ove grane znači da je $(\neg\ell \vee \ell')$ konjunkt u ϕ. Ali, s obzirom da posmatranu valuaciju, vrednost ovog konjunkta je $\bot$. Kontradikcija.

($\Leftarrow$) Polazimo od pretpostavke da nikoja dva literala oblika $x, \neg x$ nisu jako povezana u $\mathcal{H}_\phi$. Tada ćemo sledećim iterativnim postupkom definisati jednu valuaciju u odnosu na koju će vrednost ϕ biti $\top$. Nije teško videti da je za to potrebno i dovoljno da u $\mathcal{H}_\phi$ ne postoji ni jedan put koji vodi iz čvora čija je vrednost $\top$ u čvor sa vrednošću $\bot$.

Željena valuacija se formira sledećim algoritmom:

1. Odaberi iskazno slovo x takvo da literalima $x, \neg x$ još nije dodeljena istinitosna vrednost. Ako takvo slovo ne postoji, stani.

2. Proveri da li postoji put $x \rightsquigarrow \neg x$. Ako postoji, stavi $\ell := \neg x$. U suprotnom, $\ell := x$.

3. Proširi domen i definiciju do ovog trenutka definisane (parcijalne) valuacije tako da ℓ, kao i svi čvorovi dostižni iz ℓ, dobiju vrednost $\top$ (pri tom, naravno, čvorovi koji su negacije ovih literala dobijaju vrednost $\bot$, što su upravo čvorovi *iz* kojih je dostižan $\neg\ell$).

4. Vrati se na korak 1.

Očigledno, kad god se u ovom algoritmu nekom čvoru dodeli vrednost $\top$, to isto važi i za sve čvorove koji su dostižni iz njega. Zato u odnosu na rezultujuću valuaciju ne može nastati put koji vodi iz "tačnog" u "netačan" čvor. Preostaje da se uverimo da je korak 3 logički ispravno definisan. Za to su dovoljne sledeće dve primedbe:

- Prilikom primene koraka 3, nije moguće da su iz literala ℓ istovremeno dostižni y i $\neg y$ za neko slovo y. U suprotnom, zbog simetrije $\mathcal{H}_\phi$ bi iz istih čvorova postojali putevi u $\neg\ell$. Zbog toga bismo imali put $\ell \rightsquigarrow \neg\ell$, što je u suprotnosti sa izborom načinjenim u koraku 2.

- Takođe, u koraku 3 nije moguće da iz ℓ vodi neki put u čvor ℓ' kome je ranije u algoritmu dodeljena vrednost $\bot$, jer bi u suprotnom, čvoru $\neg\ell'$ bila dodeljena vrednost $\top$ i postojao bi put $\neg\ell' \rightsquigarrow \neg\ell$, pa bi stoga $\neg\ell$ još ranije dobio vrednost $\top$, a samim tim, ℓ vrednost $\bot$ (što bi bila kontradikcija sa pretpostavkom da ℓ još nema dodeljenu vrednost).

Jasno, gornji algoritam se završava u konačno mnogo koraka, a njegov rezultat je, kao što smo pokazali, valuacija koja zadovoljava ϕ. □

Dakle, za ispitivanje zadovoljivosti formule ϕ dovoljno je sprovesti $2m$ pretraživanja u digrafu $\mathcal{H}_\phi$ (koji se konstruiše u linearnom vremenu u odnosu na $n = |\phi|$), kako bi se otkrili eventualni putevi $x_i \rightsquigarrow \neg x_i$ i $\neg x_i \rightsquigarrow x_i$, $1 \leqslant i \leqslant m$. Broj grana u digrafu $\mathcal{H}_\phi$ je reda $\mathcal{O}(n)$, pa se svaka njegova pojedinačna pretraga realizuje u linearnom vremenu. Takođe, $m = \mathcal{O}(n)$. Stoga se 2-SAT rešava u vremenu $\mathcal{O}(n^2)$.

Zadaci.

1. Primenom pravila rezolucije transformisati formulu

$$\phi = (x \vee y \vee z \vee t) \wedge (\neg x \vee y \vee z \vee t) \wedge (x \vee \neg y \vee \neg z \vee t) \wedge (\neg x \vee y \vee \neg z \vee \neg t)$$

u 3-KNF.

2. Za formulu φ iz Zadatka 6.3.1 konstruisati graf $\mathcal{H}_\varphi$, a zatim "ručnom" primenom algoritma iz dokaza Leme 6.10 odrediti jednu zadovoljavajuću valuaciju.

6.5 Problem $\neq$-SAT i problem obojivosti grafova

Razmotrimo najpre sledeću varijaciju problema 3-SAT:

ULAZ: 3-KNF ϕ.

IZLAZ: Da li postoji valuacija u odnosu na koju svaki konjunkt u ϕ ima bar po jedan tačan i netačan literal?

Ovo je problem *$\neq$-zadovoljivosti formule* ϕ (pošto se traži valuacija u odnosu na koju nijedan konjunkt nema sva tri literala iste vrednosti), u oznaci $\neq$-SAT. Traženu valuaciju iz ovog problema zovemo *$\neq$-valuacija*.

Ako za valuaciju τ sa $\overline{\tau}$ označimo njoj "komplementarnu" valuaciju (definisanu sa $\overline{\tau}(x) = v_\tau(\neg x)$ za sva iskazna slova x), tada veoma lako uočavamo da je za svaku $\neq$-valuaciju τ, $\overline{\tau}$ takođe $\neq$-valuacija. Prema tome, problem $\neq$-SAT možemo formulisati i tako da pitamo da li za datu 3-KNF ϕ postoji valuacija τ tako da τ i $\overline{\tau}$ istovremeno zadovoljavaju ϕ.

Teorema 6.11 *Problem* $\neq$-SAT *je* **NP***-kompletam.*

Dokaz. Pošto je očigledno da $\neq$-SAT $\in$ **NP**, tvrđenje će biti dokazano ako konstruišemo redukciju 3-SAT $\leqslant \neq$-SAT.

Za datu 3-KNF ϕ za svaki njen konjunkt C_i uvedimo novo iskazno slovo p_i (koje, jasno, ne figuriše u ϕ), kao i još jedno posebno slovo q. Formulu ϕ transformišemo u ϕ' (koja će takođe biti u 3-KNF) tako što za sve i, konjunkt

$$C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$$

zamenjujemo konjunkcijom dva nova konjunkta

$$(\ell_{i,1} \vee \ell_{i,2} \vee p_i) \wedge (\neg p_i \vee \ell_{i,3} \vee q).$$

Očito, formula ϕ' se konstruiše u polinomnom vremenu u odnosu na $|\phi|$. Tvrdimo da je ϕ zadovoljiva ako i samo ako je ϕ' $\neq$-zadovoljiva.

($\Rightarrow$) *ϕ je zadovoljiva.* Neka je τ valuacija za koju važi $v_\tau(\phi) = \top$. Želimo da konstruišemo valuaciju τ' koja $\neq$-zadovoljava ϕ'. Ovu valuaciju ćemo dobiti kao proširenje od τ: dovoljno će biti da na pogodan način "dodefinišemo" istinitosne vrednosti novouvedenih slova.

Dati uslovi povlače da za svako i postoji $j_i \in \{1, 2, 3\}$ tako da je $v_\tau(\ell_{i,j_i}) = \top$. Ako je $j_i \in \{1, 2\}$ definišimo $\tau'(p_i) = \bot$; ako je pak $v_\tau(\ell_{i,1}) = v_\tau(\ell_{i,2}) = \bot$ i $v_\tau(\ell_{i,3}) = \top$, stavljamo $\tau'(p_i) = \top$. U svim slučajevima je $\tau'(q) = \bot$. Sada se lako proverava da je po konstrukciji vrednost slova p_i uvek suprotna

vrednosti bar jednog od litarala $\ell_{i,1}$, $\ell_{i,2}$. S druge strane, imamo $v_{\tau'}(\neg p_i) = v_\tau(\ell_{i,3})$ ako i samo ako je ta vrednost $\top$ (to će se dogoditi kada je literal $\ell_{i,3}$ tačan, kao i bar jedan od $\ell_{i,1}$, $\ell_{i,2}$). Međutim, tada vrednost $\bot$ koju ima q obezbeđuje $\neq$-zadovoljivost formule ϕ'.

($\Leftarrow$) *ϕ' je $\neq$-zadovoljiva.* Neka je τ_1 valuacija koja $\neq$-zadovoljava ϕ'. Dokazaćemo da bar jedna od valuacija τ_1, $\overline{\tau_1}$ zadovoljava ϕ. Zaista, ako je $v_{\tau_1}(\phi) = \bot$, tada postoji i tako da je $v_{\tau_1}(\ell_{i,1}) = v_{\tau_1}(\ell_{i,2}) = v_{\tau_1}(\ell_{i,3}) = \bot$. Po izboru τ_1, tada mora biti $\tau_1(p_i) = \tau_1(q) = \top$. Međutim, sada ϕ ne može imati konjunkt u kome su sva tri literala tačna u odnosu na τ_1, $v_{\tau_1}(\ell_{i',1}) = v_{\tau_1}(\ell_{i',2}) = v_{\tau_1}(\ell_{i',3}) = \top$, jer bi u suprotnom moralo biti $\top(p_{i'}) = \bot$, pa bi konjunkt $(\neg p_{i'} \vee \ell_{i',3} \vee q)$ u ϕ' imao sva tri tačna literala u odnosu na τ_1 — kontradikcija sa izborom valuacije τ_1. Drugim rečima, u odnosu na valuaciju $\overline{\tau_1}$, nijedan konjunkt u ϕ ne sadrži sva tri netačna literala, tj. $\overline{\tau_1}$ zadovoljava ϕ. □

Problem $\neq$-SAT ćemo iskoristiti kako bismo pokazali **NP**-kompletnost jednog značajnog grafovskog problema: problema bojenja čvorova datog grafa u tri boje. Naime, pod *bojenjem (čvorova[22]) grafa $\mathcal{G} = (V, E)$ u k boja* podrazumevamo bilo koju funkciju $c : V \to \{0, \ldots, k-1\}$. Kažemo da je $\mathcal{G}$ *k-obojiv* ako postoji njegovo bojenje c u k boja tako da za svaki par susednih čvorova u, v važi $c(u) \neq c(v)$ (takvo bojenje nazivamo *pravilnim*). Drugim rečima, moguće je particionisati skup čvorova V u k klasa, $V = V_0 \cup \ldots \cup V_{k-1}$, tako da nijedna grana iz E nije incidentna sa čvorovima iz iste klase.

Dakle, problem k-obojivosti grafova k-COL (k je fiksiran broj > 0) glasi:

ULAZ: Graf $\mathcal{G}$.

IZLAZ: Da li je $\mathcal{G}$ k-obojiv?

Problem 1-COL je trivijalan, zato što je graf $\mathcal{G}$ 1-obojiv ako i samo ako je $E = \varnothing$. Dalje, 2-obojivost grafa $\mathcal{G}$ ekvivalentna je njegovoj bipartitnosti, tj. nepostojanju ciklusa neparne dužine — lako se možemo uveriti da se adekvatnom modifikacijom algoritma za pretraživanje grafova (uz BFS ili DFS tehniku pretrage) može dobiti polinomni algoritam koji za svaki čvor $u \in V$ utvrđuje da li u pripada neparnom ciklusu. Međutim, situacija je sasvim drugačija u slučaju $k = 3$.

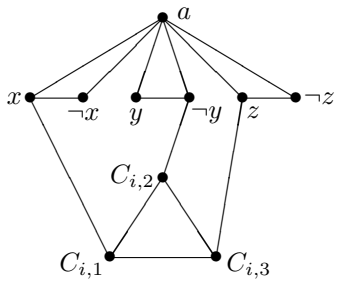
Teorema 6.12 *Problem* 3-COL *je* **NP***-kompletan.*

Dokaz. Važi 3-COL $\in$ **NP**, pošto verifikujući problem proverava pravilnost datog 3-bojenja (što očigledno troši vreme $\mathcal{O}(m)$). Naravno, sertifikat je pravil-

[22] U teoriji grafova je uobičajeno da se pod *bojenjem grafa* podrazumeva bojenje njegovih čvorova. Ako želimo da bojimo grane, to se posebno naglašava.

no 3-bojenje. Prema tome, preostaje da se pokaže da je razmatrani problem **NP**-težak. Ovo ćemo postići redukcijom $\neq$-SAT $\leqslant$ 3-COL. To znači da za proizvoljnu 3-KNF ϕ treba da konstruišemo graf $\mathcal{G}_\phi$ takav da $\mathcal{G}_\phi$ ima pravilno 3-bojenje ako i samo ako je ϕ $\neq$-zadovoljiva formula.

Čvorovi grafa $\mathcal{G}_\phi$ će biti sledeći: za svako iskazno slovo p koje se pojavljuje u ϕ imaćemo dva čvora p i $\neg p$, za svaki konjunkt C_i u ϕ definišemo tri čvora $C_{i,1}$, $C_{i,2}$ i $C_{i,3}$, i imamo i jedan specijalan čvor a. Za svako iskazno slovo p, čvorovi $a, p, \neg p$ čine trougao, kao i $C_{i,1}, C_{i,2}, C_{i,3}$ za sve i. Osim toga, čvorovi $C_{i,j}$ i ℓ (gde je ℓ neki literal) su povezani granom ako i samo ako se j-ti literal u C_i poklapa sa ℓ. Jasno, graf $\mathcal{G}_\phi$ se konstruiše u polinomnom vremenu u odnosu na $|\phi|$. Stoga prelazimo na dokaz tražene ekvivalencije.



Slika 6.5.1. Deo grafa $\mathcal{G}_\phi$ koji odgovara konjunktu $C_i = (x \vee \neg y \vee z)$

($\Rightarrow$) *$\mathcal{G}_\phi$ ima pravilno 3-bojenje*. Bez ograničenja opštosti, možemo pretpostaviti da je $c(a) = 2$. U tom slučaju, čvorovi koji odgovaraju literalima moraju biti obojeni jednom od boja $0, 1$. Definišimo valuaciju τ tako da je $\tau(p) = \top$ ako i samo ako je $c(p) = 1$. Tvrdimo da ova valuacija $\neq$-zadovoljava ϕ. Zaista, ako bi formula ϕ imala konjunkt C_i u kome svi literali imaju istu istinitosnu vrednost u odnosu na τ, to bi značilo da su u $\mathcal{G}_\phi$ čvorovi $C_{i,1}, C_{i,2}, C_{i,3}$ susedni sa čvorovima-literalima koji su svi iste boje. Sledilo bi da u trouglu $C_{i,1}, C_{i,2}, C_{i,3}$ figurišu samo dve boje, što protivreči pravilnosti uočenog bojenja.

($\Leftarrow$) *ϕ je $\neq$-zadovoljiva.* Neka τ $\neq$-zadvoljava ϕ. Sada konstruišemo pravilno 3-bojenje grafa $\mathcal{G}_\phi$ čitajući, u izvesnom smislu, prethodnu implikaciju "unazad". Obojimo a u boju 2, a čvorove koje odgovaraju literalima u boje 0 i 1 u zavisnosti od njihove istinitosne vrednosti: ako je $\tau(p) = \top$, bojimo čvor p u 1, a $\neg p$ u 0, dok u suprotnom slučaju postupamo obratno. Preostaje da obojimo trouglove koji odgovaraju konjunktima. Kako svaki konjunkt, po pretpostavci, sadrži po jedan tačan i netačan literal, izaberimo jedan takav par iz svakog konjunkta. U tom paru, čvor koji odgovara poziciji tačnog literala posmatranog

konjunkta obojimo u 0, a čvor koji odgovara poziciji netačnog literala u 1. Preostali čvor trougla koji odgovara razmatranom konjunktu bojimo u 2. Lako se proverava da je na ovaj način konstruisano bojenje pravilno. □

Zadaci.

1. Odrediti graf $\mathcal{G}_\phi$ iz dokaza Teoreme 6.12 za 3-KNF ϕ datu u Primeru 6.5. Naći jedno pravilno 3-bojenje tog grafa i odrediti odgovarajuću valuaciju koja $\neq$-zadovoljava ϕ.

2. Dokazati da je problem k-COL **NP**-kompletan za svako $k \geqslant 4$.

6.6 NP-kompletni problemi iz diskretne optimizacije

U ovom odeljku dajemo pregled nekih **NP**-kompletnih problema koji se javljaju uglavnom u tehničkim primenama matematike i operacionim istraživanjima. Kao što ćemo kasnije videti, svi ovi problemi predstavljaju specijalne slučajeve opšteg problema celobrojnog linearnog programiranja.

Najpre ćemo razmotriti problem *sume podskupa* (SP):

ULAZ: Konačan skup S, težinska funkcija $w : S \to \mathbb{N}$ i $B \in \mathbb{N}$ (tzv. *ciljni broj*).

IZLAZ: Da li postoji podskup $S' \subseteq S$ takav da je

$$\sum_{x \in S'} w(x) = B \quad ?$$

Par (S, w) je jedan od načina da se formalno definiše *multiskup* prirodnih brojeva. Veličina ulaznih podataka u ovom problemu je ukupna dužina zapisa brojeva $w(x)$, $x \in S$, i B, dakle, $\log B + \sum_{x \in B} \log w(x)$. Jednostavnosti radi, tu veličinu možemo asimptotski proceniti sa $n \log B$, gde je $n = |S|$.

Imajući u vidu ove primedbe, lako se vidi da SP $\in$ **NP**: sertifikat je podskup $S' \subseteq S$, a verifikacija se sastoji u sabiranju brojeva $w(x)$, $x \in S'$, oduzimanju B i poređenju dobijenog rezultata sa 0, što čini $|S'|$ sabiranja i oduzimanja ulaznih podataka i jednu primenu operacije JZERO. Stoga se verifikujući problem rešava u linearnom vremenu. Međutim, važi i više.

Teorema 6.13 SP *je* **NP**-*kompletan problem.*

Dokaz. Kako bismo dokazali ovo tvrđenje, daćemo redukciju 3-SAT $\leqslant$ SP. Ova redukcija polazi od 3-KNF ϕ i konstruiše instancu (S_ϕ, w_ϕ, B_ϕ) problema

SP tako da je ϕ zadovoljiva ako i samo ako se iz multiskupa (S_ϕ, w_ϕ) može odabrati podskup težine B_ϕ.

Pretpostavimo da se u formuli ϕ pojavljuju iskazna slova $x_1, \ldots, x_m$ i da ona ima k konjunkta. Definišemo

$$S_\phi = \{y_1, z_1, \ldots, y_m, z_m, u_1, v_1, \ldots, u_k, v_k\}.$$

Dakle, svakom iskaznom slovu x_i, $1 \leqslant i \leqslant m$, a takođe i svakom konjunktu C_j, $1 \leqslant j \leqslant k$, pridružili smo po jedan par elemenata skupa S_ϕ. Težine $w_\phi(x)$, $x \in S_\phi$, i ciljni broj B_ϕ će biti vrste u narednoj tabeli, shvaćene kao decimalni zapisi.

	x_1	x_2	x_3	x_4	$\cdots$	x_m	C_1	C_2	$\cdots$	C_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_m						1	0	0	$\cdots$	0
z_m						1	0	0	$\cdots$	0
u_1							1	0	$\cdots$	0
v_1							1	0	$\cdots$	0
u_2								1	$\cdots$	0
v_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
u_k										1
v_k										1
B_ϕ	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

U gornjoj levoj "četvrtini" tablice cifra 1 se nalazi na poljima (y_i, x_i) i (z_i, x_i), $1 \leqslant i \leqslant m$, dok se u donjem desnom delu 1 nalazi na poljima (u_j, C_j) i (v_j, C_j), $1 \leqslant j \leqslant k$ (ostala polja su prazna, ili sadrže 0, po potrebi).

U gornjem desnom delu tabele, polje (y_i, C_j) sadrži 1 ako i samo ako konjunkt C_j sadrži literal x_i, dok (z_i, C_j) sadrži 1 ako i samo ako C_j sadrži $\neg x_i$. Ponovo, sva ostala polja sadrže 0. U gornjem primeru, tablica je popunjena za formulu ϕ oblika

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \ldots) \wedge \ldots \wedge (\neg x_3 \vee \ldots).$$

Preostaje da se uverimo da je ovo zaista željena redukcija.

($\Rightarrow$) *ϕ je zadovoljiva.* Neka je τ valuacija koja je zadovoljava. Za svako i, $1 \leqslant i \leqslant m$, uključićemo tačno jedan od elemenata y_i, z_i u podskup $S' \subseteq S_\phi$, i to tako da $y_i \in S'$ ako $\tau(x_i) = \top$, a inače (ako $\tau(x_i) = \bot$) odabiramo $z_i \in S'$. Suma do sada odabranih elemenata je broj koji u decimalnom zapisu na prvih m mesta sleva ima sve jedinice, a na preostalih k mesta, cifre su $1, 2$ ili 3, pošto τ zadovoljava ϕ, pa svaki konjunkt sadrži između jednog i tri tačna literala. Kako bismo postigli traženi ciljni broj, dovoljno je u S' uključiti odgovarajući broj elemenata u_j, v_j; tačnije, stavićemo da bude $u_j \in S'$ ako konjunkt C_j sadrži najviše dva tačna literala u odnosu na τ, a $v_j \in S'$ ako C_j sadrži tačno jedan tačan literal. Sa ovako odabranim skupom S', težina njegovih elemenata je baš B_ϕ.

($\Leftarrow$) *S_ϕ ima podskup S' čiji je zbir težina B_ϕ.* Konstruisaćemo valuaciju τ koja zadovoljava ϕ. Najpre ćemo konstatovati nekoliko činjenica.

- Sve težine $w_\phi(x)$, $x \in S_\phi$, su brojevi koji se u decimalnom sistemu zapisuju isključivo ciframa 0 i 1.
- Svaka kolona gornje tablice sadrži ne više od pet jedinica. Stoga, pri sabiranju bilo koje kolekcije navedenih brojeva nikada neće doći do prenosa (tj. kolone se sabiraju međusobno nezavisno).
- Kako bi se u zbiru dobila cifra 1 u prvih m kolona, *tačno jedan* od elemenata y_i, z_i mora pripadati S' ($1 \leqslant i \leqslant m$).

Tražena valuacija je sada sledeća:

$$\tau(x_i) = \begin{cases} \top & y_i \in S', \\ \bot & \text{inače,} \end{cases}$$

gde je $1 \leqslant i \leqslant m$.

Tvrdimo da je $v_\tau(\phi) = \top$, budući da je zbir u zadnjih k kolona baš 3. Od tog zbira 3, u koloni koja odgovara konjunktu C_j, najviše 2 može da potiče od težina elemenata u_j, v_j. Zbog toga, skupu S' pripada bar jedan element oblika y_i ili z_i takav da se na polju koje odgovara tom elementu i C_j pojavljuje cifra 1. Ako je to element oblika y_i, to znači da se literal x_i javlja u konjunktu C_j. Kako je tada po gornjoj definiciji $\tau(x_i) = \top$, sledi da je $v_\tau(C_j) = \top$. Obratno, ako je u pitanju element tipa z_i, tada C_j sadrži literal $\neg x_i$, koji je tačan, jer je sada $\tau(x_i) = \bot$. Kako je u ovom razmatranju j bilo proizvoljno, sledi da je svaki konjunkt u ϕ tačan u odnosu na τ, što je i trebalo dokazati.

Na kraju, primetimo da se opisana tablica konstruiše u polinomnom vremenu. Ona sadrži $2(m+k)^2+m+k$ polja. Pošto je dužina svakog konjunkta ograničena, izračunavanje svake cifre u ovoj tabeli zahteva samo konstantan broj koraka. Zato se tabela formira u vremenu $\mathcal{O}(n^2)$, gde je $n = |\phi|$. □

Veoma srodni problemu SP su i sledeći.

- RANAC. Dat je konačan skup S, funkcija *težine* $w : S \to \mathbb{N}$, funkcija *dobiti* $b : S \to \mathbb{N}$ i dva ciljna broja $W, B \in \mathbb{N}$. Utvrditi da li postoji podskup $S' \subseteq S$ tako da važe nejednakosti

$$\begin{aligned} \sum_{x \in S'} w(x) &\leqslant W, \\ \sum_{x \in S'} b(x) &\geqslant B. \end{aligned}$$

  Intuitivno, možemo zamisliti objekte iz skupa S kao predmete koje želimo da ponesemo na kampovanje (u ruksaku, otuda i ime): svaki od njih ima neku numeričku karakteristiku $b(x)$ koji meri koliko je x zaista potreban, ali i zapreminu $w(x)$. Pri tome je W zapremina (odnosno nosivost) ruksaka.

- POLOVLJENJE. Dat je konačan skup S i funkcija težine $w : S \to \mathbb{N}$. Utvrditi da li postoji podskup $S' \subseteq S$ tako da je

$$\sum_{x \in S'} w(x) = \sum_{x \in S \setminus S'} w(x).$$

- BIN PACKING ("Pakovanje u korpe"). Dat je konačan skup S i funkcija *zapremine* $w : S \to \mathbb{N}$, zapremina korpe B i prirodan broj $k \geqslant 1$. Da li je moguće rasporediti elemente skupa S u ne više od k korpi?

Lako se vidi da svi nabrojani problemi pripadaju **NP**. Štaviše, svaki od njih je i **NP**-kompletan, kao što će to naredne (polinomne) redukcije pokazati.

SP $\leqslant$ POLOVLJENJE. Neka je (S, w, B) instanca problema SP, i neka je

$$\Sigma = \sum_{x \in S} w(x).$$

Konstruišemo multiskup (S_1, w_1) tako što definišemo $S_1 = S \cup \{a, b\}$ (gde $a, b \notin S$), i proširujemo funkciju w do w_1 tako da je $w_1(a) = N - B$ i $w_1(b) = N - (\Sigma - B)$, gde je N dovoljno veliki prirodan broj (dovoljno je uzeti $N > \Sigma$).

Sada ako postoji $S' \subseteq S$ tako da je težina S' jednaka B, tada je očito težina skupa $S' \cup \{a\}$ jednaka N, polovini ukupne težine $2N$ skupa S_1. Obratno, ako se S_1 može "prepoloviti", tada je jasno da a, b ne mogu biti u istoj "polovini", jer je njihova ukupna težina $2N - \Sigma > N$. Tako, neka je $S_1' \subseteq S_1$ skup čija je težina N. Bez umanjenja opštosti možemo pretpostaviti da $a \in S_1'$. Ali, tada je težina skupa $S' = S_1' \setminus \{a\}$ upravo $N - (N - B) = B$.

POLOVLJENJE $\leqslant$ RANAC. Polazimo od multiskupa (S, w) i konstruišemo instancu (S_1, w_1, b_1, W, B) problema RANAC. Lako se uočava da se sa $S_1 = S$, $w_1 = b_1 = w$ i $W = \lfloor \Sigma/2 \rfloor$, $B = \lceil \Sigma/2 \rceil$ dobija tražena redukcija.

POLOVLJENJE $\leqslant$ BIN PACKING. Očigledno, $B = \lfloor \Sigma/2 \rfloor$ i $k = 2$ daju potrebnu redukciju.

Svi navedeni problemi predstavljaju zapravo specijalne slučajeve problema *celobrojnog linearnog programiranja* (CLP). Ovaj problem, u formi problema odlučivanja, sastoji se u utvrđivanju egizstencije bar jednog celobrojnog rešenja $(x_1, \ldots, x_n)$ sistema linearnih nejednačina

$$\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i \qquad (1 \leqslant i \leqslant m),$$

gde su a_{ij}, b_i, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$, dati racionalni brojevi. (Jasno, ovi uslovi određuju u opštem slučaju oblast u $\mathbb{R}^n$ koja nastaje kao presek poluprostora. Problem CLP pita da li ta oblast sadrži celobrojnu tačku.) U formi optimizacijskog problema, za date koeficijente $c_j \in \mathbb{Q}$, $1 \leqslant j \leqslant n$, traži se maksimalna vrednost (nad $\mathbb{Z}^n$) linearne funkcije

$$\sum_{j=1}^{n} c_j x_j,$$

pri čemu vektor $(x_1, \ldots, x_n)$ zadovoljava gornje nejednakosti.

S druge strane, u klasičnom problemu *linearnog programiranja* (LP) nema ograničenja u smislu celobrojnosti rešenja x_j, $1 \leqslant j \leqslant n$, već se traži egzistencija proizvoljnog realnog rešenja (ili racionalnog rešenja, što je svejedno zbog gustine skupa $\mathbb{Q}^n$ u $\mathbb{R}^n$), tj. utvrđuje se da li date nejednačine definišu nepraznu oblast u $\mathbb{R}^n$. Pri tome pretpostavljamo da su svi koeficijenti koji opisuju problem racionalni brojevi (dati kao uređeni parovi uzajamno prostih celih brojeva). Klasično rešenje problema LP, tzv. *simpleks metod* dao je 1947.

George B. Dantzig. Ovaj metod je u praksi veoma upotrebljiv, iako je u opštem slučaju eksponencijalne složenosti. Da problem LP ima polinomno rešenje dokazao je 1979/80. *Leonid G. Hačijan* [12], i njegovo rešenje je danas poznato pod imenom *elipsoidni metod*. Ovaj metod ima prvenstveno teorijski značaj, jer zbog velikog stepena polinoma koji izražava složenost nije uspeo da se u praksi takmiči sa brzinom simpleks metoda. Međutim, *Narendra K. Karmarkar* [17] je 1984. pronašao još brži *metod unutrašnje tačke* i od tada je ubrzanje algoritama za LP i njihova implementacija (tzv. LP-solveri) postalo aktivna oblast u istraživanjima u primenjenoj matematici i informatici. Poboljšanja Karmarkarovog algoritma su i u praksi uspela da budu konkurentna sa simpleks metodom: procene su da su za velike sisteme u kojima je $n + m > 2500$ ovi savremeniji postupci brži.

Međutim, problem CLP je **NP**-kompletan. Naime, činjenicu da je CLP **NP**-težak potvrđuje redukcija SP $\leqslant$ CLP, za koju je dovoljno SP izraziti kao sistem nejednačina

$$0 \leqslant x_t \leqslant 1 \quad (t \in S),$$

$$\sum_{t \in S} w(t) x_t = B,$$

gde se $0 \leqslant x_t$ može ekvivalentno pisati kao $-x_t \leqslant 0$, dok se poslednji uslov na očigledan način zapisuje preko dve nejednačine. Da CLP uopšte pripada klasi **NP** dokazali su 1976. godine *I.Borosh* i *L.B.Treybig* [3]: ako gornji opšti sistem nejednačina ima celobrojno rešenje, tada postoji rešenje koje se može zapisati sa polinomno mnogo cifara u odnosu sa n, m i broj cifara potreban za zapisivanje brojilaca i imenilaca racionalnih brojeva a_{ij} i b_i.

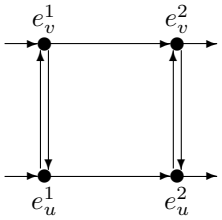
6.7 Problem Hamiltonovih kontura

Problem egzistencije Hamiltonove konture u datom grafu postoji u dve verzije: neorijentisanoj (HAM) i orijentisanoj ($\overrightarrow{\text{HAM}}$).

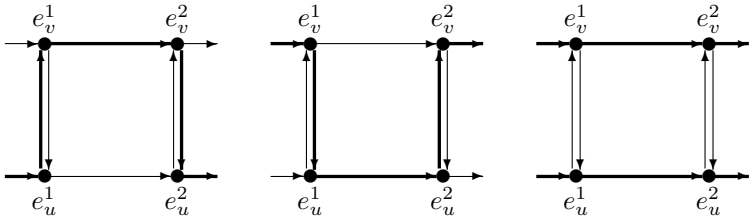
Teorema 6.14 *Problem* $\overrightarrow{\text{HAM}}$ *je* **NP***-kompletan.*

Dokaz. Pošto smo ranije već obrazložili pripadnost problema Hamiltonovih kontura klasi **NP**, pokazujemo **NP**-težinu ovog problema redukcijom nekog već poznatog **NP**-kompletnog problema na njega. U narednom ćemo opisati redukciju V-COVER $\leqslant$ $\overrightarrow{\text{HAM}}$: za dati par $(\mathcal{G}, k)$ koji se sastoji od grafa $\mathcal{G}$ i prirodnog broja k, konstruisaćemo digraf $\mathcal{H} = \mathcal{H}(\mathcal{G}, k)$ tako da $\mathcal{H}$ ima Hamiltonovu konturu ako i samo ako $\mathcal{G}$ ima čvorni pokrivač od k čvorova.

Konstrukciju digrafa $\mathcal{H}$ započinjemo tako što svakoj (neorijentisanoj) *grani* $e = uv$ grafa $\mathcal{G}$ pridružujemo sledeći digraf $\mathcal{H}_e$ (koji će biti podgraf od $\mathcal{H}$):



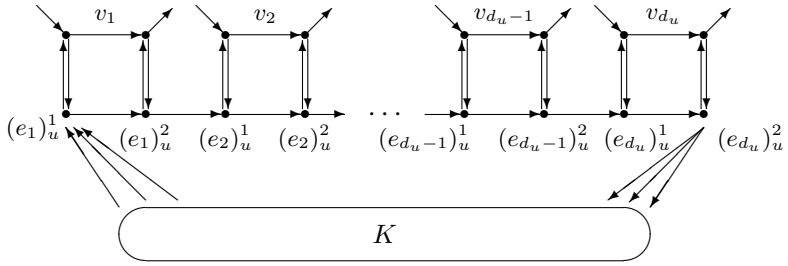
Iz drugih čvorova digrafa $\mathcal{H}$, grane će ulaziti isključivo u e_u^1 i e_v^1, a izlaziti ka drugim čvorovima isključivo iz e_u^2 i e_v^2. Pod takvim uslovima, postoji svega tri načina da neka Hamiltonova kontura u $\mathcal{H}$ (ako ona uopšte postoji) "prođe" kroz $\mathcal{H}_e$ (ovo ćemo kasnije i pokazati):



Sada prelazimo na opis digrafa $\mathcal{H}$. On će se sastojati iz svih podgrafova $\mathcal{H}_e$ za sve grane e grafa $\mathcal{G}$ i dodatnog skupa čvorova K koji ima tačno k elemenata. Tome ćemo dodati još neke orijentisane grane na sledeći način. Za proizvoljan čvor u grafa $\mathcal{G}$, neka je $Adj(u) = [v_1, \ldots, v_{d_u}]$ (gde d_u označava stepen čvora u), i neka je $e_i = uv_i$ za $1 \leqslant i \leqslant d_u$. U tom slučaju, dodajemo sledeće grane u $\mathcal{H}$:

- $(x, (e_1)_u^1)$ za sve $x \in K$,
- $((e_i)_u^2, (e_{i+1})_u^1)$ za sve $1 \leqslant i < d_u$,
- $((e_{d_u})_u^2, x)$ za sve $x \in K$.

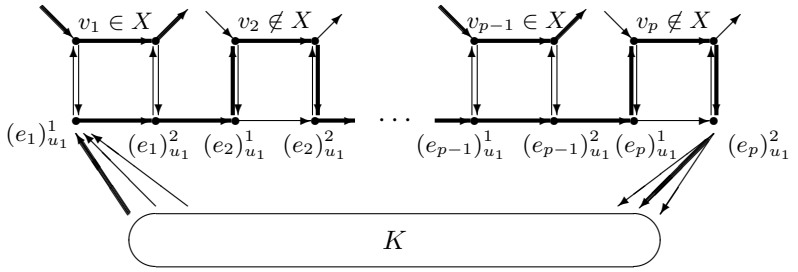
Kao rezultat ove konstrukcije, za svaki čvor u grafa $\mathcal{G}$ dobijamo podgrafove oblika $\mathcal{H}_{e_i}$ (pridružene svim granama incidentnim sa u u $\mathcal{G}$) "nanizane" po jednom usmerenom putu, i pri tome svi čvorovi iz K tuku "donji levi" čvor podgrafa $\mathcal{H}_{e_1}$, dok "donji desni" čvor podgrafa $\mathcal{H}_{e_{d_u}}$ tuče sve čvorove iz K. Vizuelno, ovu konfiguraciju (koju ćemo zvati u-*petlja*) možemo prikazati na sledeći način:



Prelazimo na dokaz korektnosti redukcije.

(⇒) *$\mathcal{G}$ ima čvorni pokrivač od k čvorova.* Neka je $\mathcal{G} = (V, E)$ i neka je $X = \{u_1, \ldots, u_k\} \subseteq V$ čvorni pokrivač. Dalje, neka je $K = \{w_1, \ldots, w_k\}$. Konstruisaćemo Hamiltonovu konturu u $\mathcal{H}$ koja nastaje kao unija usmerenih puteva $w_i \rightsquigarrow w_{i+1}$, $1 \leqslant i \leqslant k$, pri čemu je $w_{k+1} = w_1$. Svaki od ovih puteva će predstavljati jedan "prolazak" kroz u_i-petlju.

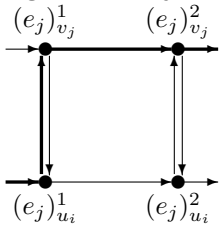
Neka je, dakle, čvor u_i incidentan redom sa granama $e_1 = u_i v_1, \ldots, e_p = u_i v_p$ u $\mathcal{G}$. Za svaki od navedenih suseda v_j, $1 \leqslant j \leqslant p$, postoje dve mogućnosti: $v_j \notin X$ ili $v_j \in X$. U prvom slučaju ćemo iz podgrafa $\mathcal{H}_{e_j}$ odabrati grane $((e_j)^1_{u_i}, (e_j)^1_{v_j})$, $((e_j)^1_{v_j}, (e_j)^2_{v_j})$ i $((e_j)^2_{v_j}, (e_j)^2_{u_i})$ (ovo odgovara prvom načinu "prolaska" kroz $\mathcal{H}_{e_j}$ sa prethodne strane). U suprotnom (ako $v_j \in X$), tada biramo samo granu $((e_j)^1_{u_i}, (e_j)^2_{u_i})$ (primetimo da će tada upravo zbog $v_j \in X$ — pošto je tada $v_j = u_{i'}$ za neko i' — biti odabrana i grana $((e_j)^1_{u_i}, (e_j)^1_{v_j})$, pa imamo treći način "prolaza" kroz $\mathcal{H}_{e_j}$). Osim toga, u put $w_i \rightsquigarrow w_{i+1}$ uključujemo svakako $(w_i, (e_1)^1_{u_i})$, $((e_p)^2_{u_i}, w_{i+1})$, kao i $((e_j)^2_{u_i}, (e_{j+1})^1_{u_i})$ za sve $1 \leqslant j < p$. Sledeća ilustracija u mnogome razjašnjava ovu konstrukciju:



Po opisanoj konstrukciji i pretpostavci da je X čvorni pokrivač (nema grana koje leže "van" X), svaki od čvorova digrafa $\mathcal{H}$ pripada tačno jednom putu $w_i \rightsquigarrow w_{i+1}$, pa unija ovih puteva čini Hamiltonovu konturu.

(⇐) *$\mathcal{H}$ ima Hamiltonovu konturu.* Razbijmo ovu konturu na k puteva oblika $w_i \rightsquigarrow w_{i+1}$, $1 \leqslant i \leqslant k$, gde je $K = \{w_1, \ldots, w_k\}$ i $w_{k+1} = w_1$, i posmatrajmo

jedan od tih puteva. Jasno, drugi čvor na ovom putu mora biti oblika $(e_1)^1_{u_i}$ za neki čvor u_i i neku granu e_1 incidentnu sa u_i (ta grana odgovara prvom čvoru sa liste $Adj(u_i)$). Ako je $|Adj(u_i)| = r$, tada indukcijom pokazujemo da uočeni put $w_i \rightsquigarrow w_{i+1}$ sadrži *sve* čvorove $(e_1)^1_{u_i}, (e_1)^2_{u_i}, (e_2)^1_{u_i}, \ldots, (e_r)^2_{u_i}$ i to baš u navedenom redosledu, pri čemu su svaka dva uzastopna čvora u nizu ili susedna na putu $w_i \rightsquigarrow w_{i+1}$, ili su na rastojanju 3 na tom putu. Zaista, pretpostavimo da je ovo tvrđenje tačno zaključno sa čvorom $(e_j)^1_{u_1}$. Ovaj čvor u $\mathcal{H}$ tuče svega dva čvora: $(e_j)^2_{u_i}$ i $(e_j)^1_{v_j}$, gde je $e_j = u_i v_j$. U drugom slučaju je jedina raspoloživa grana za nastavak Hamiltonove konture $((e_j)^1_{v_j}, (e_j)^2_{v_j})$. Izlazni stepen čvora $(e_j)^2_{v_j}$ je opet 2; međutim, Hamiltonova kontura sada ne može da se nastavlja duž v_j-petlje, jer bi u suprotnom čvor $(e_j)^2_{u_i}$ bio "zagrađen" (i tako izostavljen iz Hamiltonove konture, što je nemoguće) — njega tuku svega dva čvora, $(e_j)^1_{u_i}$ i $(e_j)^2_{v_j}$, a nijedna od odgovarajućih grana ne bi tada bila deo razmatrane konture. Naredna slika opisuje ovu nemoguću situaciju:



Zbog toga, sledeća grana puta $w_i \rightsquigarrow w_{i+1}$ mora biti $((e_j)^2_{v_j}, (e_j)^2_{u_i})$. Stoga se duž posmatranog puta iz $(e_j)^1_{u_1}$ do $(e_j)^2_{u_i}$ stiže u jednom ili tri koraka. Nakon toga, naredna grana očito mora biti $((e_j)^2_{u_i}, (e_{j+1})^1_{u_i})$ (ili $((e_r)^2_{u_i}, w_{i+1})$ u slučaju $j = r$).

Prema tome, zaključak je da put $w_i \rightsquigarrow w_{i+1}$ jednoznačno određuje čvor u_i grafa $\mathcal{G}$ (kao čvor kroz čiju petlju taj put prolazi) i da ona mora izgledati baš kao na drugoj slici s prethodne strane. Neka je X skup svih na ovaj način određenih čvorova. Jasno, $|X| = k$. Kako uočena Hamiltonova kontura "obilazi" svaki od podgrafova oblika $\mathcal{H}_e$ (i to, kao što smo upravo pokazali, na jedan od tri načina opisan na strani 145), svaka grana grafa $\mathcal{G}$ mora biti incidentna sa bar jednim čvorom iz X, tj. X je čvorni pokrivač u $\mathcal{G}$.

Na kraju, primetimo da ako $\mathcal{G}$ ima n čvorova i m grana, tada $\mathcal{H}$ ima $k + 4m$ čvorova, dok je broj grana

$$2k(n - |V_0|) + 6m + \sum_{u \in V \setminus V_0} (d_u - 1) = (2k - 1)(n - |V_0|) + 8m,$$

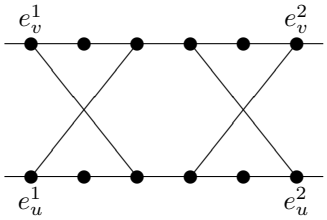
gde je $V_0 \subseteq V$ skup svih izolovanih čvorova grafa $\mathcal{G}$ (čvorova stepena 0). Na prvi pogled se može učiniti da su dimenzije digrafa $\mathcal{H}$ eksponencijalne u

odnosu na dimenzije ulaznih podataka (koje su $n, m, \log k$). Međutim, ovo se rešava primedbom da možemo pretpostaviti da je $k \leqslant n$, jer u suprotnom problem čvornog pokivača ima trivijalno negativno rešenje, pa tada $\mathcal{H}$ iz prethodne konstrukcije možemo zameniti bilo kojim fiksnim digrafom bez Hamiltonove konture. Uz ovakvu modifikaciju, $\mathcal{H}$ ima $\mathcal{O}(n+m)$ čvorova i $\mathcal{O}(n^2)$ grana. Digraf $\mathcal{H}$ se konstruiše na osnovu inspekcije lista susedstva grafa $\mathcal{G}$, dakle, u polinomnom vremenu, pa je teorema dokazana. □

Potpuno analogno se pokazuje da važi

Teorema 6.15 *Problem* HAM *je* **NP***-kompletan.*

U prethodnom dokazu dovoljno je digraf oblika $\mathcal{H}_e$ pridružen svakoj grani grafa $\mathcal{G}$ (vidi stranu 145) zameniti neorijentisanim grafom:



pri čemu su, naravno, i sve preostale grane u gornjem dokazu neorijentisane.

Kao prirodno uopštenje problema HAM za težinske grafove, javlja se *problem trgovačkog putnika* (TSP, od eng. *travelling salesman problem*). U formi problema odlučivanja, ovaj problem za dati težinski graf $\mathcal{G} = (V, E, w)$ i $k \in \mathbb{N}$ pita da li $\mathcal{G}$ ima Hamiltonovu konturu težine $\leqslant k$. Kao i u slučaju CLP, može se razmatrati i optimizacijska varijanta ovog problema, koja za težinski graf $\mathcal{G}$ traži minimalnu težinu Hamiltonove konture u $\mathcal{G}$, ukoliko ona uopšte postoji. TSP je **NP**-kompletan problem, jer se HAM očigledno redukuje na njega: dovoljno je dati graf $\mathcal{G}$ pretvoriti u težinski definišući $w(e) = 1$ sa sve grane e.

Zadaci.

1. Odrediti optimalnu turu za trgovačkog putnika koji obilazi Beograd i vojvođanske gradove, pri čemu je težiski graf puteva dat na Slici 4.7.1.

Literatura

[1] W.Ackermann, Zum Hilbertischen Aufbau der reelen Zahlen, *Math. Annalen* **99** (1928), 118–133.

[2] A.V.Aho, J.E.Hopcroft, J.D.Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, 1975.

[3] I.Borosh, L.B.Treybig, Bounds on positive integral solutions of linear diophantine equations, *Proc. Amer. Math. Soc.* **55** (1976), 299–304.

[4] S.N.Burris, *Logic for Mathematics and Computer Science*, Prentice-Hall, Upper Saddle River, 1998.

[5] A.Church, An unsolvable problem of elementary number theory, *Amer. J. Math.* **58** (1936), 345–363.

[6] S.A.Cook, The complexity of theorem-prooving procedures, u: *Proc. 3rd Symp. Foundations of Comput. Sci.*, pp.151–158, IEEE, 1971.

[7] T.H.Cormen, C.E.Leiserson, R.L.Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, 1990.

[8] E.W.Dijkstra, A note on two problems in connexion with graphs, *Numerische Math.* **1** (1959), 269–271.

[9] M.Ferenczi, *Matematikai logika*, Műszaki Könyvkiadó, Budapest, 2002.

[10] M.R.Garey, D.S.Johnson, *Computers and Intractability: a Guide to the Theory of NP-completeness*, W.H.Freeman, San Francisco, 1979.

[11] K.Gödel, Über formal unentschiedbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatshefte für Mathematik und Physik* **38** (1931), 173–198.

[12] L.G.Hačijan, Polinomni algoritmi u linearnom programiranju, *Žurnal Vičisliteljnoj Matematiki i Matematičeskoj Fiziki* **20** (1980), 53–72 (ruski).

[13] J.Hartmanis, R.E.Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965), 285–306.

[14] H.Hermes, *Enumerability, Decidability, Computability*, Springer-Verlag, Berlin, 1965.

[15] J.E.Hopcroft, J.D.Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979.

[16] N.Immerman, Nondeterministic space is closed under complementation, *SIAM J. Comput.* **17** (1988), 935–938.

[17] N.K.Karmarkar, A new polynomial-time algorithm for linear programming, *Combinatorica* **4** (1984), 373–395.

[18] R.M.Karp, Reducibility among combinatorial problems, u: (eds. R.E. Miller, J.W.Thatcher) *Complexity of Computer Computations*, pp.85–103, Plenum Press, New York, 1972.

[19] S.C.Kleene, General recursive functions of natural numbers, *Math. Annalen* **112** (1936), 727–742.

[20] D.E.Knuth, *The Art of Computer Programming*, Vol. I–III, Addison-Wesley, Reading, 1973.

[21] D.C.Kozen, *The Design and Analysis of Algorithms*, Springer-Verlag, New York, 1992.

[22] D.C.Kozen, *Automata and Computability*, Springer-Verlag, New York, 1997.

[23] J.Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, *Proc. Amer. Math. Soc.* **7** (1956), 48–50.

[24] L.A.Levin, Universal sorting problems, *Probl. Inform. Transmission* **9** (1973), 265–266.

[25] H.R.Lewis, C.H.Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, Englewood Cliffs, 1981.

[26] R.Sz.Madarász, S.Crvenković, *Uvod u teoriju automata i formalnih jezika*, Univerzitet u Novom Sadu, Stylos, Novi Sad, 1995.

[27] A.I.Mal'cev, *Algorithms and Recursive Functions*, Wolters-Noordhoff, Groningen, 1970.

[28] D.Mašulović, *Odabrane teme diskretne matematike*, Prirodno-matematički fakultet, Novi Sad, 2007.

[29] E.Mendelson, *Introduction to Mathematical Logic*, D.van Nostrand Co., Princeton, 1964.

[30] Ž.Mijajlović, Z.Marković, K.Došen, *Hilbertovi problemi i logika*, Zavod za udžbenike i nastavna sredstva, Beograd, 1986.

[31] S.Milić, *Elementi matematičke logike i teorije skupova*, Institut za matematiku, Novi Sad, 1981.

[32] V.Ya.Pan, Strassen's algorithm is not optimal, u: *Proc. 19th Symp. Foundations of Comput. Sci.*, pp.166–176, IEEE, 1978.

[33] C.H.Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, 1994.

[34] C.H.Papadimitriou, K.Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, 1982.

[35] S.Peinter, *Istorija srednjeg veka (284–1500)*, Clio, Glas srpski, Beograd, Banja Luka, 1997.

[36] V.Petrović, *Teorija grafova*, Univerzitet u Novom Sadu, 1999.

[37] L.Rónyai, G.Ivanyos, R.Szabó, *Algoritmusok*, Typotex, Budapest, 1999.

[38] W.J.Savitch, Relationship between nondeterministic and deterministic tape classes, *J. Comput. Syst. Sci.* **4** (1970), 177–192.

[39] J.C.Shepherdson, H.E.Sturgis, Computability of recursive functions, *J. ACM* **10** (1963), 217–255.

[40] M.Sipser, *Introduction to the Theory of Computation*, PWS Publishing Co., Boston, 1997.

[41] V.Strassen, Gaussian elimination is not optimal, *Numerische Math.* **13** (1969), 354–356.

[42] R.Szelepcsényi, The method of forcing for nondeterministic automata, *Bull. EATCS* **33** (1987), 96–100.

[43] R.E.Tarjan, *Data Strucures and Network Algorithms*, Regional Conference Series in Applied Mathematics, Vol.44, SIAM, Philadelphia, 1983.

[44] V.Tasić, *Matematika i koreni postmodernog mišljenja*, Svetovi, Novi Sad, 2002.

[45] R.Tošić, *Kombinatorika*, Univerzitet u Novom Sadu, 1999.

[46] R.Tošić, S.Crvenković, *Zbirka zadataka iz teorije algoritama*, Institut za matematiku, Novi Sad, 1980.

[47] A.M.Turing, On computable numbers with an application to the Entscheidungsproblem, *Proc. London Math. Soc.* **42** (1936), 230–265. Erratum: Ibid., **43** (1937), 544–546.

[48] D.Veljan, *Kombinatorika s teorijom grafova*, Školska knjiga, Zagreb, 1989.

[49] G.Vojvodić, *Predavanja iz matematičke logike i algebre*, Univerzitet u Novom Sadu, 1998.

[50] A.N.Whitehead, B.Russel, *Principia Mathematica I-III*, Cambridge University Press, Cambridge, 1910-1913.

Indeks