V. gimnazija
Zagreb, Klaićeva 1

Algoritmi u teoriji grafova

Učenik: Filip Nikšić
Mentor: prof. Petar Mladinić

Zagreb, svibanj 2004.

V. gimnazija
Zagreb, Klaićeva 1

Maturalni rad

Algoritmi u teoriji grafova

Učenik: Filip Nikšić, 4.f

Zagreb, svibanj 2004.

Sadržaj

1. Uvod

Puno sam vremena proveo razmišljajući o svojoj maturalnoj radnji. Još mnogo prije četvrtog razreda znao sam da će tema biti matematička, jer je to predmet koji mi se *prirodno nametnuo* tokom školovanja. Odlučivanje se svelo samo na izbor područja matematike o kojem ću pisati.

Neko vrijeme sam ozbiljno razmišljao da pišem o *matricama*. To zanimljivo područje pobudilo je u meni velik interes pa sam čak u nekoliko navrata držao predavanja o matricama u razredu, a napisao sam o njima i članak za naš matematičko-informatički časopis *Playmath*.

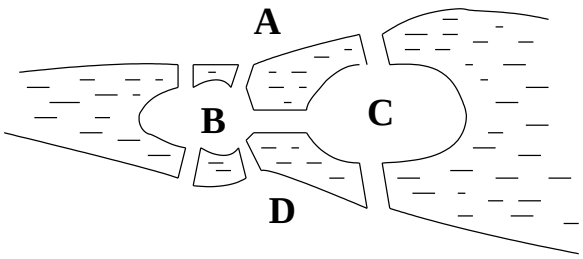
U zadnje vrijeme moju je pažnju snažno privukla i *teorija skupova*, za koju se može reći da zadire u same temelje matematike i da je moderna matematika bez nje nezamisliva. Neki tvrde da se čitava današnja matematika može izvesti iz teorije skupova.

No, i jedno i drugo su područja o kojima u mojoj glavi postoji još mnogo nepoznanica pa sam se odlučio držati nečega što nešto dulje proučavam i s čime sa sigurnošću mogu tvrditi da sam bolje upoznat.

S teorijom grafova susreo sam se i prije nego sam došao u srednju školu. Kako sam u svojim osnovnoškolskim danima vrlo uspješno sudjelovao na raznim informatičkim natjecanjima, bio je red da se upoznam s tim za informatiku neophodnim područjem matematike. Vrlo velik dio onoga što danas znam o grafovima dugujem predavanjima dr. Igora Urbihe u sklopu Zimske škole informatike u Osijeku početkom 2001. godine. Bilješke s tih predavanja su mi od neprocjenjive vrijednosti. Ostatak sam "pokupio" s raznih Internet stranica i iz raznih knjiga.

1.1. Povijesni pregled

Temelje teoriji grafova udario je dobro poznati švicarski matematičar **Leonhard Euler** (1707.–1783.) koji je 1736. godine riješio problem *königsberških mostova*. Naime, stari pruski grad Königsberg, današnji Kalinjingrad u Rusiji, smješten je na obalama rijeke Pregel. Dio grada nalazi se na dvije ade (rječna otoka), koje su povezane s kopnom i međusobno sa sedam mostova (vidi sliku 1.1.).

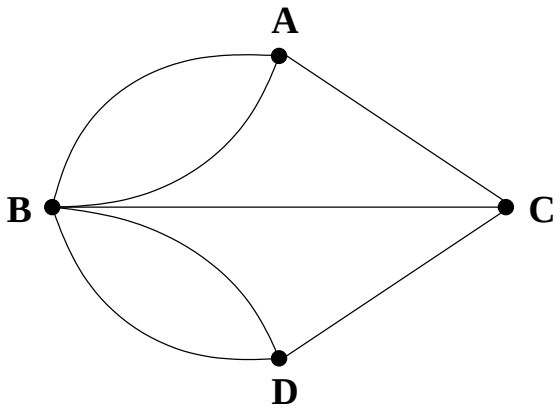


Slika 1.1. Königsberški otoci.

Građani Königsberga voljeli su šetati po mostovima, no ljutilo ih je što nitko nije mogao pronaći način da prošeta gradom tako da svih sedam mostova prijeđe samo jednom i zatim se vrati kući. Za pomoć su se obratili Euleru koji je u to vrijeme djelovao u Petrogradu. Euler je vrlo brzo pokazao da je takva šetnja nemoguća.

Pretpostavimo da je takva šetnja moguća. Tada ona završava u komponentama kopna A, B, C ili D, eventualno tamo gdje je započela. Primijetimo da svaka komponenta kopna u kojoj šetnja nije započela i nije završila mora biti spojena s parnim brojem mostova, jer za svaki most preko kojeg se dolazi na tu komponentu mora postojati i most preko kojeg se odlazi s te komponente. Zato svaka komponenta koja je spojena s neparnim brojem mostova mora biti ili početak ili kraj šetnje. No u slučaju Königsberga svaka komponenta A, B, C i D ima neparan broj mostova s kojima je povezana, a kako najviše dvije komponente mogu biti početak, odnosno kraj šetnje, zaključujemo da takva šetnja nije moguća.

Ako komponente A, B, C i D prikažemo kao točke ili vrhove, a mostove koji ih spajaju kao bridove ili spojnice, dobivamo pojednostavljenu shemu vrhova i njihovih spojnica, tj. graf.



Slika 1.2. Graf.

Problem šetanja po königsberškim mostovima ekvivalentan je problemu obilaska ovog grafa tako da krenemo iz jednog vrha i sve bridove prijeđemo točno jedanput.

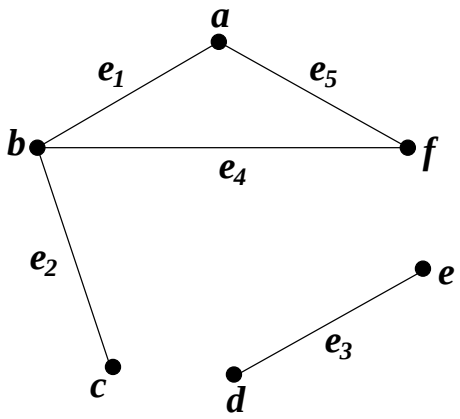
2. Teorija grafova

Postoji nebrojeno mnogo skupova s konačnim brojem elemenata među kojima postoje neke relacije. Primjerice, skup bi se mogao sastojati od nekoliko ljudi, životinja, država, kompanija, sportskih timova ili gradova; relacije između dva elementa A iz B takvog skupa bi mogle biti takve da osoba A pozna osobu B, životinja A se hrani životinjom B, država A graniči s državom B, kompanija A surađuje s kompanijom B, sportski tim A je igrao protiv tima B, grad A ima direktnu željezničku (ili cestovnu) liniju do grada B.

Takve i mnoge druge konačne skupove i strukture proučava matematička disciplina koja se zove *kombinatorna i diskretna matematika* ili kraće *kombinatorika* (više o tome u [3]). Vrlo važan dio kombinatorike su *grafovi*, kao jedna od osnovnih matematičkih struktura koja omogućuje modeliranje relacija između elemenata konačnih skupova.

2.1. Što je graf?

Definicija 2.1. Graf G je uređeni par $G = (V, E)$, gdje je V neprazan skup *vrhova*, a E je skup *bridova*. Svaki brid $e \in E$ spaja dva vrha $u, v \in V$ koji se zovu *krajevi* od e. Brid čiji se krajevi podudaraju zove se *petlja*, a ako dva ili više bridova povezuju isti par vrhova, zovu se *višestruki bridovi*. Ako graf sadrži višestruke bridove, zove se *multigraf*, a ako nema ni petlja ni višestrukih bridova, zove se *jednostavan graf*. Broj vrhova u grafu označavamo s $v(G)$, a broj bridova s $e(G)$.



Slika 2.1. Neusmjereni graf.

Primjer 2.1. Graf sa slike 2.1. definira se na slijedeći način:

$G = (V, E)$

$V = \{a, b, c, d, e, f\}$

$E = \{e_1, e_2, e_3, e_4, e_5\}$

$e_1 = \{a, b\}$

$e_2 = \{b, c\}$

$e_3 = \{d, e\}$

$e_4 = \{b, f\}$

$e_5 = \{f, a\}$

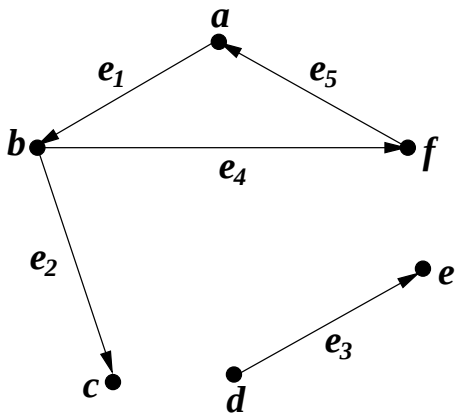
U prikazanom grafu bridovi su *neusmjereni*, tj. po svakom bridu se može "ići u oba smjera". Drugim riječima, u primjeru 2.1. se iz vrha a moglo doći u vrh b isto kao i iz vrha b u vrh a. No ponekad su grafovi *usmjereni* pa bridovi imaju svoj *smjer*. Na slikama se smjer označava strelicama.

Primjer 2.2. (Vidi sliku 2.2.) Kod zapisivanja koji su vrhovi pridruženi kojem bridu više ne koristimo vitičaste, nego okrugle zagrade jer kod usmjerenih grafova postaje bitno koji vrh je prvi naveden.

$e_1 = (a, b)$

$e_2 = (b, c)$

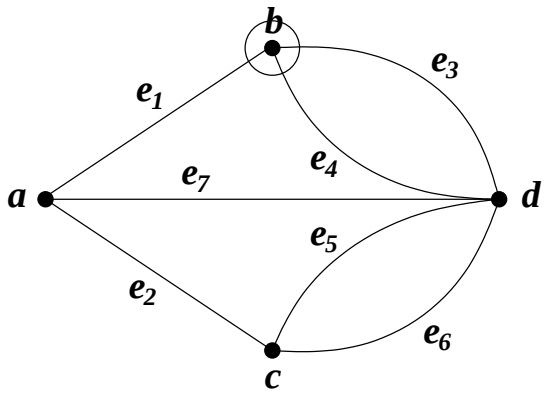
$e_3 = (d, e)$



Slika 2.2. Usmjereni graf.

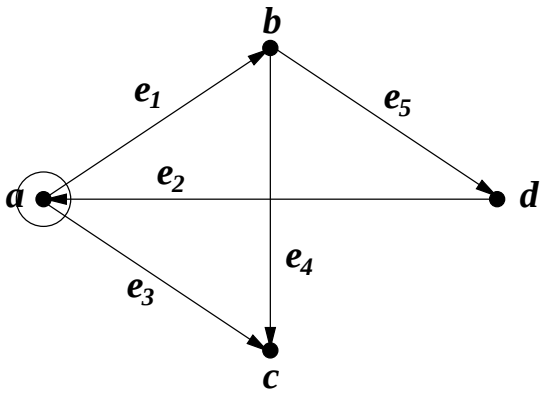
$e_4 = (b, f)$
$e_5 = (f, a)$

Definicija 2.2. *Stupanj vrha* je broj bridova s kojima je vrh spojen, tj. koji su *incidentni* vrhu. Kod usmjerenih grafova *izlazni stupanj* je broj bridova koji "izlaze" iz vrha, a *ulazni stupanj* je broj bridova koji "ulaze" u vrh. (Vidi slike 2.3. i 2.4.)



Slika 2.3. Stupanj vrha b je 3.

Vrlo često se javlja potreba da svakom bridu e grafa G pridružimo realan broj $w(e)$ koji zovemo *težina brida* e. Takav graf zovemo *težinski graf*. Na



Slika 2.4. Stupanj vrha a je 3, izlazni stupanj je 2,
a ulazni stupanj je 1.

primjer, ako vrhovi grafa predstavljaju gradove neke države, a bridovi ceste koje ih povezuju, težina brida može predstavljati duljinu ceste, ili cestarinu koju vozač mora platiti da se vozi tom cestom.

2.2. Šetnje, putovi i povezanost

Definicija 2.3. *Šetnja* u grafu G je naizmjeničan niz vrhova i bridova

$$(v_0 e_1 v_1 e_2 \dots v_{k-1} e_k v_k),$$

pri čemu je

$$v_0, v_1, v_2, \dots, v_k \in V,$$

$$e_1, e_2, e_3, \dots, e_k \in E.$$

Pritom je za usmjereni graf $e_i = (v_{i-1}, v_i)$, a za neusmjereni graf $e_i = \{v_{i-1}, v_i\}$, $1 \leq i \leq k$. U jednostavnom grafu šetnja je jednoznačno određena samo nizom svojih vrhova $(v_0 v_1 \dots v_k)$. Kažemo da je v_0 *početak*, a v_k *kraj* šetnje.

Tako primjerice na slici 2.3. imamo slijedeću šetnju: $(ae_1be_4de_6ce_6d)$.

Staza je šetnja bez ponovljenih bridova. Na slici 2.3. primjer staze je $(ae_1be_4de_3b)$.

Put je šetnja u kojoj su svi vrhovi različiti. Na slici 2.3. možemo vidjeti put ($ae_2ce_5de_3b$).

Ako su početak i kraj šetnje isti vrhovi, govorimo o *zatvorenoj šetnji* (na slici 2.3. to je npr. ($ae_1be_4de_4be_1a$)).

Ciklus je zatvorena staza. (Npr. ($ce_6de_3be_4de_5c$).) Ciklus je jednostavan ako u njemu nema ponovljenih vrhova osim početka i kraja. ($ae_1be_4de_5ce_2a$)

Eulerovi grafovi

Vjerojatno svi znaju za razne probleme crtanja grafova bez podizanja olovke s papira. Ti problemi, zajedno s mostovima Königsberga spadaju u najstarije probleme teorije grafova, a karakterizira ih to što se u biti radi o traženju zatvorene staze u grafu koja sadrži svaki brid ili, u "lakšim" slučajevima, o traženju staze koja ne mora biti zatvorena, a sadrži svaki brid.

Staza u grafu koja sadrži svaki brid zove se *Eulerova staza*, a ako je takva staza još i zatvorena, zove se *Eulerova tura*. Ako graf ima Eulerovu turu, zove se *Eulerov graf*.

Teorem 2.1. *Neusmjereni graf G je Eulerov ako i samo ako mu je svaki vrh parnog stupnja.*

Karakterizaciju Eulerova grafa znao je već i sam Euler 1736. godine, no dokaz ovog teorema nađen je tek 1873. Čitatelju je ostavljeno da ga sam pokuša izvesti, ili neka ga pročita u [3], str. 277.-278.

Iz teorema 2.1. dobiva se i slijedeći korolar:

Posljedica 2.1. *Neusmjereni graf G ima Eulerovu stazu ako i samo ako ima najviše dva vrha neparnog stupnja.*

Dokaz ovog korolara vrlo je sličan dokazu da ne postoji Eulerova staza u slučaju königsberških mostova (str. 5.), a može se vidjeti u [3], str. 278.

Hamiltonovi ciklusi

Za razliku od obilaska svih bridova, zanimljivi su i problemi obilaska svih vrhova koje je proučavao irski matematičar **William R. Hamilton** (1805.-1865.).

Put u grafu koji sadrži svaki njegov vrh zove se *Hamiltonov put*, a jednostavan ciklus koji sadrži sve vrhove zove se *Hamiltonov ciklus*. Ako graf ima Hamiltonov ciklus, zove se *Hamiltonov graf*.

Problem egzistencije i traženja Hamiltonovog ciklusa u grafu mnogo je teži od problema Eulerove ture i staze. Inače je poznatiji *problem trgovačkog putnika*: za n gradova, pronađi najkraći put koji prolazi kroz sve gradove. Do danas još nisu otkrivni efikasni algoritmi koji rješavaju te probleme.

Povezanost

Za graf kažemo da je *povezan* ako postoji put između svaka dva vrha. Graf na slici 2.1. nije povezan.

Komponenta grafa je najveći podgraf (podskup vrhova i podskup pripadnih bridova) takav da postoji put između svaka dva vrha komponente. Graf na slici 2.1. ima dvije komponente: $\{a, b, c, f\}$ i $\{d, e\}$.

Ako u usmjerenom grafu postoji put od svakog vrha do svakog drugog vrha, kažemo da je takav graf *strogo povezan*. *Strogo povezana komponenta* usmjerenog grafa je vrh u i skup svih vrhova v tako da postoji put iz u u v, i iz v u u.

2.3. Reprezentacija grafova

Za algoritamsku obradu nekog grafa potreban nam je sustavan i praktičan način da prikažemo njegove vrhove i bridove. Ovo je posebno bitno kod računala jer graf moramo pohraniti u njegovu ograničenu memoriju. Dobar izbor reprezentacije vrlo je važan jer, ovisno o algoritmu koji primjenjujemo i ovisno o vrsti grafa koji obrađujemo, različite reprezentacije zauzimaju vrlo

različite količine memorije, a i vrijeme izvršavanja algoritma jako varira za svaku.

Kako vrhove obično numeriramo i možemo ih indeksirati pomoću njihovog broja, reprezentacije se usmjeravaju na pohranjivanje bridova.

Lista bridova

Način prikazivanja grafa koji je najlogičniji i odmah se nameće je praćenje liste parova vrhova koji predstavljaju krajeve bridova. Ovaj način smo već susreli u primjerima 2.1. i 2.2. (str. 7.).

Lista bridova se jednostavno implementira u računalo i koristi malo njegove memorije, no određivanje bridova koji su incidentni određenom vrhu je vremenski skupo, kao i određivanje jesu li dva vrha susjedna.

Kod težinskih grafova za svaki brid pamtimo još i težinu brida, kod usmjerenih grafova pamtimo uređene umjesto neuređenih parova, a prikaz multigrafova je isto trivijalan.

Matrica susjedstva

Svakom grafu s $v(G)$ vrhova možemo pridružiti kvadratnu matricu M dimenzija $v(G) \times v(G)$ u čijem se i-tom retku i j-tom stupcu nalazi broj bridova koji spajaju i-ti i j-ti vrh. Takva matrica naziva se *matrica susjedstva*. Graf sa slike 2.3. ima slijedeću matricu susjedstva:

	a	b	c	d
a	0	1	1	1
b	1	0	0	2
c	1	0	0	2
d	1	2	2	0

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 0 & 2 \\ 1 & 2 & 2 & 0 \end{bmatrix}$$

Ovaj način reprezentacije grafa je isto jednostavan za implementaciju u računalo, ali zauzima mnogo više memorije od liste bridova. Vremenski je skupo otkriti je li određeni brid incidentan određenom vrhu, no provjera jesu li dva vrha susjedna obavlja se u jednom koraku.

Kod jednostavnih težinskih grafova na mjesto (i, j) u matrici dolazi težina brida, a za težinske multigrafove ovo nije lako proširiti.

Matrica incidencije

Graf G je moguće prikazati i *matricom incidencije*, $v(G) \times e(G)$ matricom u čijem se i-tom retku i j-tom stupcu nalazi broj (0, 1 ili 2) koliko su puta vrh v_i i brid e_j incidentni. Kako je ova matrica u pravilu mnogo veća od matrice susjedstva, rijetko se koristi.

Liste susjedstva

Kod grafova s relativno malo bridova (u usporedbi s brojem vrhova) u matrici susjedstva se pojavljuje mnogo nula, pa u tom slučaju ona i nije najefikasniji način pohrane grafa u računalo. Kod takvih grafova koriste se *liste susjedstava* – vektori od $v(G)$ elemenata koji predstavljaju vrhove, gdje je i-tom elementu pridružena lista susjednih vrhova od vrha v_i.

Ovaj način je teži za implementaciju od ostalih, no zauzima otprilike jednako memorije kao lista bridova. Traženje vrhova susjednih određenom vrhu je vrlo brzo, no provjera jesu li dva vrha susjedna nešto je sporija.

Proširenje za spremanje težinskih grafova je jednostavno (za svaki vrh u listama pamtimo dodatan broj – težinu), multigrafovi se mogu prikazati bez proširenja, a usmjerene grafove je isto jednostavno prikazati.

Ostale reprezentacije

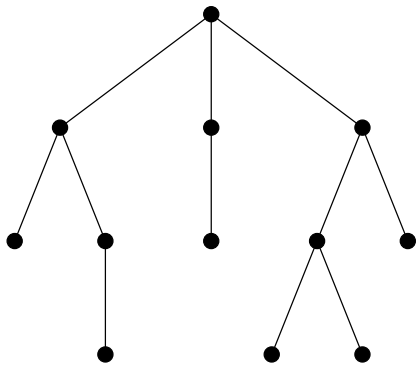
Za neke specifične slučajeve sam graf uopće nije potrebno pohraniti, jer je lako izračunati susjedne vrhove, provjeriti susjedstvo i utvrditi sve bridove bez stvarnog pohranjivanja informacija.

Primjerice, imamo slijedeći problem: Za zadanu šahovsku ploču veličine $n \times n$, pronađi najmanji broj poteza da skakač s gornjeg-lijevog polja dođe na polje (i, j). Ovdje imamo slijedeći graf: vrhovi su polja ploče, a ako se s jednog vrha može "skočiti" na drugi, između njih postoji brid. Umjesto

da koristimo matricu susjedstva koja bi za klasičnu 8×8 ploču bila veličine 64×64, mi vrlo lako za zadanu poziciju na ploči možemo izračunati na koja sve polja skakač može skočiti i to možemo iskoristiti u rješavanju problema.

2.4. Posebni grafovi

Neusmjereni graf koji ne sadrži cikluse i povezan je zove se *stablo*.



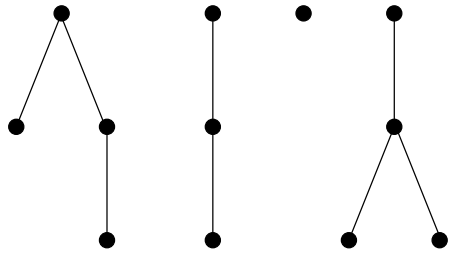
Slika 2.5. Stablo.

Stabla su najjednostavniji, ali i najvažniji grafovi jer su na neki način svi ostali grafovi sagrađeni od njih. Čest je problem traženja *razapinjućeg stabla* u grafu – stabla koje sadrži sve vrhove grafa, ali samo one bridove koji su dovoljni za postojanje puta između svaka dva vrha u stablu (takvih bridova ima $v(G) - 1$). O razapinjućim stablima bit će riječi u kasnijim poglavljima. Također, stabla se koriste u dvije fundamentalne vrste algoritama u računalnoj znanosti: sortiranju i pretraživanju.

Ponekad je neki vrh stabla istaknut kao *korijen* (na slikama je to obično vrh koji je na vrhu stabla). U tom slučaju možemo definirati relaciju *roditelj-dijete*: svaki vrh osim korijena ima jednog roditelja (susjedni vrh koji je bliže korijenu) i bilo koji broj djece (ostali susjedni vrhovi).

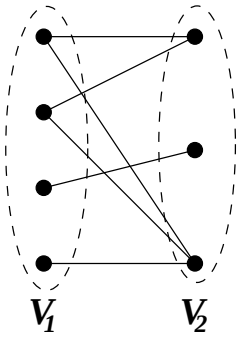
Neusmjereni graf koji ne sadrži cikluse zove se *šuma*.

Za graf kažemo da je *potpun* ako postoji brid između svaka dva para vrhova.



Slika 2.6. Šuma.

Bipartitni graf je graf čije vrhove možemo podijeliti u dva podskupa V_1 i V_2 tako da ne postoji brid između dva vrha u V_1 ili između dva vrha u V_2.



Slika 2.7. Bipartitni graf.

3. Osnovni obilasci grafova

Obilazak ili pretraživanje grafa je sistematičan prolazak njegovim bridovima kako bi se obišli svi njegovi vrhovi. Algoritmi koji obavljaju takav prolazak otkrivaju mnogo o strukturi grafa. Primjerice, pomoću njih možemo saznati je li graf povezan i od kojih se komponenti sastoji ako nije, ima li u njemu ciklusa i sl. Zbog toga su algoritmi za pretraživanje temelj većine ostalih algoritama u teoriji grafova, tj. možemo reći da su ostali algoritmi nadgradnje algoritama za pretraživanje.

Dva najčešća algoritma za obilazak ili pretraživanje grafa su *pretraživanje u širinu* (engl. *Breadth-First Search*) i *pretraživanje u dubinu* (engl. *Depth-First Search*).

3.1. Pretraživanje u širinu

Za zadani graf $G = (V, E)$, i početni vrh s, pretraživanje u širinu sistematično istražuje bridove grafa da bi "otkrilo" svaki vrh dostupan iz s. Ideja je da se iz vrha s prvo posjete svi susjedni vrhovi od s, zatim "susjedi susjeda" itd. sve dok se ne posjete svi vrhovi dostupni iz s. Pritom se računa *udaljenost* (najmanji broj bridova) od s do svih dostupnih vrhova (dakle, ovaj algoritam usput rješava problem najkraćeg puta od s do ostalih vrhova).

Ime je dobio po tome što se granica između otkrivenih i neotkrivenih vrhova "širi" kroz graf, tj. algoritam prvo otkriva vrhove na udaljenosti k od s, a tek onda one na udaljenosti $k + 1$.

Da bi pratio napredak, ovaj algoritam *boji* vrhove bijelom, sivom ili crnom

bojom. Na početku su svi vrhovi bijeli. Kad je neki vrh prvi put otkriven, on postaje siv, a kad sve njemu susjedne bijele vrhove obojimo u sivo, postaje crn. Na taj način su svi susjedi nekog crnog vrha sivi ili crni, dok sivi vrh može imati bijele vrhove među svojim susjedima. Sivi vrhovi predstavljaju granicu između otkrivenih i neotkrivenih vrhova.

Ovome odgovara struktura podataka *red (queue)* uz koju vežemo dvije operacije: možemo *staviti* nešto na kraj reda i *uzeti* nešto s početka reda. U redu će uvijek biti sivi vrhovi; uzet ćemo jedan sivi vrh s početka reda, sve njegove bijele susjede staviti na kraj reda (pritom će oni postati sivi) i obojiti vrh u crno.

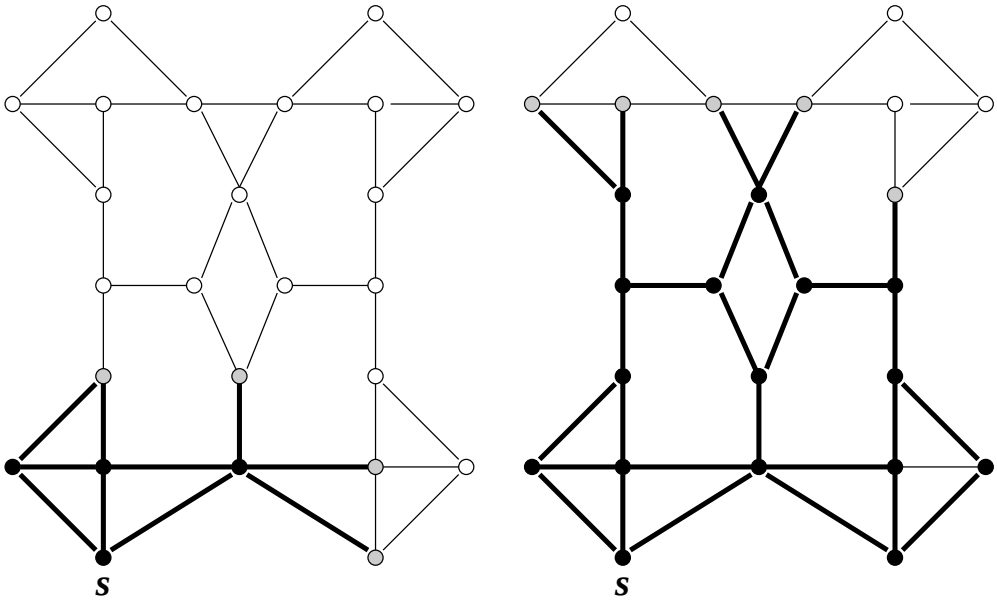
Za reprezentaciju grafa koristit ćemo liste susjedstva *lista* gdje $lista(x)$ predstavlja listu susjednih vrhova vrhu x. S $d(x)$ ćemo označiti najmanji broj bridova od s do x, a s $boja(x)$ boju vrha x.

Algoritam 3.1. Pretraživanje u širinu

1. Za svaki vrh $u \in V \setminus \{s\}$ postavi $d(u) \leftarrow \infty$ i $boja(u) \leftarrow$ bijela.
2. $d(s) \leftarrow 0$.
3. $boja(s) \leftarrow$ siva.
4. Stavi s na kraj *reda*.
5. Dok je *red* neprazan radi slijedeće:
   - 5.1. Uzmi u s početka *reda*.
   - 5.2. Za svaki vrh $v \in lista(u)$, ako je $boja(v) =$ bijela, radi slijedeće:
     - 5.2.1. $d(v) \leftarrow d(u) + 1$.
     - 5.2.2. $boja(v) \leftarrow$ siva.
     - 5.2.3. Stavi v na kraj reda.
   - 5.3. $boja(u) \leftarrow$ crna.

Algoritam radi na slijedeći način: Korak 1. postavlja sve udaljenosti na nulu, kao i sve boje na bijelu. Koraci 2.–4. postavljaju udaljenost početnog vrha na 0, boje ga u sivo (jer je on otkriven na početku) i stavljaju ga u red. Petlja pod 5. korakom je glavni dio. Dok god ima sivih vrhova, red je neprazan i sadrži vrhove koji su otkriveni, ali njihove liste susjedstva još

nisu pregledane. Algoritam redom uzima sive vrhove s početka reda (korak 5.1.), pregledava vrhove u njihovim listama susjedstva (korak 5.2.) i, ako su bijeli, otkriva ih, boji u sivo, postavlja im udaljenost $d(v)$ na $d(u)+1$ i stavlja ih u red (koraci 5.2.1.–5.2.3.). Konačno, kad su svi vrhovi u listi susjedstva pregledani, vrh u postaje crn (5.3.).



Slika 3.1. Pretraživanje u širinu.

Ako ovakvim obilaskom kroz graf $G = (V, E)$ konstruiramo stablo (podgraf) $T = (V, E_T)$ tako da za svaki bijeli susjedni vrh v vrha u dodamo brid (u, v) u E_T, dobit ćemo *razapinjuće stablo* grafa G poznato pod nazivom *breadth-first search tree*. U ovom stablu jasno se vide najkraći putovi od vrha s do ostalih vrhova; najkraći put u grafu G od vrha s do vrha u odgovara putu od s do u u razapinjućem stablu T.

3.2. Pretraživanje u dubinu

Drugi algoritam za obilazak grafa je *pretraživanje u dubinu*. Ovaj algoritam također sistematično pronalazi sve vrhove, ali na nešto drukčiji način od

pretraživanja u širinu. Naime, umjesto da se širi u grafu, on nastoji uvijek ići "dublje" u graf, tj. za svaki susjedni vrh v vrha u pretraživanje u dubinu se rekurzivno poziva i provjerava susjede vrha v prije nego se vrati i provjeri preostale susjede vrha u.

Slično kao kod pretraživanja u širinu, i pretraživanje u dubinu boji vrhove kako bi pratilo napredak. Svaki vrh je na početku bijel, kad je *otkriven* postaje siv, a kad je *završen* (tj. kad su mu svi vrhovi na listi susjedstva otkriveni i provjereni) postaje crn.

Opet za reprezentaciju koristimo liste susjedstva, a svakom vrhu v ćemo ovim algoritmom pridružiti broj $br(v)$ od 1 do $v(G)$.

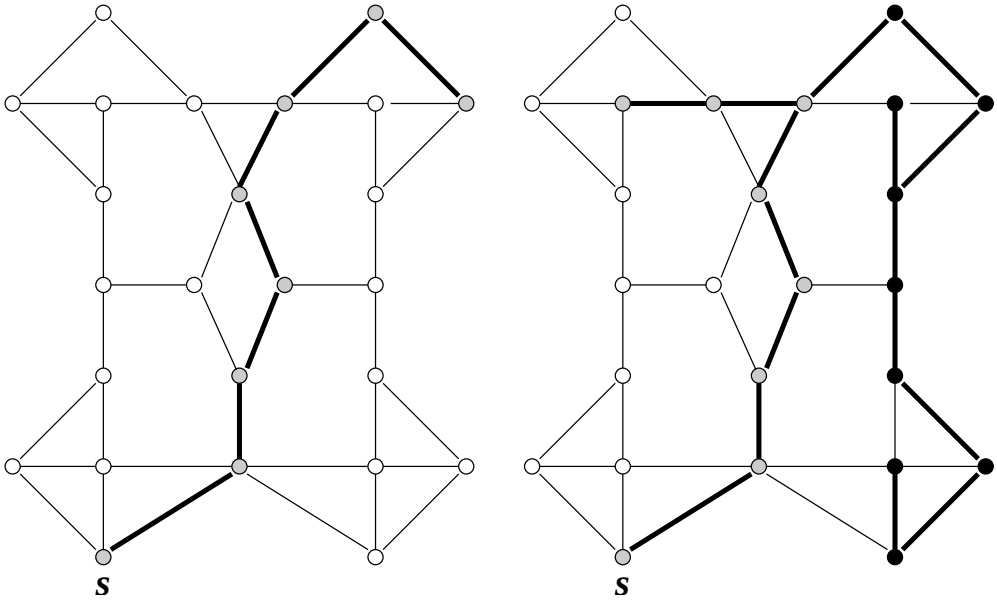
Algoritam 3.2. Pretraživanje u dubinu

1. Za svaki vrh $u \in V$ postavi $br(u) \leftarrow 0$ i $boja(u) \leftarrow$ bijela.
2. $i \leftarrow 0$.
3. *Posjeti vrh s*.

4. Počni proceduru *Posjeti vrh u*.
   4.1. $boja(u) \leftarrow$ siva.
   4.2. $i \leftarrow i + 1$.
   4.3. $br(u) \leftarrow i$.
   4.4. Za svaki vrh $v \in lista(u)$, ako je $boja(v) =$ bijela, *Posjeti vrh v*.
   4.5. $boja(u) \leftarrow$ crna.

Algoritam radi na slijedeći način: U koracima 1. i 2. svi brojevi $br(u)$ se postavljaju na nulu, sve boje se postavljaju na bijelu i inicijalizira se brojač i. U koraku 3. poziva se procedura *Posjeti vrh u* za početni vrh s. Korakom 4. započinje glavni dio algoritma—rekurzivna procedura *Posjeti vrh u*: u koracima 4.1.–4.3. boja vrha u postaje siva, brojač se uvećava za 1 i pridružuje se varijabli $br(u)$. Korakom 4.4. za svaki susjedni vrh v vrha u procedura *Posjeti vrh u* se rekurzivno poziva. Konačno, posljednjim korakom 4.5. boja vrha u postaje crna, što znači da su svi njegovi susjedni vrhovi otkriveni i završeni.

Slično konstrukciji breadth-first search stabla kod pretraživanja u širinu,



Slika 3.2. Pretraživanje u dubinu.

pretraživanjem u dubinu može se konstruirati razapinjuće stablo *depth-first search tree*.

4. Najmanje razapinjuće stablo

Recimo da imamo slijedeći problem: n gradova treba povezati cestama tako da uvijek postoji put između dva grada, a da troškovi izgradnje cesta budu minimalni (pritom znamo koliko košta izgradnja ceste između svaka dva grada). Ovaj problem može se svesti na općenitiji problem traženja "najpovoljnijeg" načina za povezivanje svih vrhova grafa, tj. na problem traženja *najmanjeg ili minimalnog razapinjućeg stabla.*

Definicija 4.1. Najmanje razapinjuće stablo grafa $G = (V, E)$ je njegov podgraf $T = (V, E_T)$ koji sadrži sve vrhove V i samo one bridove koji su dovoljni za postojanje puta između svaka dva vrha, a da pritom zbroj težina tih bridova bude najmanji mogući. Najmanje razapinjuće stablo ne mora biti jedinstveno.

Primijetimo da nema smisla govoriti o najmanjem razapinjućem stablu u netežinskom grafu budući da su u tom pogledu sva razapinjuća stabla "jednaka" (jer bridovi nemaju težina).

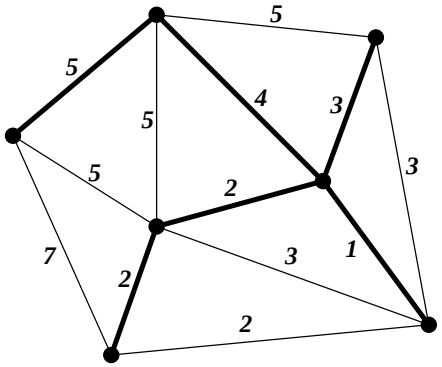
Postoji više načina da se pronađe najmanje razapinjuće stablo. U ovom poglavlju pokazat ću dva takva načina, a oba se temelje na slijedećem svojstvu:

Svojstvo 4.1. *Za bilo koju podjelu vrhova grafa u dva skupa, najmanje razapinjuće stablo sadrži brid najmanje težine koji povezuje neki vrh u prvom skupu s nekim vrhom u drugom skupu.*

Ovo svojstvo lako se dokazuje kontradikcijom. Neka je s brid najmanje težine koji povezuje dva skupa vrhova. Pretpostavimo da s nije u najmanjem

razapinjućem stablu. Promotrimo sada graf koji nastaje kad najmanjem razapinjućem stablu dodamo brid s: takav graf ima ciklus u kojem postoji još jedan brid koji povezuje dva skupa vrhova. Brisanjem tog brida i dodavanjem s dobivamo manje razapinjuće stablo što je u suprotnosti s pretpostavkom da s nije dio najmanjeg razapinjućeg stabla.

Najmanje razapinjuće stablo, dakle, možemo konstruirati tako da krenemo od bilo kojeg vrha i uvijek dodajemo vrh koji je "najbliže" već dodanim vrhovima. Drugim riječima, pronađemo brid najmanje težine koji povezuje vrhove koji su već dio stabla i vrhove koji to još nisu, i dodamo stablu taj brid i vrh do kojeg on vodi (u slučaju da postoje dva brida najmanje težine, bilo koji će poslužiti). Svojstvo 4.1. nam jamči da je dodani brid dio najmanjeg razapinjućeg stabla.



Slika 4.1. Najmanje razapinjuće stablo.

4.1. Primov algoritam

Za Primov[1] algoritam možemo reći da je generalizacija algoritama za obilazak grafa iz prethodnog poglavlja. Naime, kako je pretraživanje u dubinu rekurzivan algoritam, rekurziju je moguće maknuti korištenjem strukture podataka *stog (engl. stack)* koja slično kao red ima dvije operacije: možemo

[1] **Robert Prim**, američki matematičar, 1957. je otkrio algoritam za traženje najmanjeg razapinjućeg stabla.

nešto *staviti* na vrh stoga i *uzeti* nešto s vrha stoga (prisjetimo se, kod reda smo stavljali na kraj reda, a uzimali s početka reda). Pretraživanje u širinu i dubinu se, dakle, u suštini razlikuju samo po organizaciji strukture podataka za pamćenje "sivih" vrhova.

Struktura podataka koja se koristi za Primov algoritam je *prioritetni red (engl. priority queue)*. Podatci su organizirani u prioritetnom redu po nekom prioritetu. Kad dodamo novi podatak, on se ovisno o prioritetu pomiče bliže početku ili bliže kraju prioritetnog reda. Prema potrebi, prioritet podatka možemo mijenjati. Kad nam je potreban podatak iz prioritetnog reda, uzimamo onaj s početka. Ispada da određivanjem prikladnih prioriteta, prioritetni red postaje red, odnosno stog.

U Primovom algoritmu krećemo s jednim vrhom s, koji će biti korijen najmanjeg razapinjućeg stabla. Svi vrhovi koji *nisu* dio stabla nalaze se u prioritetnom redu Q poredani po *ključu*. Za svaki vrh v, $kljuc(v)$ je najmanja težina w bilo kojeg brida koji povezuje v s nekim vrhom u stablu; ∞ ako takav brid ne postoji. U svakom koraku algoritam uzima iz prioritetnog reda vrh koji je bridom minimalne težine povezan sa stablom i pridružuje taj njegov brid skupu E_T, skupu koji će u konačnici sadržavati bridove najmanjeg razapinjućeg stabla $T = (V, E_T)$. $\pi(v)$ je roditelj vrha v u stablu.

Algoritam 4.1. Primov algoritam

1. $Q \leftarrow V$.
2. $E_T \leftarrow \emptyset$.
3. Za svaki vrh $u \in Q$ postavi $kljuc(u) \leftarrow \infty$.
4. $kljuc(s) \leftarrow 0$.
5. $\pi(s) \leftarrow$ NIŠTA.
6. Dok je $Q \neq \emptyset$ radi slijedeće:
   - 6.1. $u \leftarrow$ uzmi s početka Q.
   - 6.2. Ako je $\pi(u) \neq$ NIŠTA, $E_T \leftarrow E_T \cup \{\{u, \pi(u)\}\}$.
   - 6.3. Za svaki vrh $v \in lista(u)$, ako je $(v \in Q) \wedge (w(u,v) < kljuc(v))$ radi slijedeće:
     - 6.3.1. $\pi(v) \leftarrow u$.

6.3.2. $kljuc(v) \leftarrow w(u, v)$.

Na početku sve vrhove postavljamo u prioritetni red Q i ključeve postavljamo na ∞. Ključ početnog vrha s postavljamo na 0, a $\pi(s)$ na NIŠTA jer je s korijen stabla. Dok god je Q neprazan, u postaje vrh izvan stabla s minimalnom težinom brida kojim je spojen sa stablom te se pridružuje stablu (na način da ga se makne iz Q, a brid kojim je spojen sa stablom se doda u E_T). Svim njegovim susjedima $v \in Q$ mijenja se $kljuc(v)$ i roditelj $\pi(v)$ ako je težina brida kojim su spojeni s u manja od dotadašnjeg $kljuca$, tj. ako je taj vrh v "bliže" stablu u kojem je sad novi vrh u nego što je bio prije.

4.2. Kruskalov algoritam

Kruskalov[2] algoritam radi na bitno drukčiji način od Primovog. Dok smo kod Primovog algoritma stalno povećavali jedno stablo dodavajući nove bridove i vrhove, kod Kruskalovog algoritma krećemo sa šumom u koju u svakom koraku dodajemo brid s najmanjom težinom koji ne stvara ciklus u postojećoj šumi. Na taj način na kraju završavamo s najmanjim razapinjućim stablom.

Algoritam 4.2. Kruskalov algoritam

1. Sortiraj bridove $e_i \in E$ $(i = 1, 2, \ldots, e(G))$ uzlazno po $w(e_i)$.
2. $E_T \leftarrow \emptyset$.
3. $i \leftarrow 0$.
4. Dok je $i < e(G)$ radi slijedeće:
   4.1. $i \leftarrow i + 1$.
   4.2. Ako $E_T \cup \{e_i\}$ nema ciklus postavi $E_T \leftarrow E_T \cup \{e_i\}$.

Na početku algoritam sortira bridove uzlazno prema njihovoj težini. E_T se postavlja na prazan skup, a brojač i na nulu. Algoritam zatim provjerava svih $e(G)$ bridova; ako brid e_i u dotadašnjoj šumi ne stvara cikluse, dodajemo ga u šumu. Šuma u konačnici postaje najmanje razapinjuće stablo $T = (V, E_T)$.

[2]**Joseph Kruskal**, američki matematičar, 1956. je otkrio algoritam za traženje najmanjeg razapinjućeg stabla.

I Kruskalov i Primov algoritam spadaju u skupinu *pohlepnih algoritama (engl. greedy algorithm).* Naime, u svakom koraku biraju ono što je u tom trenutku najbolje. Pohlepni algoritmi su često netočni jer najbolje u određenom trenutku ne mora biti optimalno i na kraju, no kod ova dva algoritma svojstvo 4.1. garantira njihovu točnost.

5. Najkraći put

Problem koji se uz traženje najmanjeg razapinjućeg stabla često javlja u teoriji grafova je problem traženja *najkraćeg puta*: traži se put u težinskom grafu koji povezuje dva vrha x i y tako da zbroj težina bridova na tom putu bude najmanji mogući. Problem je zanimljiv i u netežinskom grafu - traži se put s najmanjim brojem bridova koji povezuje x i y. Na stranici 18. vidjeli smo da se ovakav problem jednostavno rješava pretraživanjem u širinu.

Općenito govoreći, kako bi najkraći put od x do y mogao proći bilo kojim bridom, problem se pretvara u traženje najkraćeg puta od x do svih ostalih vrhova. Uz to, još nije poznat algoritam koji bi riješio problem traženja puta od x do y brže od algoritma za traženje puta od x do svih ostalih vrhova. Najpoznatiji algoritmi za traženje najkraćeg puta su *Dijkstrin algoritam* i *Bellman-Fordov algoritam*. U ovom poglavlju bit će pokazan Dijkstrin algoritam, a za Bellman-Fordov algoritam čitatelj se upućuje da pogleda [2], str. 532.

5.1. Dijkstrin algoritam

Dijkstrin[1] algoritam gotovo je identičan Primovom algoritmu po načinu rada. On održava skup S u kojem se nalaze vrhovi v za koje je najkraća udaljenost $d(v)$ od početnog vrha x već određena. Algoritam u svakom koraku odabire vrh $u \in V \setminus S$ tako da je udaljenost $d(u)$ minimalna (za spremanje vrhova koji

[1] **Edsger W. Dijkstra** (1930.–2002.), nizozemski matematičar, otkrio algoritam za traženje najkraćeg puta u težinskom grafu.

nisu u skupu S koristimo prioritetni red Q organiziran prema ključu $d(v)$) te prema potrebi *poboljšava* udaljenosti svojih susjeda $v \in V \setminus S$. Prethodnik $\pi(v)$ vrha v predstavlja susjedni vrh vrha v koji se u najkraćem putu nalazi bliže početnom vrhu. Ovo nam pomaže kod ispisa puta.

Algoritam 5.1. Dijkstrin algoritam

1. Za svaki vrh $v \in V$ postavi $d(v) \leftarrow \infty$ i $\pi(v) \leftarrow$ NIŠTA.
2. $d(x) \leftarrow 0$.
3. $S \leftarrow \emptyset$.
4. $Q \leftarrow V$.
5. Dok je $Q \neq \emptyset$ radi slijedeće:
   - 5.1. $u \leftarrow$ uzmi s početka Q.
   - 5.2. $S \leftarrow S \cup \{u\}$.
   - 5.3. Za svaki vrh $v \in lista(u)$, ako je $(v \in Q) \wedge (d(v) > d(u) + w(u, v))$, radi slijedeće:
     - 5.3.1. $d(v) \leftarrow d(u) + w(u, v)$.
     - 5.3.2. $\pi(v) \leftarrow u$.

Algoritam prvo radi već uobičajenu inicijalizaciju—postavlja sve udaljenosti na ∞, sve prethodnike na NIŠTA, udaljenost početnog vrha x od samog sebe na 0, S na prazan skup i stavlja sve vrhove u prioritetni red Q (linije 1.–4.). Dok god ima vrhova u prioritetnom redu, algoritam s njegovog početka uzima u za koji je $d(u)$ najmanji i prema potrebi modificira njegove susjede koji su jos u Q tako da im smanji udaljenost od početnog vrha i promijeni prethodnika (petlja pod korakom 5.).

Zbog toga što u svakom koraku uzima vrh s najkraćom udaljenosti, Dijkstrin algoritam također spada u pohlepne algoritme, ali, naravno, stvarno pronalazi najkraći put. (Za dokaz pogledati [2], str. 529.)

Da bismo ispisali pronađeni najkraći put od x do y, poslužit ćemo se informacijama koje smo skupili o prethodnicima svakog vrha na najkraćem putu i slijedećom procedurom:

Algoritam 5.2. Ispis najkraćeg puta

1. Počni proceduru $Ispis(G, x, y)$.
2. Ako je $x = y$, ispiši x
3. inače radi slijedeće:
   3.1. Ako je $\pi(x) =$ NIŠTA ispiši "Ne postoji put od " x "do" y
   3.2. inače $Ispis(G, x, \pi(y))$, ispiši y.

Procedura rekurzivno prolazi putem od y do x pomoću sačuvanih prethodnika π i na povratku ispisuje svaki vrh na putu od x do y.

6. Zaključak

Algoritmi prikazani u ovoj maturalnoj radnji općenito se koriste za rješavanje praktičnih problema u teoriji grafova pa koriste razumne količine resursa (poput računalne memorije i procesorskog vremena). Nažalost, kao što je već spomenuto u odjeljku o Hamiltonovim ciklusima (str. 12.), za neke probleme još nisu otkriveni efikasni algoritmi. Što je još gore, nije nam poznato postoje li uopće efikasni algoritmi za takve probleme.

Spomenuli smo slijedeći problem: "Pronaći najkraći put od vrha x do vrha y." U prethodnom poglavlju vidjeli smo vrlo brz i efikasan algoritam koji nam ovo rješava. No, ako se zapitamo koji je *najdulji* put (bez ciklusa) od x do y, imamo problem za koji nitko ne zna pametnije rješenje od provjere svih mogućih putova. Algoritam za takav problem mogao bi imati eksponencijalno vrijeme izvršavanja.

Općenito, možemo reći da je eksponencijalno vrijeme izvršavanja algoritma proporcionalno s barem 2^N za unos veličine N. To znači da, na primjer, ne možemo garantirati da će algoritam s eksponencijalnim vremenom izvršavanja raditi za unos veličine 100 ili više jer nitko ne može čekati da algoritam napravi 2^{100} koraka. Napredak tehnologije ne znači ništa u usporedbi s eksponencijalnim rastom: superračunalo može biti trilijun puta brže od abakusa, ali ni jedan ni drugi nisu ni blizu rješavanju problema koji ima 2^{100} koraka.

No svejedno, problemi za koje ne postoji efikasno rješenje ne mogu se ignorirati. Uzmimo za primjer problem trgovačkog putnika za čije je rješenje naivnom provjerom svih putova potrebno vrijeme proporcionalno s $N!$. Ovaj

problem se vrlo često pojavljuje. I što da sad trgovački putnik radi kad smo mi nemoćni da riješimo problem za, primjerice, 1000 vrhova? Da se radije prihvati uzgoja krumpira?

Na generacijama koje dolaze ostaje da pokušaju riješiti ove probleme. Kao što smo vidjeli, tehnološki napredak nije toliko značajan. Stvarni napredak čovječanstva ostvaruje se na područjima kao što je matematika.

Ovom maturalnom radnjom dotaknuo sam samo mali dio teorije grafova i algoritama koji se u njoj primjenjuju. Primjerice, nisam pisao o problemima protoka u transportnim mrežama koji se učestalo javljaju. Ovi problemi obuhvaćaju pitanja poput "koju ulicu u gradu treba proširiti da bi se spriječio prometni kolaps" i sl. Također, nisam pisao o problemima sparivanja u bipartitnim grafovima. To su problemi poput slijedećeg: n dječaka i djevojčica sudjeluje u školskom plesu. Svaki dječak želi plesati samo s nekim djevojčicama, a svaka djevojčica želi plesati samo s nekim dječacima. Pronađi najveći broj plesnih parova.

Pisanje o tim temama bi učinilo maturalni rad preopširnim. Čitatelj se upućuje na naslove iz bibliografije ako ga one posebno zanimaju.

Bibliografija

[1] R. Sedgewick: *Algorithms in C*, Addison-Wesley Publishing Company, 1990.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest: *Introduction to Algorithms*, The MIT Press, 1990.

[3] D. Veljan: *Kombinatorna i diskretna matematika*, Algoritam, 2001.