Janez Žerovnik

E-gradivo za predmet PRINCIPI MODELIRANJA V LOGISTIKI

Univerza v Mariboru
Fakulteta za logistiko
Celje 2015

Vsebina

Vsebina . . . . .

1 DEFINICIJE IN PRIMERI **8**
1.1. Osnovni pojmi 8
1.2. Izomorfizem grafov 13
1.3. Sprehodi in razdalje 14
1.4. Primeri grafov 16
1.5. Predstavitve grafov 22
1.6. Primeri uporabe grafov 24
1.7. Naloge z rešitvami 28

2 EULERJEVI IN HAMILTONOVI GRAFI **36**
2.1. Uvod 36
2.2. Eulerjevi grafi 37
2.3. Problemi Eulerjevega tipa 40
2.4. Potrebni in zadostni pogoji za Hamiltonskost grafov 42
2.5. Problemi hamiltonskega tipa 43
2.6. Naloge z rešitvami 46

3 DREVESA **58**
3.1. Karakteristične lastnosti dreves 58
3.2. Centri in centroidi 59
3.3. Pregledovanje grafov 61
3.4. Konstrukcije dreves 64
3.5. Naloge z rešitvami 71

4 BARVANJE GRAFOV **78**
4.1. Barvanja vozlišč 78
4.2. Algoritmi za barvanje vozlišč 80
4.3. Barvanja povezav 83
4.4. Primeri uporabe barvanja 85
4.5. Naloge z rešitvami 89

5 ALGORITMI **95**
5.1. Algoritmi in programski jeziki 96
5.2. Razredi problemov 101
5.3. Primerjava algoritmov 103
5.4. Naloge z rešitvami 106

6 RAČUNANJE NAJKRAJŠIH POTI IN RAZDALJ **112**
6.1. Pregledovanje v širino 113
6.2. Dijkstrov algoritem za iskanje najkrajših poti 116
6.3. Naloge z rešitvami 119

7 LOKALNO ISKANJE **127**
7.1. Splošni problem kombinatorične optimizacije 127

7.2. Naloge z rešitvami 132

PREDGOVOR

Vsebina dela pokriva večino vsebin, ki sem jih v zadnjih letih predaval pri podiplomskem predmetu Matematika - izbrana poglavja na fakulteti za strojništvo. Deloma se vsebina pokriva tudi s programi pri podiplomskih predmetih Operacijske raziskave (FG), Teorija odločanja (FG), Matematika - teorija grafov (FERI), Kombinatorična optimizacija (FERI) in Verjetnostni algoritmi (FERI). Pri predavanjih sem opazil precejšnje pomanjkanje primerne literature, zato sem se odločil za objavo teh zapiskov.

V zadnjih nekaj letih sta bili pri podiplomskem predmetu Matematika - izbrana poglavja najpogosteje izbrani poglavji Teorija grafov in Osnove diskretne optimizacije. Ker študentje ponavadi niso imeli dovolj predznanja iz diskretne matematike, in ker je čas, namenjen predavanjem, omejen, sem se odločil za podrobnejšo obravnavo prvega poglavja (Teorije grafov) in za manj podrobno in bolj projektno usmerjeno drugo poglavje (Osnove diskretne optimizacije). Ta odločitev se delno odraža tudi na tem tekstu, saj so nekatera poglavja iz teorije grafov obdelana precej podrobno, pri poglavjih o diskretni optimizaciji pa se pogosteje zatečem k citiranju virov in manj podrobni obravnavi. Nekaj razdelkov je zelo specialnih, pa ne toliko zaradi pomembnosti vsebine kot zaradi želje, da bi študente seznanil s sorazmerno novimi raziskovalnimi rezultati.

Ker je tekstov s tega področja v slovenskem jeziku sorazmerno malo (meni znane izjeme navajam na koncu predgovora), sem ob zapisovanju poskušal dodati predvsem v obliki opomb različne dodatne informacije, s katerimi želim po eni strani narediti tekst bolj dostopen svojim študentom, po drugi strani pa morda zanimiv tudi za druge bralce.

Tekst je tako opremljen z velikim številom opomb, ki so vsaj dveh vrst. Prve na mnogih mestih, kjer se zaradi omejenega obsega obravnava ustavi, nakažejo pot naprej. V opombah pogosto dodam referenco na tekste v slovenskem jeziku, ki jih poznam. V primerih, ko ne poznam ustreznega teksta v našem jeziku ali utegne biti nadalnje branje sorazmerno zanimivo, (do)dam referenco na tujo literaturo. Druge opombe so namenjene študentom nematematikom, da prehitro ne izgubijo poguma, saj jim osvežijo spomin na osnovne matematične pojme, ki so jih verjetno že obvladali v prvih letih študija.

Namen matematičnih predmetov pri študiju tehniških strok je seznaniti študente z matematičnimi orodji, ki se v njihovi stroki uporabljajo, in obenem predstaviti matematiko kot znanost. Če si sposodim še nekaj misli iz predgovora k učbeniku Matematika II [Tomšič, Mramor-Kosta, Orel, Fakulteta za elektrotehniko in računalništvo, Ljubljana 1995]: Pri predavanjih in pri pisanju učbenika smo ves čas pred dilemo, v kolikšni meri predstaviti matematiko kot orodje, torej kot zbirko „receptov“, ki v stroki lahko koristijo, ali vključiti več matematičnih pojmov in abstraktnih struktur, definicij izrekov in dokazov. Pravega odgovora na to vprašanje ni, saj nekatere abstraktna teorija brez praktičnih primerov zmede, drugim pa pomaga k boljšemu razumevanju, ker z njeno pomočjo laže povežejo različne probleme. Razen tega nekritična uporaba matematike brez globjega in natančnega razumevanja predpostavk skriva cel kup pasti, ki lahko vodijo k napačnim izračunom s čisto praktičnimi posledicami. V tem delu sem se, ker gre v prvi vrsti za podiplomski učbenik, ponekod, predvsem v začetnih poglavjih, večinoma odločil za zelo natančne dokaze trditev in izrekov, drugje, predvsem v kasnejših poglavjih, pa se pogosteje zatečem k navajanju receptov in citiranju literature.

Teorija grafov je eno najhitreje razvijajočih se področij sodobne matematike, verjetno predvsem zaradi mnogih uporab v drugih znanostih in v sodobnih tehnologijah. Ker osnovni pojmi niso del

standardnih učnih programov, je predvsem v prvih poglavjih precejšnje število preprostih nalog s podrobnimi rešitvami. Kljub navidezni preprostosti v teoriji grafov kmalu naletimo na zelo težke, mnogokrat že dolgo odprte raziskovalne probleme. Nekateri so omenjeni tudi v tej knjigi. Uporabnost teorije grafov poudarim tako, da, kjer je mogoče, predstavim kakšno aplikacijo, tudi če je zaradi časa in prostora ne morem razdelati v popolnosti.

Osnovne pojme teorije grafov potrebujemo pri vpeljavi osnovnih pojmov diskretne optimizacije. Za študij diskretne optimizacije je potrebno še precej znanja iz drugih področij. Zato je za obravnavo tega poglavja priporočljivo, da bralec pozna osnove nekaterih poglavij matematike, na primer linearne algebre, pri obravnavi verjetnostnih hevristik pa tudi osnove verjetnostnega računa. Prav tako ne gre brez definicije algoritma in vpeljave formalnega programskega jezika, privzamem pa vsaj malo izkušenj z osnovami programiranja. Potrebno predznanje je torej kar obširno, zato so poglavja pri koncu knjige napisana, tako kot sem nakazal že prej, malo drugače kot v prvem delu. Bralec lahko ta poglavja razume vsaj na dva načina: ali kot osnovno informacijo o delu obširnega področja diskretne optimizacije, ali pa kot prvo branje in motivacijo za kasnejši podrobnejši študij po drugih virih. Tekst je seveda prekratek, da bi bil popoln. Poznavalci področja diskretne optimizacije bodo verjetno pogrešali katero od standardnih vsebin, na primer poglavje o pretokih v omrežjih ali poglavje o osnovah linearnega programiranja. Mislim, da je za bodoče magistre in doktorje tehniških znanosti zelo pomembno, da spoznajo osnove diskretne optimizacije, ki se v času digitalizacije in avtomatizacije pojavlja na vse več področjih, in upam, da jim bo ta učbenik pri tem vsaj malo pomagal.

To delo seveda ni v celoti originalno. Pri pripravi predavanj in teh zapiskov sem se opiral na številne učbenike. Večinoma so navedeni spodaj v seznamu dodatne literature, tu pa naštejem samo tiste, ki sem jih največ uporabljal: prevod Wilson in Watkins: Uvod v teorijo grafov, Kozak: Podatkovne strukture in algoritmi in Biggs: Discrete mathematics.

Rokopis sta strokovno pregledala prof. dr. Aleksander Vesel in doc. dr. Blaž Zmazek, jezikovno pa Jana Žerovnik. Za skrbno branje se vsem najlepše zahvaljujem. Opozorili so na mnoge napake in pomankljivosti, ki sem jih po svojih močeh odpravil. Za preostale se bralcem opravičujem in jih prosim, naj mi odkrite napake sporočijo, da jih bo mogoče popraviti pri morebitni popravljeni izdaji.

Spodnja Senica, Maribor, julij 2003 avtor

Dodatna literatura:

- Najprej nekaj del v slovenščini s področja teorije grafov:
  - D.Bajc, T.Pisanski, Najnujnejše o grafih, DMFA, Ljubljana 1985.
  - V.Batagelj, S.Klavžar, DS2, Algebra in teorija grafov, naloge, DMFA, Ljubljana 1992.
  - V.Batagelj, Diskretne strukture (zapiski predavanj, 4. zvezek, grafi), Samozaložba, Ljubljana 1996.
  - V.Domanjko, Grafi v kombinatoriki, MATH, Ljubljana 1996. (naloge z rešitvami)
  - R.Wilson, M.Watkins, Uvod v teorijo grafov, DMFA, Ljubljana 1997.

  - M.Juvan, P.Potočnik, Teorija grafov in kombinatorika (Primeri in rešene naloge), DMFA, Ljubljana 2000.

- Poglavja o algoritmih dopolnjujeta učbenika:
  - J.Kozak, Podatkovne strukture in algoritmi, DMFA, Ljubljana 1986.
  - B.Robič, Aproksimacijski algoritmi, Založba FRI, Ljubljana 2002.

- Tuja literatura je seveda neprimerno bolj bogata. Tukaj zato navedimo samo
  - poglavje o diskretni optimizaciji v: E.Kreyszig, Advanced Engineering Mathematics, John Wiley & Sons, New York 1993 (7. izdaja).
  - N.Biggs, Discrete Mathematics, Clarendon Press, Oxford 1989.
  - E.Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York 1976.
  - B.Korte, J.Vygen, Combinatorial Optimization, Theory and Algorithms, Springer, Berlin 2000.

    in, morda bolj dostopni,
  - D.Veljan, Kombinatorika s teorijom grafova, Školska knjiga Zagreb 1989.
  - D.Cvetković, V.Kovačević-Vujčić, (ur.), Kombinatorna optimizacija, DOPIS Beograd 1996.

- Precej citatov na članke in knjige je navedeno v opombah. Med navedenimi viri so odlične reference za nadaljnji študij posameznih poglavij.

- Za konec omenimo še dve knjigi s specialnih področij teorije grafov, katerih soavtorja sta slovenska matematika:
  - B.Mohar, C.Thomassen, Graphs on Surfaces, The Johns Hopkins University Press, Baltimore 2001.
  - W.Imrich, S.Klavžar, Graph Products, Structure and Recognition, John Wiley & Sons, New York 2000.

Vsebinsko se druga izdaja ne razlikuje od prve.

Pred novim natisom sem popravil nekaj tiskarskih škratov. Precej sem jih našel s pomočjo študentov, ki se jim ob tej priložnosti za pomoč lepo zahvaljujem. (Najdaljša spiska pripomb sta mi dala Hedvika Mojca Perat in Boštjan Hernavs.) Zagotovo so v tekstu ostale meni skrite napake in pomankljivosti, zato bralce ponovno prosim, naj mi odkrite napake sporočijo, da jih bo mogoče popraviti pri morebitni popravljeni izdaji.

Spodnja Senica, Maribor, januar 2005 avtor

Nova izdaja je namenjena študentom Fakultete za logistiko kot študijsko gradivo pri predmetu Principi modeliranja v logistiki. Izdaja se od prejšnje razlikuje minimalno, popravljene so napake na straneh 13, 56 in 84.

Trnovec, december 2015 avtor

1

DEFINICIJE IN PRIMERI

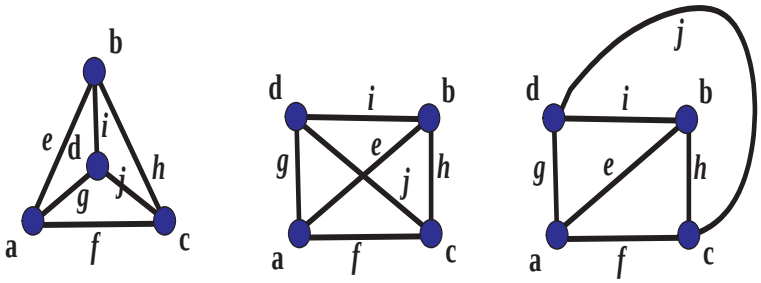
1.1. Osnovni pojmi

Graf $G = (V(G), E(G))$ je določen z množico vozlišč in seznamom povezav. Elemente množice $V(G)$ imenujemo **vozlišča**, elemente seznama $E(G)$ pa **povezave** grafa G. Kadar je jasno, kateri graf obravnavamo, uporabljamo krajši oznaki $V = V(G)$ in $E = E(G)$.[1]

Terminologija v teoriji grafov ni povsem standardna. Za vozlišče grafa je alternativni izraz **točka grafa**, povezave pa včasih imenujemo **robovi**.

Povezavi grafa priredimo vozlišče začetek (prvo krajišče) in vozlišče konec (drugo krajišče) povezave. Če sta obe krajišči povezave isto vozlišče, povezavi rečemo **zanka**. Povezavi, ki imata isti krajišči, sta **vzporedni povezavi**. Graf je **enostaven**, če nima zank in večkratnih povezav.

Iz gornjega sledi, da je graf določen, brž ko poznamo vozlišča in ko vemo, kateri pari vozlišč so povezani s povezavami. Graf lahko narišemo tako, da za vsako vozlišče narišemo krogec, povezave pa prikažemo s črtami, ki povezujejo vozlišča[2]. Pogosto tako dobljeni **risbi grafa** pravimo kar graf. Na sliki 1.1 so tri različne risbe grafa, ki ga lahko popolnoma opišemo z množicama: $V(G) = \{a, b, c, d\}$, $E(G) = \{e, f, g, h, i, j\}$.



Slika 1.1: Tri risbe istega grafa.

Posebno za majhne grafe je risba grafa zelo uporabna. Kadar opazujemo strukturo grafa pogosto oznake vozlišč niso pomembne, zato v takem primeru graf narišemo brez oznak. Ponavadi to pomeni, da si lahko oznake poljubno izberemo.

V nekaterih aplikacijah je pri povezavi pomembno, kako je usmerjena. Takrat si povezavo predstav-

[1] **Seznam** je posplošitev pojma množice. Množica je seznam, v katerem lahko element nastopa največ enkrat. V nadaljevanju bodo seznami pogosto kar običajne množice. V nekaterih primerih, na primer ko imamo grafe z večkratnimi povezavami, pa nam pride prav, da lahko isti element nastopa v seznamu več kot enkrat.

[2] Odtod poimenovanje *grafi* za abstraktne kombinatorične objekte.

ljamo kot usmerjeno povezavo ali puščico. Grafe z usmerjenimi povezavami imenujemo **usmerjeni grafi** ali **digrafi**. Če nas usmerjenost povezav ne zanima, govorimo enostavno o množici krajišč povezave (ki ima dva elementa ali enega, če je povezava zanka). Tako dobimo **neusmerjene grafe**.

Strogo formalne definicije grafa v jeziku teorije množic niso preveč zahtevne. V različnih virih se razlikujejo predvsem zaradi tega, ker so ene bolj primerne za neusmerjene, spet druge za usmerjene grafe. Podobno je definicija, ki jo izberemo, če obravnavamo samo enostavne grafe, preprostejša od definicije za neenostavne grafe[3].

Če se omejimo na enostavne neusmerjene grafe, lahko definiramo:

Enostaven graf je urejen par množic $G = (V(G), E(G))$, kjer je $V(G)$ poljubna množica vozlišč, $E(G)$ pa množica parov vozlišč, ki jih imenujemo povezave grafa G.[4]

Če v neusmerjenem grafu ni večkratnih povezav, potem zaradi enostavnosti povezavo grafa e s krajiščema u in v označimo kar z $e = uv = vu$.[5]

Usmerjene grafe lahko definiramo takole:

Usmerjen graf ali **digraf** je urejen par množic $G = (V(G), E(G))$, kjer je $V(G)$ poljubna množica vozlišč, $E(G)$ pa množica urejenih parov vozlišč, ki jih imenujemo (usmerjene) povezave grafa G. Če velja, da v $E(G)$ ni urejenih parov oblike (v, v), dobimo **enostaven usmerjen graf** ali **enostaven digraf.**[6]

Vpeljimo še nekaj besednih zvez za kasnejšo uporabo. Krajišči povezave sta **povezani** s povezavo. To zapišemo z $uv \in E(G)$ ali z $u \sim_G v$. Če je jasno, za kateri graf gre, zapišemo krajše $uv \in E$ ali $u \sim v$. Če sta vozlišči grafa v in w povezani s povezavo, potem rečemo, da sta **sosednji**. Povezava $e = uv$ **gre skozi** vozlišči u in v (je **incidentna** z vozliščema u in v, je **sosednja** vozliščema u in v). Krajišči povezave e, vozlišči u in v, sta **incidentni** z e (**ležita na** povezavi e).

Število vozlišč grafa običajno označimo z $n = |V(G)|$. Standardna oznaka za število povezav je $m = |E(G)|$. Elemente množice vozlišč pogosto označimo z naravnimi števili 1,2,3,..., torej $V(G) = \{1, 2, \ldots, n\}$. (Včasih je primernejša izbira oznak 0,1,2,..., torej $V(G) = \{0, 1, 2, \ldots, n-1\}$.)

Utežen graf (ali **omrežje**) je graf $G = (V(G), E(G))$ z dano preslikavo $\lambda : E(G) \to \mathbb{R}$, ki vsaki povezavi grafa priredi utež. Utežem na povezavah pogosto lahko rečemo dolžine ali cene

[3] Sorazmerno preprosta definicija, ki zajema tako usmerjene kot neusmerjene grafe je naslednja.
Digraf (usmerjen graf): V... poljubna množica vozlišč. E... množica povezav; vsaki povezavi e sta prirejeni dve vozlišči: ini(e) in ter(e), (začetno in končno krajišče).
Potem velja: e zanka $\iff$ int(e)=ext(e); e in f vzporedni $\iff$ ini(e)=ini(f) in ter(e)=ter(f); e in f nasprotni: ini(e)=ter(f) in ter(e)=ini(f).
Neusmerjen graf: v gornji definiciji ne ločimo med ini(e) in ter(e), definiramo samo množico krajišč: $e \mapsto$ ext= {ini(e),ter(e)}. Povezavi sta vzporedni, če je $ext(e) = ext(f)$. Povezava je zanka, če je $|ext(e)| = 1$.
Definicija enostavnih grafov je še preprostejša. Enostavni digrafi: V je poljubna množica vozlišč, množica povezav pa je (poljubna) podmnožica $E \subseteq (V \times V) \setminus \{(v, v) \mid v \in V\}$. ($\times$ je oznaka za kartezični produkt množic, $\setminus$ pa za razliko množic). Enostavne neusmerjene grafe lahko dobimo tako, da vzamemo $E \subseteq$množice parov vozlišč.

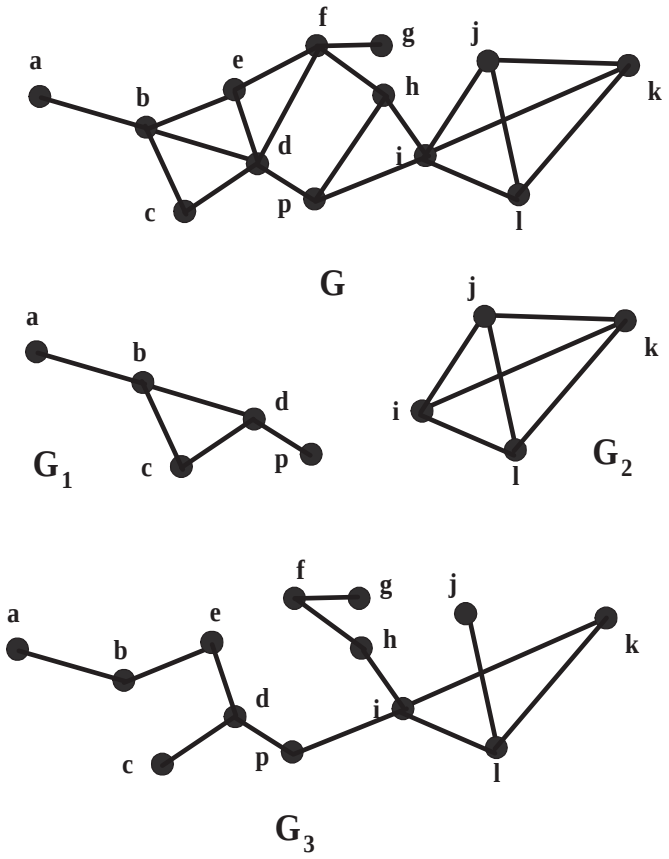
[4] Iz dogovora v teoriji množic, po katerem so vsi elementi množice različni, sledi, da ima povezava dve različni krajišči. Torej zank ni. Ker je povezava določena s krajiščema, tudi večkratne povezave po tej definiciji niso mogoče.

[5] Včasih tudi z $\{u, v\}$. Tak dogovor seveda ni smiselen za usmerjene grafe.

[6] Par je množica z dvema *različnima* elementoma. Urejen par ima prvi in drugi element, ki sta lahko tudi enaka. Formalno je urejen par element kartezičnega produkta dveh množic.

povezav. V nekaterih aplikacijah utežimo vozlišča grafa, takrat graf $G = (V(G), E(G))$ opremimo s preslikavo $w : V(G) \to \mathbb{R}$, ki vsakemu vozlišču priredi utež. Seveda lahko hkrati definiramo uteži na povezavah in na vozliščih.

Tako v matematiki kot v tehniki pogosto proučujemo zapletene objekte tako, da gledamo enostavnejše objekte istega tipa, ki jih ti vsebujejo, manjši objekti pa pogosto dobijo ime s predpono „pod-“. Proučujemo na primer podmnožice množic, podsisteme sistemov, podgrupe grup itd.



Slika 1.2: Graf G s podgrafi G_1, G_2 in G_3.

Naj bo G graf z množico vozlišč $V(G)$ in množico povezav $E(G)$ in G' graf z množico vozlišč $V(G')$ in množico povezav $E(G')$. Če je $V(G')$ podmnožica množice $V(G)$ in če je vsaka povezava iz $E(G')$ tudi v $E(G)$, potem je G' **podgraf** grafa G.

Graf G' je **vpet podgraf** grafa G, če je $V(G') = V(G)$. Graf G' je **induciran** (ali **porojen**) **podgraf** (na množici vozlišč $V(G')$), če za poljuben par vozlišč $u, v \in E(G')$ velja: $uv \in E(G) \Rightarrow uv \in E(G')$. Graf G' je **induciran podgraf** (na množici povezav $E(G')$), če je induciran na množici krajišč $E(G')$.

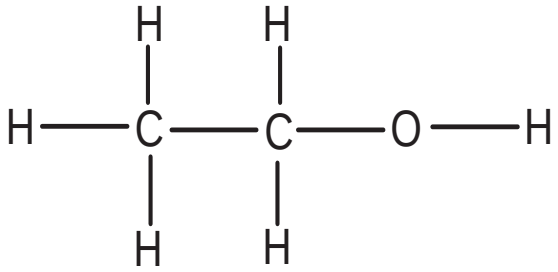
Na sliki 1.2 je G_1 podgraf grafa G, *induciran na vozliščih* $\{a, b, c, d, p\}$, graf G_2 je podgraf *induciran na povezavah* $\{ij,ik,il,jk,jl,kl\}$. G_3 je *vpet* podgraf grafa G.

Naj bo G poljuben graf in $S \subseteq V(G)$ podmnožica množice vozlišč grafa. Potem z $G \setminus S$ označimo graf induciran na množici vozlišč $V(G) \setminus S$. V posebnih primerih, ko je množica S samo eno vozlišče v ali samo krajišči ene povezave e, uporabimo oznaki $G \setminus v$ in $G \setminus e$.

Naj bo G graf brez zank, v pa vozlišče grafa G. **Stopnja** vozlišča v je število povezav, ki vsebujejo v. Stopnjo vozlišča označimo s $\mathrm{st}(v)$[7]. Največjo stopnjo vozlišča grafa G označimo z $\Delta(G)$, najmanjšo pa z $\delta(G)$. Za vsako vozlišče $v \in V(G)$ torej velja

$$\delta(G) \leq \mathrm{st}(v) \leq \Delta(G).$$

V kemiji uporabljajo izraz *valenca* za število kemijskih vezi, ki povezujejo atom s sosednjimi atomi. Na diagramu, ki predstavlja molekulo etanola, ima na primer vsak atom ogljika valenco 4, kisikov atom ima valenco 2, vsak vodikov atom pa ima valenco 1.[8]



Slika 1.3: Molekula etanola.

Soseščina vozlišča v, $N_G(v)$, je množica vseh vozlišč, ki so sosednja z vozliščem v, ali krajše

$$N_G(v) = \{u \mid u \in V(G), uv \in E(G)\}.$$

Če je jasno, za kateri graf gre, uporabljamo krajšo oznako $N(v) = N_G(v)$.

Graf (a) na sliki 1.4 ima stopnje vozlišč:

$$\mathrm{st}(u) = 2,\ \mathrm{st}(v) = 3,\ \mathrm{st}(w) = 4,\ \mathrm{st}(z) = 1.$$

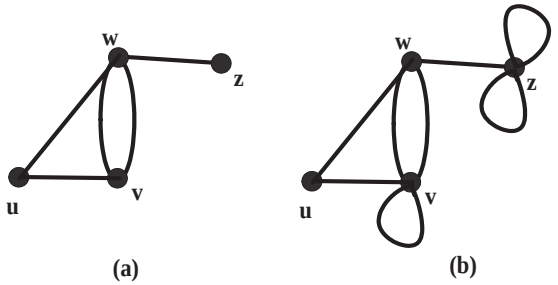
Stopnja vozlišča je bila doslej definirana samo za grafe brez zank. Definicijo lahko preprosto razširimo na grafe z zankami tako, da se dogovorimo, da naj vsaka zanka prispeva 2 k stopnji vozlišča. Graf (b) na sliki 1.4 ima potem stopnje vozlišč:

$$\mathrm{st}(u) = 2,\ \mathrm{st}(v) = 5,\ \mathrm{st}(w) = 4,\ \mathrm{st}(z) = 5.$$

Pogosto je prikladno zapisati stopnje vozlišč grafa; pri tem jih običajno zapišemo v nepadajočem vrstnem redu (to je, v naraščajočem vrstnem redu, vendar dovolimo ponovitve, kjer je potrebno). Dobljeni seznam imenujemo **zaporedje stopenj** (vozlišč) grafa. Na primer: zaporedje stopenj grafa (a) je (1,2,3,4) in zaporedje stopenj grafa (b) je (2,4,5,5).

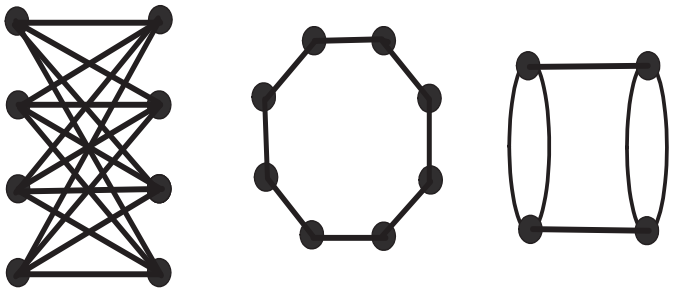
[7] Pogosto uporabljane oznake za stopnjo vozlišča v so tudi $\mathbf{deg}(v)$ in $d_G(v)$ ali $d(v)$.

[8] Predstavitev molekul na ta način je leta 1864 uporabil Alexander Crum Brown (1838-1922) in z njim pojasnil pojav izomerizma. Glej tudi opombo na strani 26.



Slika 1.4: Stopnje vozlišč.

Rečemo, da je graf **regularen**, če imajo vsa vozlišča grafa enako stopnjo. Če imajo vsa vozlišča grafa stopnjo r, rečemo, da je graf **regularen stopnje** r ali r-**regularen**. Na naslednji sliki je nekaj primerov regularnih grafov različnih stopenj:



Slika 1.5: Regularni grafi.

Prvi graf na sliki 1.5 je regularen stopnje 4 in ima osem vozlišč, torej je vsota vseh stopenj vozlišč grafa enaka 32. Vidimo tudi, da ima ta graf 16 povezav. Torej vsota stopenj vozlišč je natanko dvakrat število povezav. Izkaže se, da to velja za vse grafe in rezultat včasih imenujemo *lema o rokovanju*.[9]

IZREK 1.1. (LEMA O ROKOVANJU.) *V vsakem grafu je vsota stopenj vozlišč enaka številu povezav, pomnoženem z 2.*

Lema o rokovanju velja tudi za grafe z zankami, saj tudi vsaka zanka prispeva k stopnji pripadajočega vozlišča natanko 2.

Poimenovanje lema o rokovanju izvira iz dejstva, da lahko z grafom predstavimo skupino ljudi, ki se rokuje na zabavi. V takem grafu so ljudje predstavljeni z vozlišči, povezavo med človekoma pa dodamo, če sta se ta na zabavi rokovala. V tej interpretaciji je število povezav enako številu vseh rokovanj, stopnja vozlišča je število rokovanj osebe, ki jo predstavlja vozlišče, vsota stopenj pa je število rok, ki so sodelovale v rokovanjih. Lema o rokovanju torej pravi, da je število vseh rok, ki so se rokovale, dvakrat večje od števila rokovanj – seveda preprosto zato, ker v vsakem rokovanju

[9]Sklepamo takole: Ker ima vsaka povezava dve krajišči, prispeva k vsoti stopenj grafa natanko 2. Rezultat sledi.

sodelujeta natanko dve roki.

Lema o rokovanju ima nekaj pomembnih posledic (Glej tudi nalogo 1.7.).

Posledice leme o rokovanju

1. V vsakem grafu je vsota vseh stopenj vozlišč grafa sodo število.
2. V vsakem grafu je število vozlišč lihe stopnje sodo.
3. Če ima graf G n vozlišč in je regularen stopnje r, ima G natanko $\frac{1}{2}nr$ povezav.

1.2. Izomorfizem grafov

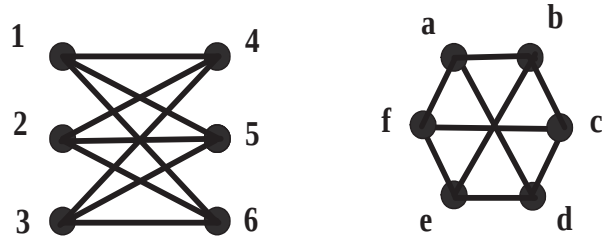
Dva grafa G in H sta **izomorfna**, $G \approx H$, če obstaja bijektivna preslikava $\varphi : V(G) \to V(H)$ med množicama vozlišč, za katero velja:

$$u \sim v \iff \varphi(u) \sim \varphi(v)$$

za poljubni vozlišči u, $v \in V(G)$. Preslikavi φ rečemo **izomorfizem**.

Iz definicije sledi, da sta grafa G in H izomorfna, če lahko H dobimo iz G tako, da spremenimo oznake vozlišč – torej, če obstaja povratno enolična preslikava[10] med vozlišči G in vozlišči H, tako da je število povezav, ki povezujejo katerikoli par vozlišč v G, enako številu povezav, ki povezujejo pripadajoči par vozlišč v H.

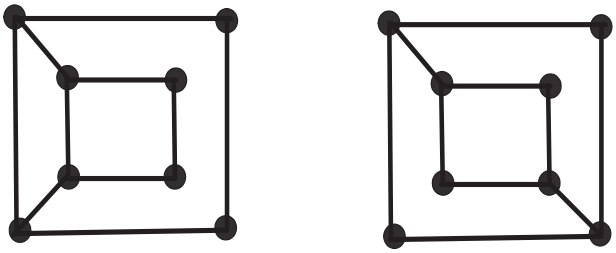
Grafa na sliki 1.6 sta izomorfna, ker je preslikava, definirana z: $1 \mapsto a$, $2 \mapsto c$, $3 \mapsto e$, $4 \mapsto b$, $5 \mapsto f$, $6 \mapsto d$, izomorfizem. (Ker je vozlišč malo, lahko to enostavno preverimo s pregledom vseh parov vozlišč in njihovih slik.)



Slika 1.6: Izomorfna grafa.

Grafa na sliki 1.7 nista izomorfna. Kako to dokažemo? Vidimo, da sta v prvem grafu dve sosednji vozlišči stopnje 2. V drugem grafu takega para vozlišč ni. Ni težko videti, da izomorfizem ohranja stopnjo vozlišča. Sosednja vozlišča se preslikajo v sosednja (Podrobnosti glej v nalogi 1.8.).

[10] Preslikava $f : A \mapsto B$ je **povratno enolična** (ali bijektivna ali "1-1"), če je **injektivna**, torej če vsakemu elementu $x \in A$ priredi natanko en element $f(x) \in B$, in **surjektivna**, torej če je vsak element $y \in B$ slika nekega elementa iz A.



Slika 1.7: Neizomorfna grafa.

Našli smo torej lastnost, ki se pri izomorfizmu ohranja in jo eden od grafov ima, drugi pa ne. Grafa torej nista izomorfna.

Invarianta grafa ali **stalnica** je lastnost grafa, ki se ohranja pri izomorfizmu. Invariante grafa, ki smo jih že srečali, so: število vozlišč, število povezav, zaporedje stopenj. Kasneje bomo srečali še vrsto lastnosti, ki so invariante: premer grafa, biti Eulerjev, biti Hamiltonov, itd.

Pogosto grafe uporabljamo v okoliščinah, ko je pomembna struktura, oznake vozlišč pa ne. Zato izomorfnih grafov ne ločimo.[11] Lahko rečemo, da obravnavamo neoznačene grafe.[12]

1.3. Sprehodi in razdalje

Sprehod med vozliščema v_0 in v_k v grafu G je zaporedje vozlišč in povezav grafa G oblike

$$v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$$

kjer so $v_0, v_1, v_2, \ldots, v_k$ vozlišča in $e_1, e_2, \ldots, e_k$ povezave grafa in $e_i = v_{i-1}v_i$ za $i = 1, 2, \ldots, k$. **Dolžina sprehoda** je število povezav sprehoda.[13]

V posebnih primerih lahko sprehod na krajši način opišemo kot zaporedje vozlišč ali zaporedje povezav. Tako na primer lahko grafu brez vzporednih povezav v opisu sprehoda brez škode izpustimo povezave, saj je povezava natanko določena s svojima krajiščema.[14]

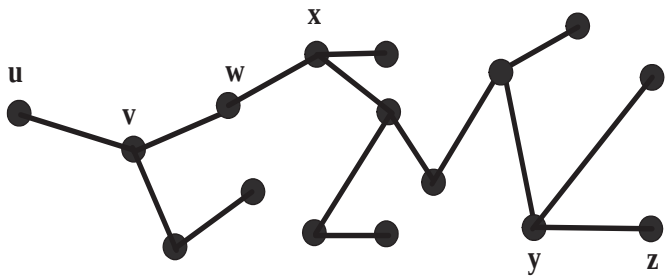
V definiciji sprehoda je „drugo vozlišče“ povezave vedno isto kot „prvo vozlišče“ naslednje povezave. Intuitivno si sprehod na sliki 1.8 zares lahko predstavljamo kot sprehod od u do v, potem do w, potem do x, in tako naprej, dokler nazadnje ne končamo v vozlišču z. Ker povezave niso usmerjene, lahko tak sprehod razumemo tudi kot potovanje od z nazaj proti y, in tako dalje do vozlišč x, w, v in nazadnje do u. Isti sprehod bi lahko torej označili tudi z $zy \ldots xwvu$ in mu

[11] Objekti teorije so potem ekvivalenčni razredi grafov glede na relacijo izomorfizma grafov. Zato pogosto namesto $H \approx G$ zapišemo kar $H = G$.

[12] Oznake seveda lahko uporabljamo, vendar jih lahko poljubno izberemo. Pri obravnavi označenih grafov so oznake vozlišč določene vnaprej.

[13] Če so povezave grafa utežene, potem je smiselno definirati dolžino sprehoda kot vsoto uteži povezav sprehoda.

[14] V zapisu sprehodov bomo vejice ponavadi izpustili, če bo zapis brez njih nedvoumen.



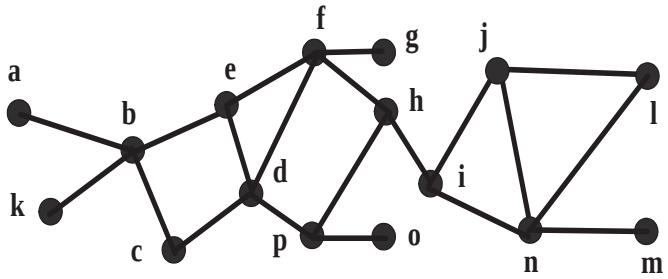
Slika 1.8: Sprehodi na grafu.

rekli *sprehod med z in u*.

V definiciji sprehoda nismo zahtevali, da bi bile vse povezave ali vozlišča različna.

Če so vse povezave sprehoda različne, potem sprehod poimenujemo **enostaven sprehod** ali **sled**. Če so v enostavnem sprehodu vsa vozlišča različna, potem sprehodu rečemo **pot**.

Posebej poimenujemo tiste sprehode in poti, ki se začnejo in končajo v istem vozlišču. Sprehod $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$ imenujemo **sklenjen sprehod** ali **obhod**, če velja $v_0 = v_k$. Če so vse povezave obhoda različne, potem ga poimenujemo **enostaven obhod** ali **sklenjena sled**. Če so v obhodu vse povezave in vsa vozlišča (razen $v_0 = v_k$) različna, potem ga imenujemo **cikel**.



Slika 1.9: Sprehodi na grafu.

V grafu na sliki 1.9 je obhod $b\,c\,d\,p\,h\,f\,d\,e\,b$ *sklenjena sled*, ki ni cikel (saj se vozlišče d pojavi dvakrat), medtem ko so sklenjene sledi $b\,c\,d\,p\,h\,f\,e\,b$, $i\,j\,l\,n\,i$ in $b\,c\,d\,e\,b$ tudi *cikli*. Cikel dolžine tri, kot sta na primer $d\,e\,f\,d$ ali $n\,i\,j\,n$, imenujemo **trikotnik**. Pri opisovanju sklenjenih sprehodov lahko uporabimo za začetek katerokoli vozlišče sprehoda. Prav tako se dogovorimo, da prvega (ki je hkrati tudi zadnje vozlišče) ne bomo zapisali dvakrat. Tako na primer trikotnik $d\,e\,f\,d$ lahko enakovredno zapišemo z $d\,e\,f$, $e\,f\,d$ ali $f\,d\,e$.

Zdaj lahko uporabimo definicijo poti za definicijo *povezanega grafa*.

Graf G je **povezan**, če obstaja pot med poljubnim parom vozlišč, sicer je **nepovezan**. Nepovezan graf razpade na nekaj povezanih podgrafov, ki jih imenujemo (povezane) **komponente** grafa (ali komponente povezanosti grafa). **Most** je povezava v povezanem grafu, brez katere bi bil graf

nepovezan. Z drugimi besedami, povezava e povezanega grafa G je most, če je graf $G \setminus e$ nepovezan.

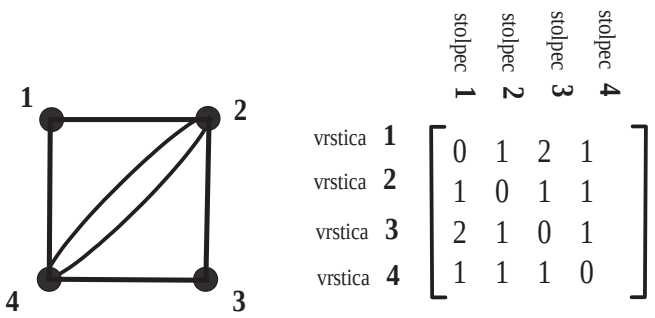
Razdalja $d_G(u, v)$ med vozliščema u in v v grafu G je dolžina najkrajšega sprehoda[15] med njima. Če med vozliščema ni nobenega sprehoda, potem zapišemo

$$d_G(u, v) = \infty\,.$$

Graf G je povezan natanko tedaj, ko za vsak par vozlišč $u, v \in V(G)$ velja $d_G(u, v) < \infty$. Če je jasno, na katerem grafu gledamo razdaljo, potem uporabimo oznako $d(u, v)$.[16]

Matriko $D(G)$ velikosti $n \times n$, v kateri element v i-ti vrstici in j-tem stolpcu pove razdaljo med vozliščema i in j v grafu G, imenujemo **matrika razdalj** grafa G (glej sliko 1.10). Hitro vidimo, da so diagonalni elementi v matriki razdalj enaki 0. V neusmerjenem grafu je matrika razdalj simetrična[17].

Računanje razdalj v grafu bomo podrobneje obravnavali kasneje.



Slika 1.10: Graf in matrika razdalj $D(G)$.

1.4. Primeri grafov

Vpeljali bomo nekaj pomembnih vrst grafov, ki jih bomo pogosto uporabljali v nadaljevanju. Nekaterih se bomo kasneje lotili bolj podrobno, drugi pa nam bodo služili kot primeri, na katerih bomo predstavili nove pojme.

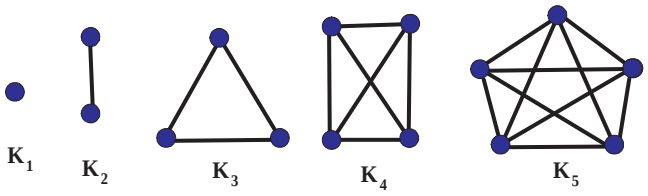
Polni grafi

Poln graf je graf, v katerem je vsak par različnih vozlišč povezan z natanko eno povezavo. Poln graf na n vozliščih označimo s K_n. Razen grafa K_4 običajno narišemo polne grafe v obliki pravilnega mnogokotnika (Slika 1.11).

[15] Ker je najkrajši sprehod vedno pot (o tem se ni težko prepričati), bi lahko definirali razdaljo tudi kot dolžino najkrajše poti med vozliščema. Formalno je pogosto enostavneje uporabiti definicijo s sprehodi.

[16] Tako definirana razdalja je **metrika** (v topološkem smislu), saj velja: (a) $d_G(u, v) \geq 0$ in ($d_G(u, v) = 0 \iff u = v$), (b) $d_G(u, v) \geq d_G(v, u)$ in (c) $d_G(u, v) + d_G(v, w) \leq d_G(u, w)$.

[17] Simetrična matrika ima lastnost $a_{ij} = a_{ji}$ za vse i in j.



Slika 1.11: Polni grafi.

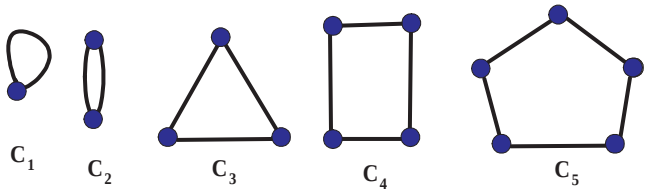
Graf K_n je regularen stopnje $n-1$, zato ima po tretji posledici leme o rokovanju $\frac{1}{2}n(n-1)$ povezav.

Prazni grafi

Prazen graf je graf brez povezav. Prazen graf na n vozliščih označimo z N_n. N_n je regularen stopnje 0.

Cikli

Cikel je graf, ki ga sestavlja en sam cikel. Graf označimo s C_n.



Slika 1.12: Cikli.

Graf C_n je regularen stopnje 2 in ima n povezav. Vozlišča C_n običajno označimo z $V(C_n) = \{0, 1, 2, \ldots, n-1\}$. Povežemo zaporedna vozlišča in $n-1$ z 0. Formalno to lahko zapišemo takole: $i \sim j \iff j = i+1 \pmod{n}$.[18]

Poti

Pot je graf, ki ga sestavlja ena sama pot. Pot na n vozliščih označimo s P_n.

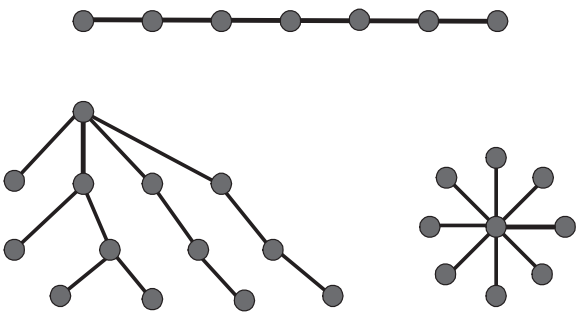
Graf P_n ima $n-1$ povezav in ga lahko dobimo iz cikla C_n z odstranitvijo katerekoli povezave.

Drevesa

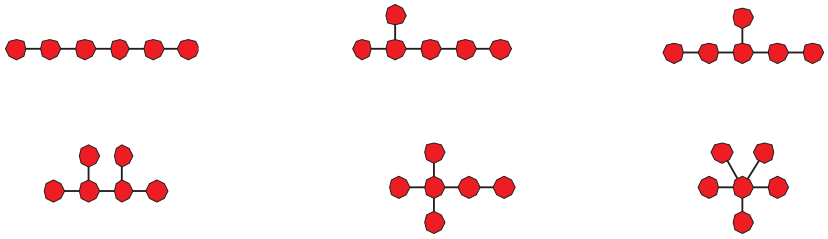
Povezan graf brez ciklov imenujemo **drevo**. Nekaj primerov dreves je narisanih na sliki 1.13. Na

[18] Zapis $k = \ell \pmod{n}$ preberemo „k je enako ℓ po modulu n“ in pomeni, da sta ostanka celih števil k in ℓ pri deljenju z n enaka.

sliki 1.14 so vsa različna drevesa na šestih vozliščih. O drevesih bomo povedali več v poglavju 3.



Slika 1.13: Nekaj dreves



Slika 1.14: Vsa neizomorfna drevesa na šestih vozliščih.

Naj bo G poljuben povezan graf. Podgraf grafa G, ki vsebuje vsa vozlišča grafa G in je drevo, imenujemo **vpeto drevo** grafa G.

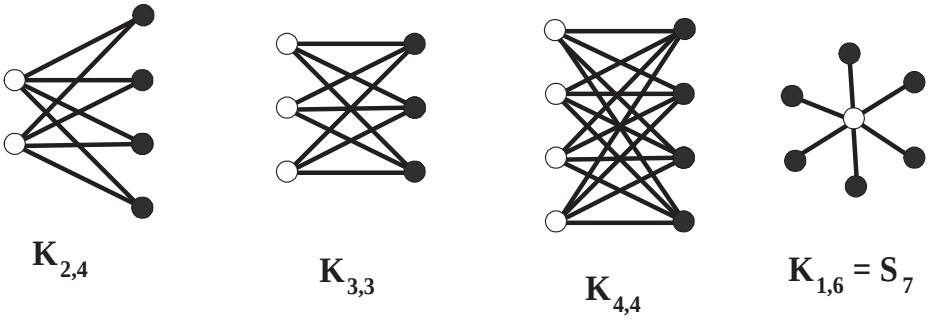
Dvodelni grafi

V praksi so zelo pomembni dvodelni grafi. **Dvodelen graf** je graf, pri katerem lahko množico vozlišč razbijemo[19] na podmnožici A in B, tako da vsaka povezava grafa G povezuje po eno vozlišče iz podmnožice A z enim vozliščem iz podmnožice B. Vozlišča podmnožic A in B lahko razločimo na primer tako, da pobarvamo prva s črno, druga pa z belo barvo. Potem vsaka povezava povezuje po eno črno in eno belo vozlišče.

Med grafi, ki smo jih že srečali, je kar nekaj primerov dvodelnih grafov. Med polnimi grafi je dvodelen samo K_2, ničelni grafi so trivialno dvodelni. Sodi cikli so dvodelni, lihi cikli pa niso dvodelni grafi. Vse poti in vsa drevesa so dvodelni grafi.

Poln dvodelen graf je dvodelen graf, v katerem je vsako črno vozlišče povezano z vsakim belim vozliščem z natanko eno povezavo. Polni dvodelni graf z r črnimi vozlišči in z s belimi vozlišči označimo s $K_{r,s}$. Poln dvodelen graf oblike $K_{1,s}$ imenujemo **zvezda**.[20] Nekaj primerov polnih dvodelnih grafov je na sliki 1.15.

[19] Množici A in B sta **razbitje** množice V, če velja: $A \cup B = V$ in $A \cap B = \emptyset$.
[20] Zvezdo označimo tudi z S_n, torej velja $K_{1,s} = S_{s+1}$.

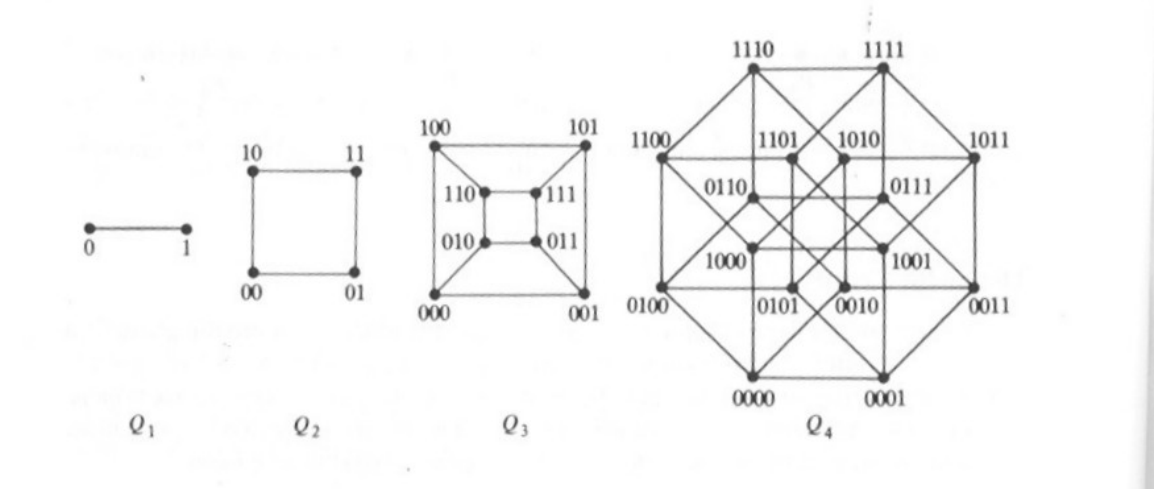


Slika 1.15: Polni dvodelni grafi.

Graf $K_{r,s}$ ima $r + s$ vozlišč (r vozlišč stopnje s in s vozlišč stopnje r) in rs povezav. Vedno velja $K_{r,s} = K_{s,r}$; običajno zapišemo najprej manjše od števil r in s.

Hiperkocke

Med dvodelnimi grafi so posebno zanimivi grafi hiperkocke. Ti grafi so pomembni v teoriji kodiranja. Lahko jih konstruiramo tako, da vzamemo za vozlišča vse besede nad abecedo $\{0, 1\}$ (zaporedja ničel in enic) dane dolžine in povežemo dve besedi, če se besedi razlikujeta na natanko enem mestu. Tako dobljeni graf iz besed dolžine k imenujemo k-**hiperkocka** ali k-**dimenzionalna kocka** in ga označimo s Q_k. (Glej tudi nalogi 1.13. in 1.14.)



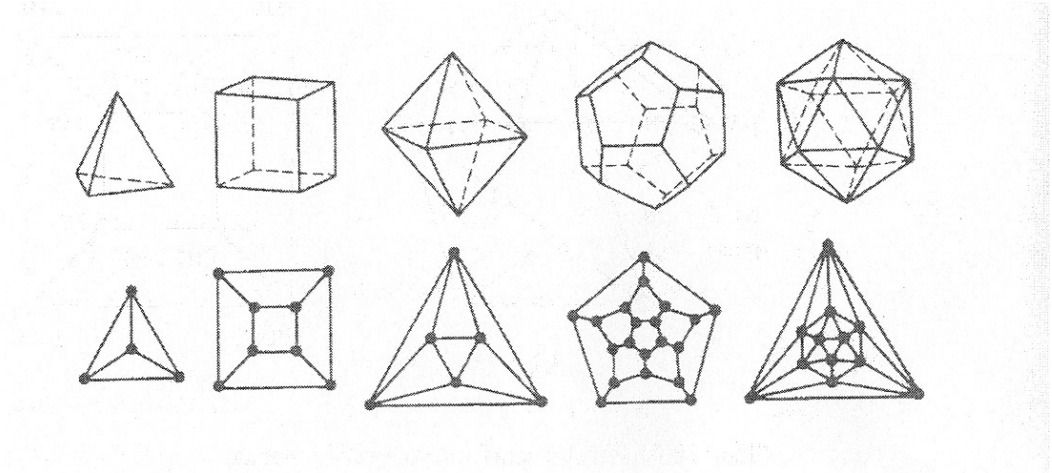
Slika 1.16: Hiperkocke.

Graf Q_k ima 2^k vozlišč in je regularen stopnje k. Z uporabo tretje posledice leme o rokovanju izračunamo, da je število povezav enako $k \cdot 2^{k-1}$.

Grafi pravilnih teles (platonski grafi)

Platonska telesa[21] so: tetraeder, kocka, oktaeder, dodekaeder in ikozaeder.

Oglišča teles lahko uporabimo za vozlišča, robove pa za povezave grafa in dobimo **grafe pravilnih teles** ali **platonske grafe**, ki jih običajno narišemo tako, kot smo naredili na sliki 1.17.



Slika 1.17: Graf tetraedra, kocke, oktaedra, dodekaedra in ikozaedra.

Platonski graf dobimo s projekcijo telesa na ravnino. Na drug način lahko podobno sliko dobimo, če gledamo žičnat model platonskega telesa iz vozlišča v bližini sredine enega od lic.

Petersenov graf

Petersenov graf lahko narišemo na različne načine, dva od njih sta na sliki 1.18.

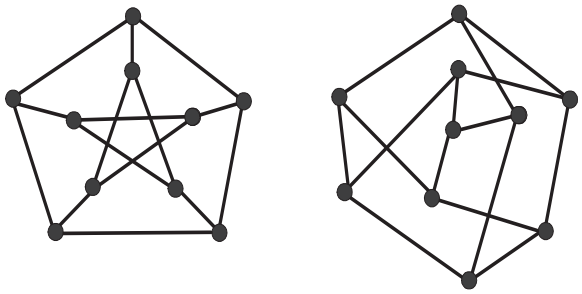
Petersenov graf ima nekaj zanimivih lastnosti in pogosto ga najdemo med protiprimeri za kakšno domnevo[22].

Unije in komplementi

Obstaja kar nekaj operacij, ki jih lahko delamo na grafih in tako naredimo nove grafe. Najenostavnejša operacija je **unija** grafov, ki jo dobimo tako, da naredimo graf, katerega komponente so posamezni grafi.

[21] Ime *platonski* izvira iz omembe teh pravilnih teles v Platonovem delu *Timaeus*. Obstaja natanko pet regularnih komveksnih poliedrov, in to so ravno platonska telesa. Presenetljivo preprost je dokaz te trditve s pomočjo teorije grafov [R.Wilson, M.Watkins, Uvod v teorijo grafov, DMFA Ljubljana 1997, stran 262].

[22] Julius Petersen (1839-1910) je bil danski matematik, ki je ta graf obravnaval v članku leta 1898.



Slika 1.18: Petersenov graf.

Ničelni graf N_n je na primer unija n ničelnih grafov N_1.

Če je G enostaven graf, naredimo **komplement** $\bar{G}$ tako, da uporabimo množico vozlišč grafa G in povežemo vozlišči $u \neq v$ natanko tedaj, ko nista povezani v grafu G. Na primer, komplement polnega grafa K_n je ničelni graf N_n in komplement grafa $K_{4,4}$ je unija dveh K_4. Če naredimo komplement grafa $\bar{G}$, dobimo nazaj originalni graf G.

Produkti grafov

Kartezični produkt grafov $G = (V(G), E(G))$ in $H = (V(H), E(H))$ je graf $G\Box H$, katerega množica vozlišč je kartezični produkt množic vozlišč faktorjev, $V(G\Box H) = V(G)\times V(H)$. Povezave so določene takole: označimo z $x_1, y_1 \in V(G)$ vozlišči faktorja G in z $x_2, y_2 \in V(H)$ vozlišči grafa H. Vozlišči (x_1, x_2) in (y_1, y_2) grafa $G\Box H$ sta povezani s povezavo natanko tedaj, ko sta vozlišči v enem od faktorjev povezani, v drugem pa enaki. (Drugače zapisano: $(x_1, x_2) \sim (y_1, y_2) \iff x_1 = y_1$ in $x_2 \sim_H y_2$ ali $x_2 = y_2$ in $x_1 \sim_G y_1$.)

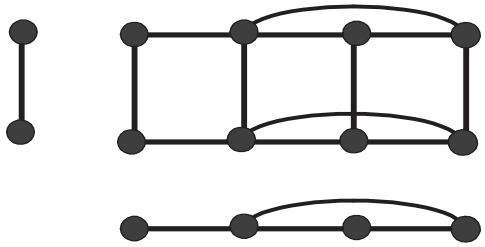
Ni težko videti, da je produkt $H\Box G$ izomorfen produktu $G\Box H$ in da sta izomorfna produkta $(G\Box H)\Box K$ in $G\Box(H\Box K)$. Zato se dogovorimo, da so vozlišča produkta k faktorjev $G_1, G_2, \ldots, G_k$ k-terice $v = (v_1, v_2, \ldots, v_k)$, kjer je $v_i \in G_i$. Vozlišči $x = (x_1, x_2, \ldots, x_k)$ in $y = (y_1, y_2, \ldots, y_k)$ sta povezani natanko tedaj, ko obstaja i, tako da je $x_i \sim_{G_i} y_i$ in $x_j = y_j$ za vse $j \neq i$.[23] Kartezični produkt k faktorjev G_1, G_2, …, G_k, označimo z $\Box_{i=1}^{k} G_i$.

Na naraven način je mogoče definirati še nekatere druge produkte grafov, ki se od kartezičnega produkta razlikujejo v definiciji množice povezav produkta[24].

Primer. Kartezični produkt k faktorjev K_2 je hiperkocka dimenzije k, $Q_k = \Box_{i=1}^{k} K_2 = K_2\Box \ldots \Box K_2$. Vozlišča Q_k so torej k-terice ničel in enic, dve vozlišči pa sta povezani natanko tedaj, ko se razlikujeta v natanko eni komponenti. (Glej definicijo na strani 19 in nalogo 1.13.)

[23] Kartezični produkt je torej komutativna in asociativna operacija (Glej nalogo 1.12.). Enota je K_1. Kartezični produkt ima lastnost enolične faktorizacije: vsak graf je mogoče do izomorfizma in do vrstnega reda natančno enolično zapisati kot produkt enostavnih faktorjev (*pragrafov*). Glej npr. uvodni članek [J.Žerovnik, O kartezičnem produktu grafov, Obzornik za matematiko in fiziko 42 (1995) 8-16].

[24] Produkti grafov so moderno specialno področje teorije grafov, z vidnimi prispevki slovenskih matematikov, glej knjigo [W.Imrich, S.Klavžar, Graph products, structure and recognition, John Wiley & Sons, New York, 2000].



Slika 1.19: Kartezični produkt grafov.

1.5. Predstavitve grafov

Doslej smo videli dva načina predstavitve grafov – kot risbo grafa, sestavljeno iz vozlišč, povezanih s črtami, ter kot urejen par z množico vozlišč in seznamom povezav. Grafična predstavitev je koristna v mnogih primerih, posebno če želimo opazovati strukturo vsega grafa, njena vrednost pa se zmanjša, brž ko moramo opisovati velike in zapletene grafe. Če želimo na primer v računalniku shraniti velik graf, potem je grafična predstavitev neprimerna. Uporabiti moramo kakšno drugo metodo.

Ena možnost je, da shranimo seznam vozlišč in k vsakemu vozlišču zapišemo seznam sosednjih vozlišč. Graf enostavno rekonstruiramo tako, da povežemo vsako vozlišče z njegovimi sosedami. Ta način pogosto uporabljamo v praksi, posebno kadar je graf „redek“ - to je takrat, ko ima graf veliko vozlišč in relativno malo povezav. Naslednja načina uporabljata matrike. Pri prvem zapišemo tabelo, v kateri označimo, kateri pari vozlišč so povezani, pri drugem pa tabelo, v kateri označimo, katera vozlišča so incidentna s katerimi povezavami.[25]

Matriki s k vrsticami in l stolpci rečemo matrika razsežnosti $k \times l$ ali kar matrika $k \times l$. Matrike so posebej primerne za računanje in v mnogih primerih lahko z njimi na najnaravnejši način formuliramo problem. Obstajajo različne vrste matrik, s katerimi lahko predstavimo graf. Tu bomo opisali dve najpomembnejši vrsti: *matriko sosednosti* in *incidenčno matriko*. Zaradi enostavnosti se tu omejimo na grafe brez zank, dovolimo pa večkratne povezave.

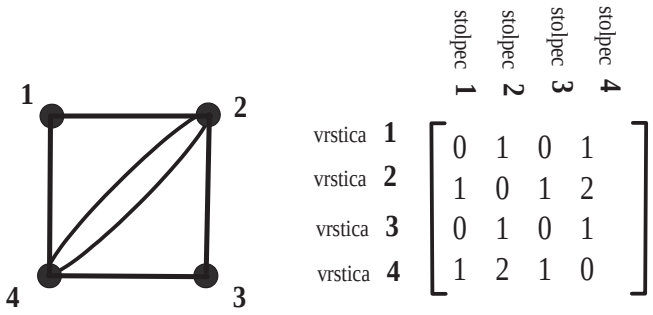
Matrika sosednosti

Oglejmo si naslednji primer na sliki 1.20:

Na levi strani imamo graf s *štirimi vozlišči*, na desni strani pa imamo matriko 4×4. Števila v matriki pomenijo število povezav, ki povezujejo ustrezni vozlišči grafa, na primer:

- vozlišči 1 in 2 sta povezani z eno povezavo, zato se pojavi v drugem stolpcu prve vrstice in v prvem stolpcu druge vrstice 1;
- vozlišči 2 in 4 sta povezani z dvema povezavama, zato je v matriki v četrtem stolpcu druge vrstice in v drugem stolpcu četrte vrstice 2;

[25] Zapis grafa z matriko povezuje teorijo grafov z linearno algebro. Izkaže se, da je mogoče nekatere rezultate iz linearne algebre koristno uporabiti pri obravnavi teorije grafov, pa tudi obratno. Glej na primer [R.A.Brualdi, The symbiotic relationship of combinatorics and matrix theory, Linear Algebra Appl. 162-164 (1992) 65-105.]



Slika 1.20: Graf in matrika sosednosti $A(G)$.

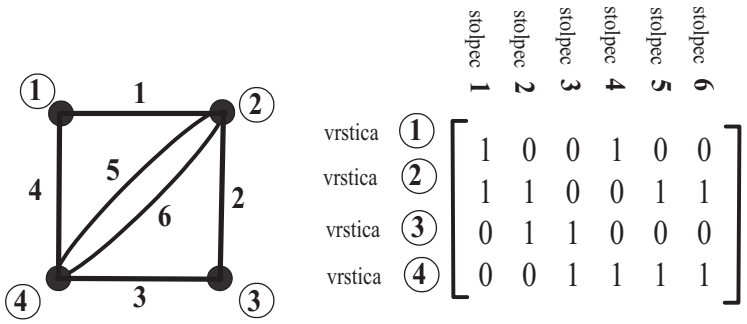
vozlišči 1 in 3 nista povezani (ali: sta povezani z nič povezavami), zato je v tretjem stolpcu prve vrstice in v prvem stolpcu tretje vrstice 0.

Opombi. Vsa števila na glavni diagonali (levo-zgoraj proti desno-spodaj) so enaka 0, saj graf nima nobene zanke. Vidimo tudi, da je matrika simetrična glede na to diagonalo.

To idejo lahko posplošimo takole: Naj bo G graf brez zank z n vozlišči, označenimi z 1, 2, 3, ..., n. **Matrika sosednosti** $A(G)$ je matrika razsežnosti $n \times n$, v kateri element v j-tem stolpcu i-te vrstice pove število povezav, ki povezujejo vozlišči i in j.

Incidenčna matrika

V matriko sosednosti zapišemo podatke o sosednosti vozlišč. V incidenčno matriko pa zapišemo informacije, katere vozlišča in povezave so incidentne. Poglejmo naslednji primer; oznake vozlišč smo obkrožili zato, da jih ločimo od oznak na povezavah.



Slika 1.21: Graf in incidenčna matrika $I(G)$.

Na levi imamo graf s *štirimi vozlišči* in *šestimi povezavami*, na desni strani pa 4×6 matriko. Elementi matrike so 1 ali 0, odvisno od tega, ali sta ustrezno vozlišče in povezava incidentni ali ne. (vozlišče in povezava sta incidentni, če je vozlišče krajišče povezave.) Tako na primer:

- vozlišče ① leži na povezavi 4, zato imamo v četrtem stolpcu prve vrstice **1**;
- vozlišče ② ne leži na povezavi 4, zato imamo v četrtem stolpcu druge vrstice **0**.

To idejo posplošimo takole: Naj bo G graf brez zank z n vozlišči, označenimi z ①, ②, ③,..., ⓝ, in z m povezavami, označenimi z 1,2,3,...,m. **Incidenčna matrika** $I(G)$ je matrika razsežnosti $n \times m$, katere element v i-ti vrstici in j-tem stolpcu je enak 1, če vozlišče ⓘ leži na povezavi j in 0 sicer. Očitno je incidenčna matrika odvisna od tega, kako označimo vozlišča in povezave grafa. Iz ene incidenčne matrike grafa dobimo drugo incidenčno matriko tako, da zamenjamo nekatere vrstice (ustrezno spremembam oznak vozlišč) in nekatere stolpce (ustrezno spremembam oznak povezav).

1.6. Primeri uporabe grafov

Za konec poglavja si oglejmo nekaj primerov uporabe grafov in nekaj zanimivih področij teorije grafov, ki jih kasneje ne bomo podrobneje obravnavali, vendar si iz različnih razlogov zaslužijo, da jih vsaj omenimo.

Ravninski grafi in problem štirih barv

Graf je **ravninski**, če ga je mogoče narisati v ravnini tako, da se noben par povezav ne seka. Ali drugače povedano, povezave se lahko dotikajo samo v točkah grafa, drugje pa ne.

Ravninski graf je **vložen v ravnino**, če je narisan v ravnino tako, da se noben par povezav ne seka. Sklenjena območja, ki jih omejujejo narisane povezave grafa, imenujemo **lica**. Preprosti primeri ravninskih grafov so poti, cikli, drevesa in platonski grafi (na sliki 1.17). Na sliki 1.1 sta dve ravninski risbi grafa K_4, zato je graf ravninski.[26]

Brez dokazov navedimo nekaj izrekov o ravninskih grafih.[27]

IZREK 1.2. (EULER) *Naj bo G povezan ravninski graf, vložen v ravnino. Naj bo $n = |V(G)|$, $m = |E(G)|$ in f število lic te vložitve. Potem velja*

$$n - m + f = 2 \ .$$

POSLEDICA 1.3. *Za povezan ravninski graf velja $m \leq 3n - 6$.*

POSLEDICA 1.4. *Za povezan ravninski graf brez trikotnikov velja $m \leq 2n - 4$.*

POSLEDICA 1.5. *Grafa K_5 in $K_{3,3}$ nista ravninska.*

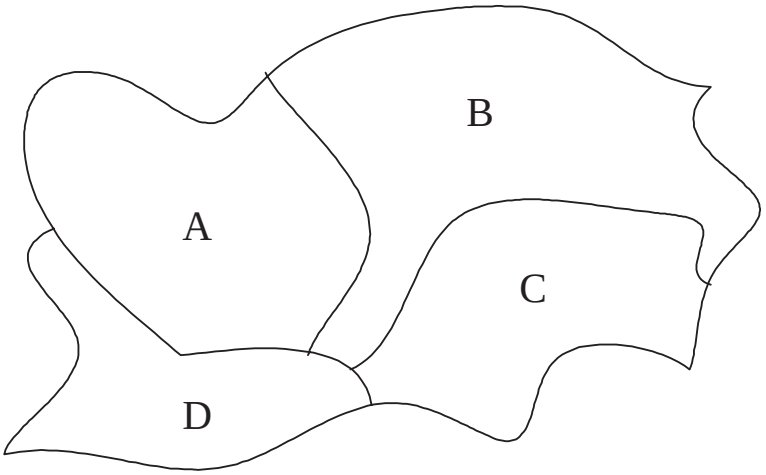
[26] Ravninski grafi so sferni grafi. Vsak graf, ki ga lahko vložimo v ravnino, lahko vložimo tudi na sfero. Uporabimo takoimenovano stereografsko projekcijo. Eno od lic na sferi se tako preslika v neskončno na ravnini.

[27] Dokazi so sorazmerno preprosti, razen dokaza izreka Kuratowskega, ki je nekoliko težji. Glej npr. [R.Wilson, M.Watkins, Uvod v teorijo grafov, DMFA Ljubljana 1997.].

IZREK 1.6. (KURATOWSKI) *Graf je ravninski natanko tedaj, ko ne vsebuje podgrafa homeomorfnega K_5 ali $K_{3,3}$.*

Grafa G in H sta **homeomorfna**, če ju lahko dobimo iz istega grafa tako, da na povezave dodamo nekaj novih vozlišč (stopnje 2).

Za konec omenimo enega najslavnejših problemov iz zgodovine teorije grafov, **problem štirih barv.** Pred dobrimi sto leti, točneje leta 1852, je mladi angleški matematik Francis Guthrie domneval, da je mogoče vsak zemljevid pobarvati s štirimi barvami, če zahtevamo, da je vsak par sosednjih držav pobarvan različno. Državi sta sosednji, če imata skupno mejo. Če se dve državi dotikata samo v eni točki, potem dovolimo, da sta pobarvani enako.[28]



Slika 1.22: Štiri države.

Tako moramo na primer na sliki 1.22 pobarvati različno državi A in B, A in D, B in C, B in D ter C in D, medtem ko sta A in C lahko enake barve.

Če si v vsaki državi izberemo točko (lahko si mislimo, da je to glavno mesto) in z železniškimi progami povežemo glavna mesta tako, da gre proga med glavnima mestoma držav A in B samo po ozemlju držav A in B, dobimo graf. Vsak tako dobljeni graf se da narisati v ravnini tako, da se noben par povezav (železniških prog) ne seka zunaj glavnih mest. Tak graf je ravninski graf. Izkaže se, da poljubnemu zemljevidu ustreza neki ravninski graf in obratno: vsak ravninski graf lahko dopolnimo do zemljevida. Pri danem ravninskem grafu je takih zemljevidov lahko veliko, saj meje niso natanko določene. Vemo namreč samo to, da meja med dvema državama poteka nekje po ozemlju med obema glavnima mestoma. Zanimivo pa je naslednje; če je mogoče vozlišča ravninskega grafa dobro pobarvati[29] s k barvami, potem je tudi (vsak) pripadajoči zemljevid mogoče pobarvati s k barvami. Velja pa tudi obratno. Tako dobimo drugo obliko Guthrijeve domneve, ki so ji rekli tudi „domneva štirih barv“: **Vsak ravninski graf je mogoče pobarvati s štirimi barvami.**

[28] Privzamemo tudi, da so države povezana območja. Če bi dovolili enklave, bi lahko dobili primere, ki jih ni mogoče pobarvati s štirimi barvami.

[29] O barvanjih grafov bomo več povedali v poglavju 4.

Preveriti, ali je ta domneva pravilna, je bil slavni „Problem štirih barv“. Za dokaz te na videz tako enostavne trditve je bilo potrebno ogromno truda mnogih matematikov. Tudi mnogi ljubitelji so poskušali, saj vsaj za razumevanje problema ni potrebno veliko predznanja. Morda je zanimivo vprašanje, koliko časa bi ljudje še potrebovali, če se ne bi v tem stoletju pojavili računalniki. Za dokaz, ki obsega več kot 600 strani knjige, so pri preverjanju mnogih podprimerov izdatno pomagali računalniški programi, ki so jih razvili Appel, Haken in Koch. Tudi njihovo delo je trajalo nekaj let, dokler niso leta 1976 objavili dokaza.[30]

Od leta 1976 lahko Guthrijevi domnevi rečemo *Izrek štirih barv*. Sedaj je torej znano, da je mogoče vsak zemljevid pobarvati s štirimi barvami.

Topološki indeksi molekulskih grafov

Kemiki so doslej uspeli sintetizirati na milijone različnih molekul. Skupaj z njihovimi ugotovljenimi lastnostmi je to ogromna količina znanja. Že vprašanje, kako naj bodo ti podatki shranjeni in organizirani, da jih bo mogoče čim bolje izkoristiti, je zelo zanimivo. Nas pa zdaj zanima, ali in kako je mogoče to znanje uporabiti za napoved lastnosti novih molekul, ki še nikoli niso bile sintetizirane? Razlogov za tako vprašanje je več, morda najpomembnejša sta naslednja dva. Teoretično je različnih možnih molekul seveda poljubno veliko. Tudi če omejimo število atomov v molekuli, hitro dobimo ogromno možnosti, saj raste število različnih molekul izredno hitro[31]. Če število kandidatov ne bi bilo preveliko, lahko postane pomemben strošek sinteze nove molekule, v tekmi med (na primer) proizvajalci zdravil pa je zelo pomemben tudi čas.

Fizikalni modeli molekul[32] lahko dobro napovedo lastnosti molekule, vendar so taki izračuni običajno izredno zahtevni, predvsem pa zanje potrebujemo ogromno časa.

Ali je mogoče s preprostim računom napovedati lastnosti molekule, še preden je sintetizirana? To bi bilo preveč enostavno. Morda pa je to mogoče vsaj za kakšno od lastnosti, in to s precejšnjo zanesljivostjo? Izkazalo se je, da je mogoče s pomočjo teorije grafov v mnogih primerih na ti vprašanji odgovoriti pritrdilno.

V naravi je molekula tridimenzionalni objekt. V nekoliko poenostavljenem jeziku lahko rečemo, da jo sestavljajo atomi, ki so lahko različnih „tipov“, in kemijske vezi med atomi.

Kemijski graf molekule je poenostavljena kombinatorična predstavitev molekule. Atomi so vozlišča grafa, vezi pa povezave grafa. V molekuli metana, na primer, ležijo štirje vodikovi atomi v ogliščih tetraedra, ogljikov atom pa v središču. Graf, ki ga dobimo, ima eno vozlišče stopnje štiri in štiri vozlišča stopnje ena. Pri tem, ko iz molekule naredimo graf, zavestno zanemarimo informacije,

[30] Kot v vsakem večjem programu, so kasneje morali popraviti še nekaj napak, vendar nobena ni bila usodna. Še vedno je zanimiva naloga poiskati krajši dokaz izreka, sedaj je ta dolg približno 40 strani, s tem da še vedno uporabi računalniški program, glej [R.Thomas, An update on the four-color theorem, Notices Amer. Math. Soc. 45 (1998), 848-859] in http://www.math.gatech.edu/ thomas/FC/fourcolor.html.

[31] Arthur Cayley je leta 1875 rešil problem preštevanja označenih dreves. Dokaz njegovega izreka, da je število označenih dreves na n vozliščih enako n^{n-2}, s uporabo Prüferjevih zaporedij najdemo v [Wilson, Watkins: Uvod v Teorijo grafov, DMFA Ljubljana 1996]. Naloga preštevanja neoznačenih dreves je precej težja. Rešil jo je George Pólya leta 1935. Število alkanov (ogljikovodikov s samimi enojnimi vezmi med atomi) na primer z rastočim številom atomov ogljika raste takole: 1, 1, 1, 2, 3, 5, 9, 18, 35, 75, 159, 355, 802, 1858 in 4347 (pri $n = 15$). Reševanja teh problemov so se lotili predvsem zaradi vprašanja štetja števila izomer v kemiji.

[32] Glej npr.[J.Koller, Struktura atomov in molekul, DZS, Ljubljana 1992.]

kot so na primer koti med povezavami, položaj atomov, in podobno. Če so v molekuli različni atomi in različni tipi povezav, včasih vozlišča in povezave opremimo z utežmi. Vsaj v primeru, ko obravnavamo ogljikovodike, pa se izkaže, da pogosto zadošča, če študiramo samo skelet molekule, ki je sestavljen iz ogljikovih atomov, torej enostaven graf brez uteži.

Topološki indeks je funkcija, ki molekulskemu grafu priredi število. Topološki indeksi so *grafovske invariante*. V kemiji je znanih preko 120 topoloških indeksov[33], njihovo število pa še vedno narašča. V nekaterih primerih se splača uporabiti primerno kombinacijo različnih indeksov.

Večina indeksov temelji na povezanosti in na razdaljah v grafu. Eden prvih, in verjetno najbolj znan na razdaljah grafa temelječi topološki indeks, je **Wienerjev indeks** ali **Wienerjevo število**. Leta 1947 je Henry Wiener opazil zelo dobro korelacijo med vreliščem alkanov[34] in nekim številom, ki ga je dobil s preprostim računom iz grafa molekule. V nadaljevanju si bomo ta indeks nekoliko pobliže ogledali.

Preprostejše invariante, kot je na primer število atomov molekule, so kemiki uporabljali že mnogo prej, široko uporabnost Wienerjevega števila in drugih indeksov pa so zares opazili in začeli uporabljati šele konec 70-tih let. V zadnjih letih so mnogi raziskovalci preizkusili številne druge indekse, s katerimi napovedujejo različne lastnosti na raznih tipih molekul. Na to temo je bilo v letih, ki so pretekla, objavljeno ogromno znanstvenih člankov. Tema pa očitno še ni izčrpana[35].

Wienerjevo število je po definiciji vsota vseh razdalj grafa,

$$W(G) = \sum_{u,v \in V(G)} d(u,v) \tag{1.1}$$

zato lahko Wienerjevo število zelo preprosto izračunamo iz matrike razdalj grafa. Za računanje Wienerjevega števila dreves obstaja hitrejša metoda.[36]

Wiener je študiral vrelišča alkanov in opazil izredno dobro korelacijo med izmerjenimi vrelišči in tem številom. Pokazal je tudi zvezo z drugimi fizikalnimi lastnostmi alkanov, na primer z volumnom molekule, izparilno toploto, in drugimi.

Szegedovo število grafa je[37]

$$Sz(G) = \sum_{e \in E(T)} w(e) \ , \tag{1.2}$$

[33] [N. Trinajstić, Chemical Graph Theory, second edition, CRC Press, Boca Raton FL 1992.]

[34] Wiener jih z zdaj nekoliko zastarelo terminologijo imenuje „parafini“.

[35] Glej uvodni članek [D.H.Rouvray, Predicting Chemistry From Topology, Scientific American September 1986, 36-]. Ob petdeseti obletnici odkritja Wienerjevega indeksa (1997) sta izšli posebni številki revij Discrete Applied Mathematics št.1 vol. 80 (1997) in Communications in Mathematical and Computer Chemistry (MATCH) vol. 35 (1997), ur.I.Gutman, S.Klavžar in B.Mohar.

[36] Glej primer 6.5. na strani 120.

[37] Definicijo dobimo z direktno posplošitvijo lastnosti Wienerjevega števila na drevesih, po kateri je Wienerjevo število drevesa vsota uteži povezav, utež povezave pa je število najkrajših poti, ki gredo čez povezavo. (glej tudi nalogo 3.14.).

kjer je

$$w(e) = n_{e,u}n_{e,v}$$

in smo z $n_{e,x}$ označili število vozlišč, ki so bližje krajišču x povezave e kot drugemu krajišču povezave e.[38]

1.7. Naloge z rešitvami

1.1. Koliko vozlišč in povezav imajo naslednji grafi? (a) Grafi na sliki 1.7 (s strani 14). (b) Grafi na sliki 1.5 (s strani 12). (c) Grafi na sliki 1.8 (s strani 15).

1.2. Izpolni naslednjo tabelo:

	K_9	N_9	C_9	$K_{9,9}$	Q_5	tetra-eder	kocka	okta-eder	dodeka-eder	ikoza-eder	Peter-senov
število vozlišč											
število povezav											
stopnje vozlišč											

1.3. Graf G ima devet vozlišč: dve vozlišči sta stopnje 1, štiri so stopnje 2 in dve stopnje 3. Eno je stopnje 4. Koliko povezav ima ta graf?

1.4. Graf G ima sedem vozlišč: dve vozlišči sta stopnje 1, dve sta stopnje 2 in tri stopnje 3. Koliko povezav ima ta graf?

1.5. Zapiši zaporedji stopenj za grafe: (a) na sliki 1.14 (s strani 18); (b) na sliki 1.7 (s strani 14); (c) za vsa drevesa na sedmih vozliščih (glej tudi nalogo 3.4. na strani 71).

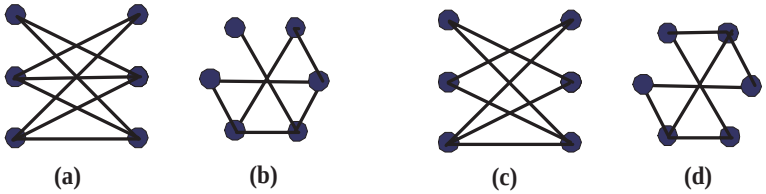
1.6. Ali obstaja graf z zaporedjem stopenj (a) (1,2,3,3,3,4) (b) (1,2,3,4,5)?

1.7. Dokaži posledice leme o rokovanju.

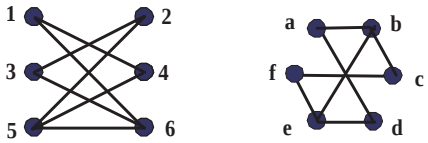
[38] I. Gutman, Formula for the Wiener Number of Trees and Its Extension to Graphs Containing Cycles, Graph Theory Notes New York **27** (1994) 9-15.

1.8. Dokaži, da grafa na sliki 1.7 s strani 14 nista izomorfna.

1.9. Kateri pari grafov na sliki so izomorfni?



1.10. Poišči izomorfizem med grafoma na sliki:



1.11. Dokaži, da sta grafa na sliki 1.18 izomorfna.

1.12. Dokaži, da je produkt $H \Box G$ izomorfen produktu $G \Box H$ in da sta izomorfna produkta $(G \Box H) \Box K$ in $G \Box (H \Box K)$.

1.13. Dokaži, da je kartezični produkt k faktorjev K_2, $\Box_{i=1}^{k} K_2$, izomorfen hiperkocki Q_k.

1.14. Dokaži, da so hiperkocke dvodelni grafi.

1.15. Dokaži trditev: Graf je dvodelen natanko tedaj, ko nima nobenega lihega cikla.

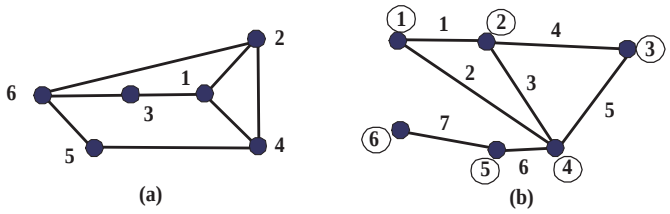
1.16. Izračunaj premer polnega dvodelnega graf $K_{r,s}$. (**Premer grafa** je največja razdalja v grafu.)[39]

1.17. Naj bo G poljuben enostaven graf. Dokaži, da je vsaj eden od grafov G, $\bar{G}$ povezan.

[39] Graf je povezan natanko tedaj, ko je premer grafa $< \infty$.

1.18. Koliko komponent povezanosti imajo naslednji grafi: (a) N_{11}, (b) Petersenov graf, (c) $\overline{K_{3,3}}$?

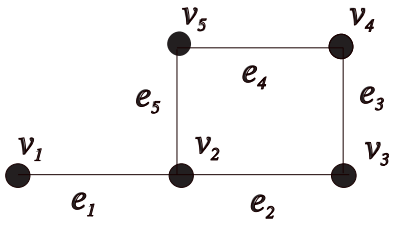
1.19. Zapiši matriko sosednosti za graf (a) in matriko sosednosti ter incidenčno matriko grafa (b).



1.20. Nariši graf z matriko sosednosti: $A(G) = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$

1.21. Za graf s slike 1.9 (s strani 15): (a) Zapiši vse sprehode dolžine 3 iz začetnega vozlišča a. (b) Poišči vsaj en cikel dolžine 3 in vsaj en cikel dolžine 4. Ali v grafu obstaja induciran cikel dolžine 5? (c) Zapiši razdalje od vozlišča g do vseh drugih vozlišč. (d) Poišči vse poti od vozlišča b do vozlišča h.

1.22. Izračunaj Wienerjevo število grafa na sliki



1.23. Izračunaj Wienerjevo in Szegedovo število grafov (a), (b), (c) in (d) iz naloge 1.9. s strani 28.

1.24. Izračunaj Wienerjevo število poti P_n in zvezde S_n.

Rešitve

1.1. (a) Grafa imata vsak po 8 vozlišč in 10 povezav.

(b) $n = 8$, $m = 16$; $n = 8$, $m = 8$; $n = 4$, $m = 6$.

(c) $n = 16$, $m = 15$.

1.2.

	K_9	N_9	C_9	$K_{9,9}$	Q_5	tetra-eder	kocka	okta-eder	dodeka-eder	ikoza-eder	Peter-senov
število vozlišč	9	9	9	18	32	4	8	6	20	12	10
število povezav	36	0	9	81	80	6	12	12	30	30	15
stopnje vozlišč	8	0	2	9	5	3	3	4	3	5	3

1.3. 10.

1.4. Tak graf ne obstaja. Po lemi o rokovanju bi moral imeti $2 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 = \frac{15}{2} = 8 + \frac{1}{2}$ povezav.

1.5. (a) (1,1,2,2,2,2,2), (1,1,1,2,2,3) (dva grafa), (1,1,1,1,3,3), (1,1,1,1,2,4), (1,1,1,1,1,5).

(b) Grafa imata enaki zaporedji stopenj: (2,2,2,2,3,3,3,3).

(c) (1,1,2,2,2,2), (1,1,1,2,2,2,3) (trije grafi), (1,1,1,1,2,3,3), (1,1,1,1,2,2,4), (1,1,1,1,2,3,3), (1,1,1,1,2,2,4), (1,1,1,1,1,3,4), (1,1,1,1,1,2,5), (1,1,1,1,1,1,6).

1.6. (a) Takih grafov je več.

(b) Ker je vsota 1+2+3+4+5 = 15 liho število, grafa z zaporednjem stopenj (1,2,3,4,5) ni.

1.7.

1. V vsakem grafu je vsota vseh stopenj vozlišč grafa sodo število.
   Dokaz: Po lemi o rokovanju je vsota stopenj enaka $2m$, torej sodo število.
2. V vsakem grafu je število vozlišč lihe stopnje sodo.
   Dokaz: Recimo, da bi obstajal graf, v katerem bi imeli liho mnogo vozlišč lihe stopnje. Ker je vsota liho mnogo lihih števil liho število, bi v takem grafu bila vsota stopenj liho število. Število povezav m je po lemi o rokovanju polovica te vsote in ne bi bila celo število, zato tak graf ne obstaja.
3. Če ima graf G n vozlišč in je regularen stopnje r, ima G natanko $\frac{1}{2}nr$ povezav.
   Dokaz: Vsota stopenj je nr. Število povezav m je po lemi o rokovanju polovica te vsote, torej $m = \frac{1}{2}nr$.

1.8. Vidimo, da sta v prvem grafu dve sosednji vozlišči stopnje 2. V drugem grafu takega para vozlišč ni.

Prepričajmo se, da se ta lastnost pri izomorfizmu ohranja. Najprej se spomnimo, da se po definiciji izomorfizma *sosednja vozlišča preslikajo v sosednja.* Izberimo si poljubno vozlišče v in opazujmo sosede. Izomorfizem α jih preslika v sosede slike $\alpha(v)$. Ker je izomorfizem bijektivna preslikava, je število sosedov vozlišča v enako številu sosedov vozlišča $\alpha(v)$. Torej *izomorfizem ohranja stopnjo vozlišča.* Formalno in bolj natančno zapisano, velja

$$N(\alpha(v)) = \alpha(N(v)) \ ,$$

izomorfizem preslika soseščino vozlišča v soseščino slike.

V našem primeru odtod sklepamo, da se morata sosednji vozlišči stopnje 2 preslikati v sosednji vozlišči stopnje 2.

1.9. Graf (a) ima osem povezav, ostali trije grafi pa po sedem, zato graf (a) ni izomorfen nobenemu od ostalih treh.

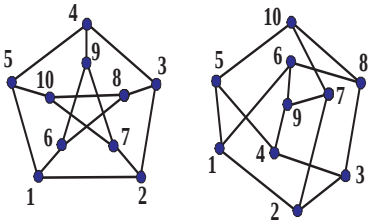
Graf (b) ima zaporedje stopenj (1,2,2,3,3,3), grafa (c) in (d) pa imata zaporedji (2,2,2,2,3,3), zato graf (b) ni izomorfen grafoma (c) in (d).

Grafa (c) in (d) sta izomorfna. Izomorfizma ni težko poiskati (glej nalogo 1.10.).

1.10. Izomorfizem je (na primer): $1 \mapsto f$, $2 \mapsto a$, $3 \mapsto d$, $4 \mapsto c$, $5 \mapsto b$, $6 \mapsto e$.

1.11.

Ker sta grafa neoznačena, lahko poljubno izberemo oznake vozlišč. Izberimo jih tako, da bo izomorfizem kar identiteta (vozlišče se preslika v vozlišče z isto oznako). Ni težko preveriti, da so v obeh grafih povezani pari vozlišč z istimi oznakami. (Na primer: sosedi vozlišča 1 so 2,5 in 6, itd.)



1.12. Namig: Izomorfizem $H\Box G \to G\Box H$ lahko definiramo s predpisom $(u, v) \mapsto (v, u)$, izomorfizem $(G\Box H)\Box K \to G\Box(H\Box K)$ pa s predpisom $((u, v), w) \mapsto (u, (v, w))$.

1.13. Po definiciji s strani 19 so vozlišča hiperkocke Q_k vse besede nad abecedo $\{0, 1\}$ (zaporedja ničel in enic) dolžine k. Vozlišči sta povezani natanko tedaj, ko se besedi razlikujeta na natanko enem mestu.

Kartezični produkt (glej stran 21) k faktorjev K_2 je graf, katerega vozlišča so vse urejene k-terice $v = (v_1, v_2, \ldots, v_k)$, kjer je $v_i \in \{0, 1\}$. Vozlišči sta povezani natanko tedaj, ko imata vse razen ene koordinate enake.

Očitno je preslikava, ki vozlišču $v = (v_1, v_2, \ldots, v_k)$ priredi vozlišče (besedo) $v_1 v_2 \ldots v_k$, izomorfizem.

1.14. Ideja: Množico vozlišč razdelimo v množico vozlišč z liho mnogo enicami in množico vozlišč s sodo mnogo enicami.

1.15. Na poljubnem obhodu se barvi (črna in bela) zaporednih vozlišč izmenjujeta. Torej ima poljuben obhod sodo dolžino.

Recimo, da je graf povezan in da v grafu ni nobenega lihega cikla. Izberimo si poljubno vozlišče v in definirajmo $v \in A$. Ali, z drugimi besedami, recimo, da je vozlišče belo. Vse sosede vozlišča v dajmo v množico B, torej naj bodo črna vozlišča. Sosede črnih vozlišč pobarvajmo z belo in tako dalje. Tako lahko pobarvamo vsa vozlišča. Barva vozlišča je na ta način dobro definirana. Če bi namreč za eno od vozlišč, na primer u, lahko dobili belo in črno barvo, bi to pomenilo, da obstaja med v in u pot lihe in pot sode dolžine. Ni težko videti, da odtod sledi, da v grafu obstaja cikel lihe dolžine. To pa po predpostavki ni mogoče. Torej je graf dvodelen.

Če graf G ni povezan in nima nobenega lihega cikla, je po pravkar povedanem vsaka povezana komponenta grafa dvodelen graf. Potem pa je tudi graf G dvodelen.

1.16.

Naj bo $V = V_1 \cup V_2$ razbitje množice vozlišč dvodelnega grafa $K_{r,s}$.

Izberimo si poljuben par vozlišč u in v. Če sta vozlišči u in v v različnih množicah razbitja, potem sta na razdalji ena. Če sta u in v različni vozlišči iz iste množice, recimo $u, v \in V_1$, potem uporabimo poljubno vozlišče iz druge množice, na primer $w \in V_2$, in dobimo pot med u in v dolžine dva, uwv. Razdalja med u in v je torej enaka dva.

Premer polnega dvodelnega grafa je torej enak 2.

Opomba: ker je premer grafa $K_{r,s}$ enak 2 (torej manj kot ∞), je vsak poln dvodelen graf $K_{r,s}$ povezan.

1.17. Če je graf G povezan, potem ni kaj dokazovati.

Pa recimo, da graf G ni povezan. Izberimo poljubno komponento povezanosti grafa G. Označimo z V_1 množico vozlišč te komponente in naj bo $V_2 = V \setminus V_1$. Ker v G ni nobene povezave z enim krajiščem v V_1 in drugim krajiščem v V_2, so v $\bar{G}$ vse take povezave. Torej G vsebuje kot vpet podgraf poln dvodelen graf (z razbitjem $V = V_1 \cup V_2$). Ker je vsak poln dvodelen graf povezan (glej nalogo 1.16.), je $\bar{G}$ povezan.

1.18. (a) N_{11} ima 11 komponent povezanosti.

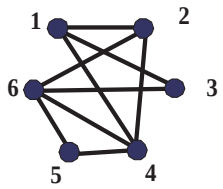
(b) Petersenov graf je povezan, torej ima eno povezano komponento.

(c) $\overline{K_{3,3}}$ ima 2 komponenti povezanosti.

1.19. (a) $A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$

(b) $A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ $I = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

1.20.



1.21. (a) *abab*, *abeb*, *abed*, *abef*, *abcb*, *abcd*, *abkb*.

(b) *efd* je cikel dolžine tri (trikotnik). *ijln* je cikel dolžine štiri, ki ni induciran, cikel *bcde* pa je induciran cikel dolžine štiri. V grafu ni nobenega induciranega cikla dolžine 5, obstajajo pa cikli dolžine 5, ki niso inducirani, na primer *bcdfe*.

(c) Razdalje od vozlišča g do drugih vozlišč so:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
razdalje	4	3	3	2	2	1	0	2	3	4	4	5	5	4	4	3

(d) *befh*, *bedfh*, *bedph*, *bcdefh*, *bcdfh*, *bcdph*.

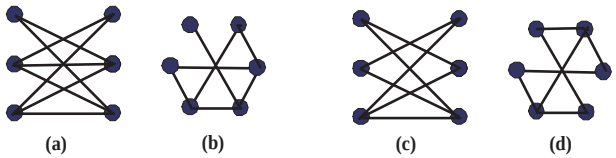
1.22.

par vozlišč	$\{v_1, v_2\}$	$\{v_1, v_3\}$	$\{v_1, v_4\}$	$\{v_1, v_5\}$	$\{v_2, v_3\}$
razdalja	1	2	3	2	1

par vozlišč	$\{v_2, v_4\}$	$\{v_2, v_5\}$	$\{v_3, v_4\}$	$\{v_3, v_5\}$	$\{v_4, v_5\}$
razdalja	2	1	1	2	1

Če izračunamo vsoto vseh razdalj v grafu iz prve tabele, dobimo 16, torej Wienerjevo število grafa $W(G) = 16$

1.23.



(a) $W = 23$, $Sz = 68$.

(b) $W = 25$, $Sz = 53$.

(c) $W = 25$, $Sz = 59$.

(d) $W = 25$, $Sz = 59$.

Spomnimo se, da sta grafa (c) in (d) izomorfna (naloga 1.9.). W in Sz sta očitno grafovski invarianti, saj je njuna definicija neodvisna od označitve vozlišč in povezav. Res sta rešitvi za (c) in (d) enaki.

1.24.

$$\begin{aligned} W(P_n) &= 1\cdot(n-1)+2\cdot(n-2)+3\cdot(n-3)+\ldots(n-1)\cdot 1 \\ &= \sum_{i=1}^{n-1} i(n-i) \\ &= n\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 \\ &= n\frac{(n-1)n}{2} - \frac{(n-1)n(2n-1)}{6} = \frac{1}{6}(n-1)n(n+1) \end{aligned}$$

$$W(S_n) = (n-1) + 2\binom{n-1}{2} = (n-1)^2$$

2

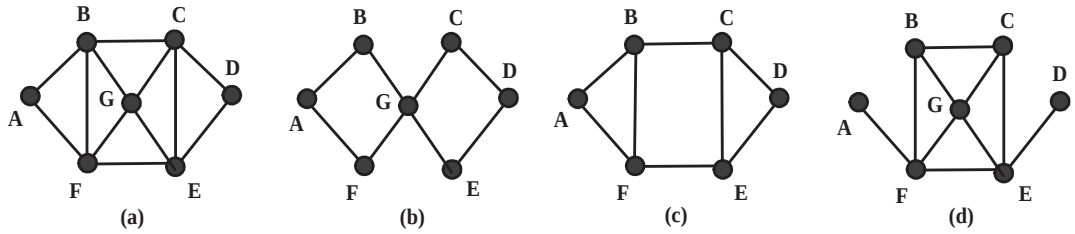
EULERJEVI IN HAMILTONOVI GRAFI

2.1. Uvod

Povezan graf je **Eulerjev**[1], če obstaja enostaven obhod, na katerem so vse povezave grafa. Tak obhod imenujemo **Eulerjev obhod**.

Povezan graf je **Hamiltonov**[2], če obstaja cikel, na katerem so vsa vozlišča grafa. Tak cikel imenujemo **Hamiltonov cikel**.

Kot primer si oglejmo štiri grafe na sliki:



Graf (a) je Eulerjev in Hamiltonov;
Graf (b) je Eulerjev in ni Hamiltonov; Eulerjev obhod je $ABGCDEGF$;
Graf (c) je Hamiltonov in ni Eulerjev; Hamiltonov cikel je $ABCDEF$;
Graf (d) ni niti Eulerjev niti Hamiltonov.

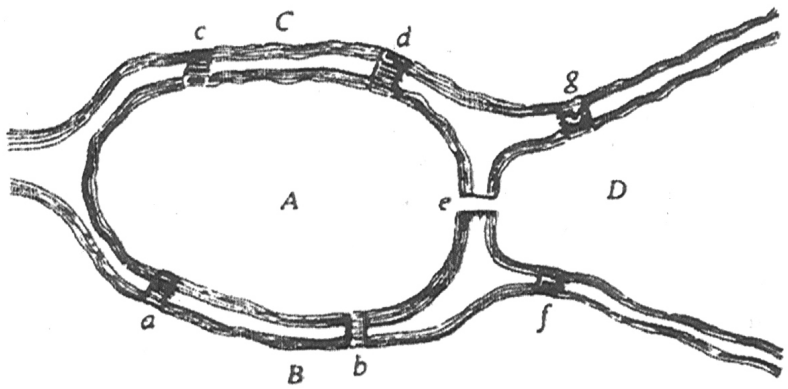
V nadaljevanju bomo obravnavali oba tipa grafov. Spoznali bomo *potreben in zadosten* pogoj za to, da je povezan graf Eulerjev, ter dva *zadostna* in en *potreben* pogoj za to, da je povezan graf Hamiltonov.

[1]Leonhard Euler (1707-1783) sodi med najplodnejše matematike vseh časov; objavil je številne prispevke na različnih področjih matematike. Eulerjev dokaz izreka, ki je bil objavljen v članku leta 1736 z naslovom *Solutio problematis ad geometriam situs pertinensis*, sicer ni bil napisan v jeziku teorije grafov, vendar ga zaradi narave uporabljenih idej upravičeno lahko štejemo za prvi članek iz teorije grafov. Glej tudi [V.Domanjko, Leonhard Euler in razvedrilna matematika, Založba MATH 2000].

[2]Sir William Rowan Hamilton (1805-1865) je bil eden od vodilnih matematikov svojega časa. Njegova *algebra kvaternionov* ali *ikozaederski račun* (kot ga je imenoval sam) lahko izrazimo s Hamiltonovimi cikli pravilnega dodekaedra. Iz problema je naredil igro, ki jo je poimenoval *ikozaedrska igra* (Icosian game), v kateri mora igralec poiskati Hamiltonove cikle z danimi začetnimi petimi črkami.

2.2. Eulerjevi grafi

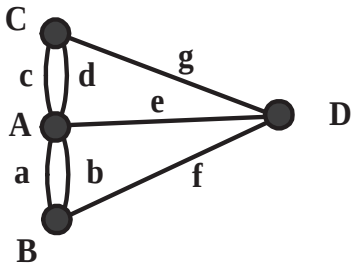
Štirje deli mesta Königsberg (A, B, C in D) so bili povezani s sedmimi mostovi (a, b, c, d, e in f), kot kaže naslednja slika:



Slika 2.1: Mostovi v Königsbergu.

Meščani so brez uspeha poskušali najti obhod mesta, ki bi prečkal vsak most natanko enkrat in se vrnil v izhodišče. Začeli so verjeti, da naloga nima rešitve. Dokazati tega ni znal nihče, dokler se problema ni lotil Euler.

V jeziku teorije grafov lahko problem königsberških mostov povemo takole. Štiri dele mesta uporabimo za vozlišča grafa, sedem mostov pa za povezave grafa. Problem poiskati *obhod mesta, ki bi prečkal vsak most natanko enkrat in se vrnil v izhodišče* natanko ustreza problemu iskanja *Eulerjevega obhoda v tem grafu*. Kot bomo videli, graf nima nobenega Eulerjevega obhoda. Torej ni nobenega obhoda, ki bi rešil problem königsberških mostov. Euler je obravnaval problem tudi na



Slika 2.2: Graf mostov v Königsbergu.

bolj splošnih razporeditvah delov mesta in mostov. Tako je našel splošno pravilo, kdaj je obhod iskanega tipa mogoče najti, torej kdaj je graf Eulerjev. Ugotovil je, da je mogoče najti obhod, ki prečka vsak most natanko po enkrat (torej poiskati Eulerjev obhod grafa), natanko tedaj, ko je

izpolnjen naslednji pogoj: *kadarkoli pridemo v neki del mesta, mora obstajati možnost, da ta del zapustimo po še neprehojenem mostu.* Z drugimi besedami: kadarkoli pridemo v vozlišče, moramo imeti možnost, da ga zapustimo po drugi povezavi. Torej vsakič, ko obiščemo vozlišče, prispevamo natanko 2 k stopnji tega vozlišča. (To velja tudi za prvo in zadnjo povezavo na obhodu, ki skupaj prispevata 2 k stopnji začetnega vozlišča.) Zato mora imeti v Eulerjevem grafu vsako vozlišče sodo stopnjo. Euler je opazil, da je pogoj tudi zadosten in (v drugačni terminologiji) dokazal izrek:

IZREK 2.1. (EULERJEV IZREK) *Naj bo G povezan graf. Potem je G Eulerjev natanko tedaj, ko ima vsako vozlišče G sodo stopnjo.*

Dokaz. Dokaz ima dva dela: (1.) Če je G Eulerjev, potem ima vsako vozlišče sodo stopnjo. (2.) Če ima v povezanem grafu vsako vozlišče sodo stopnjo, potem je G Eulerjev.

Prvi del dokazuje *potrebnost* pogoja, drugi del pa *zadostnost* pogoja.

(1.) Če G ima Eulerjev obhod, potem lahko potujemo vzdolž tega obhoda in bomo uporabili vsako povezavo natanko enkrat ter se vrnili v začetno vozlišče. Vsakič, ko obiščemo vozlišče, prispevamo 2 k stopnji vozlišča – vključno z začetnim vozliščem, saj v njem začnemo in končamo obhod. Ker vsako povezavo uporabimo natanko enkrat, je vsota prispevkov ravno enaka stopnji vozlišča. Ker so bili vsi prispevki enaki 2, so torej stopnje vozlišč sode.

(2.) Privzemimo, da imajo vsa vozlišča grafa G sodo stopnjo. Poiskati moramo Eulerjev obhod v G. Upoštevali bomo, da G vsebuje vsaj en cikel, označimo ga s C. (Dokaz te trditve je v nalogi 2.5.)

Zdaj bomo uporabili **matematično indukcijo**[3] na število povezav m. Za $m = 0$ je edini povezani graf K_1, ki je očitno Eulerjev. Recimo, da je trditev 2. pravilna za vse povezane grafe z manj kot m povezavami. Naj ima G m povezav. Iz G odstranimo cikel C. Dobljeni graf, imenujmo ga H, ima manj kot m povezav in vsako vozlišče grafa H ima sodo stopnjo. H morda ni povezan graf, vsaka povezana komponenta grafa H pa je povezan graf z vozlišči sode stopnje. Po indukcijski predpostavki je torej vsaka komponenta H Eulerjev graf. Eulerjev obhod v G lahko poiščemo takole: Začnemo v kateremkoli vozlišču v cikla C in gremo po ciklu C, dokler ne pridemo do prve komponente grafa H. Potem gremo po Eulerjevem obhodu te komponente in se vrnemo na cikel

[3]Metodo matematične indukcije uporabljamo za dokazovanje lastnosti, ki so smiselne za neskončne podmnožice naravnih števil. Osnova za dokaz je naslednja trditev:

- če velja lastnost P za neko naravno število n_0 in
- če za vsako naravno število $n \geq n_0$ velja sklep: če velja $P(k)$ za vsak k, $n_0 \leq k \leq n$, potem velja tudi $P(n+1)$,
- potem velja lastnost P za vsako naravno število $n \geq n_0$.

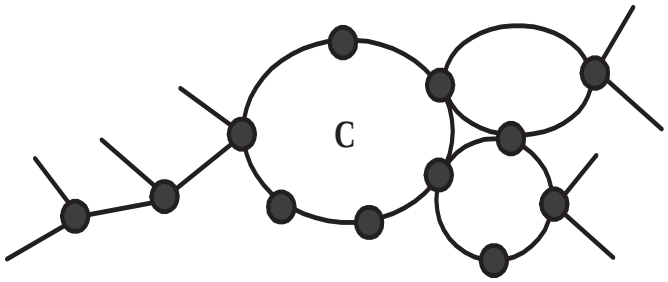
S $P(n)$ smo krajše zapisali izjavo „Lastnost P velja za naravno število n“.

Navodilo za dokaz lahko zapišemo takole:

1. Dokaži trditev za naravno število n_0.
2. Privzemi, da trditev velja za neko naravno število $n = k \geq n_0$ (in, če je potrebno, za vsa naravna števila med n_0 in k).
3. Dokaži, da mora pri predpostavki 2 trditev veljati za $n = k + 1$.

Ker trditev velja za $n = n_0$ (po točki 1) in, ker iz predpostavke 2 sledi trditev 3, velja trditev za $n = n_0 + 1$. Podobno vidimo, da mora veljati trditev za $n = (n_0 + 1) + 1 = n_0 + 2$, za $n = n_0 + 3$, in tako dalje...

Pri grafih najpogosteje naredimo indukcijo na število vozlišč ali indukcijo na število povezav.



Slika 2.3: K dokazu Eulerjevega izreka.

C. Tako nadaljujemo vzdolž C in vsakič, ko naletimo na novo komponento grafa H, prehodimo vse njene povezave po njenem Eulerjevem obhodu. Ko se nazadnje vrnemo v izhodiščno vozlišče v, smo prehodili vsako povezavo grafa G natanko po enkrat – našli smo torej Eulerjev obhod grafa G. □

Slaba stran gornjega dokaza je, da ni konstruktiven, saj nam ne pokaže postopka, po katerem bi našli Eulerjev obhod danega grafa. En način za konstrukcijo Eulerjevega obhoda je naslednji algoritem, ki ga bomo zapisali brez dokaza. Spomnimo se, da je *most* povezava v povezanem grafu, brez katere bi bil graf nepovezan.

> **FLEURYJEV ALGORITEM.** Če je G Eulerjev graf, potem lahko vedno izvedemo naslednje korake in dobimo Eulerjev obhod grafa G:

KORAK 1: Izberi začetno vozlišče.

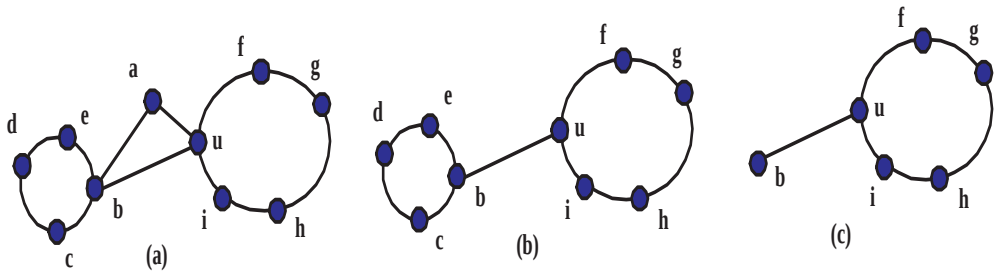
KORAK 2: Prečkaj poljubno povezavo, le most izberi samo, kadar ni na voljo nobene druge povezave.

KORAK 3: Prehojeno povezavo odstrani. Prav tako odstrani vsa vozlišča, ki so postala izolirana.

KORAK 4: Končaj, ko ni nobene povezave več.

Algoritem je sorazmerno preprost. Bistveno je, da na vsakem koraku uporabimo most samo *kot zadnjo možnost izhoda*. Če namreč prehitro uporabimo most, se ne moremo več vrniti v komponento, ki smo jo pravkar zapustili.

Fleuryjev algoritem ponazorimo na naslednjem grafu (a): Začenši v u lahko izberemo povezavo ua, potem pa ab. Ko odstranimo ti dve povezavi (in izolirano vozlišče a), dobimo graf (b). Povezave bu zdaj še ne smemo uporabiti, ker je most, zato izberimo povezavo bc, potem pa še cd, de in eb. Odstranimo uporabljene povezave (in vozlišča c d in e), pa dobimo graf (c). Zdaj izberemo povezavo bu, ki je prej nismo smeli izbrati, ker je bila most. Prehodimo še cikel $u\,f\,g\,h\,i\,u$ in zaključimo. Eulerjev obhod je torej $u\,a\,b\,c\,d\,e\,b\,u\,f\,g\,h\,i\,u$.



Slika 2.4: Fleuryjev algoritem.

2.3. Problemi Eulerjevega tipa

Poleulerjevi grafi

Kaj, če bi meščane Königsberga še vedno zanimal sprehod, ki bi prečkal vsak most natanko enkrat, vendar ne bi vztrajali, da se mora zaključiti v začetnem vozlišču? Ali bi obstajala rešitev problema pri tako omiljenih pogojih?

Odprt sprehod je sprehod, pri katerem sta začetno in končno vozlišče različni. Odprt sprehod, na katerem so vse povezave grafa, imenujemo **Eulerjev sprehod**. Povezan graf je **poleulerjev**, če obstaja odprt sprehod, na katerem so vse povezave grafa G. Z uporabo Eulerjevega izreka lahko enostavno dobimo potreben in zadosten pogoj za to, da je graf poleulerjev. Dokaz prepuščamo bralcu (glej nalogo 2.6.).

IZREK 2.2. *Naj bo G povezan graf. Potem je G poleulerjev natanko tedaj, ko ima G natanko dve vozlišči lihe stopnje.*

Uganke Eulerjevega tipa

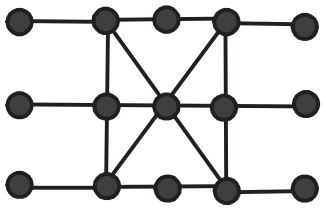
V knjigah z ugankami moramo pogosto narisati dani diagram s čim manj potezami, pri tem pa ne smemo narisati nobenega dela diagrama dvakrat. Tako je na primer enostavno narisati naslednji diagram s tremi potezami. Toda, ali morda gre z dvema? Izkaže se, da ne gre (glej nalogo 2.3.). Bolj natančno nalogo formuliramo takole: Poišči najmanjše število po povezavah disjunktnih enostavnih sprehodov, katerih unija vsebuje vse povezave grafa.

Kitajski problem poštarja

Pomemben problem, ki se je pojavil v različnih preoblekah, je tako imenovani *Kitajski problem poštarja.* [4] Lahko ga zastavimo takole:

Kitajski problem poštarja (ali **naloga kitajskega poštarja**). Poštar želi razdeliti pošto vzdolž

[4] Pridevnik *Kitajski* se nanaša na problem, ne na poštarja! Problem je formuliral leta 1962 Meigu Guan.



Slika 2.5: S koliko potezami lahko narišemo graf?

vseh ulic svojega rajona in se vrniti na poštni urad. Kakšno pot naj izbere, da bo prehodil najmanjšo možno razdaljo?

Če poštarjevemu rajonu po naključju ustreza Eulerjev graf, potem s problemom ni težav - poštar enostavno izbere Eulerjev obhod (uporabljajoč Fleuryjev algoritem, če je potrebno), in tak obhod bo gotovo imel najmanjšo možno dolžino. V praksi seveda mora običajno poštar nekatere dele ulic prehoditi večkrat in želi čimbolj skrajšati skupno dolžino teh delov. Privzeti smemo, da poznamo dolžino kateregakoli dela poštarjeve poti.

Podobni problemi so se pojavili v drugih kontekstih. Tak primer je obsežna študija o poteh plugov za čiščenje snega pred leti v Zürichu. Ker je obratovanje plugov za čiščenje snega drago, je bilo potrebno načrtati obhode tako, da bi čim manj ulic čistili po večkrat zapored. Tudi druga mesta so začela s podobnimi raziskavami za optimiranje pometanja in čiščenja ulic.

Spomnimo se, da je *utežen graf* ali *omrežje* graf, v katerem je vsaki povezavi prirejeno pozitivno število, ki ga imenujemo *utež*. Kitajski problem poštarja lahko z besednjakom uteženih grafov definiramo takole: *poišči zaprt sprehod z najmanjšo skupno težo, na katerem je vsaka povezava grafa vsaj enkrat.*

Splošna rešitev naloge je algoritem, v katerem kombiniramo ideje Fleuryjevega algoritma in algoritma za iskanje najkrajših poti (obravnavanega v poglavju 6).

Če je graf poleulerjev, lahko postopamo takole: med vozliščema lihe stopnje dodamo povezavo, ki ji damo utež enako dolžini najkrajše poti med njima. V dobljenem grafu poiščemo Eulerjev obhod. Rešitev naloge kitajskega poštarja na prvotnem grafu dobimo tako, da dodano povezavo zamenjamo s povezavami ene od najkrajših poti. Povezave na tej najkrajši poti bodo na poštarjevem obhodu nastopale po dvakrat.

Za grafe z več kot dvema vozliščema lihe stopnje lahko metodo priredimo tako, da dodamo najkrajše poti med temi vozlišči. Če želimo poiskati optimalno rešitev za nalogo kitajskega poštarja, moramo med vsemi pari vozlišč lihe stopnje poiskati najkrajše poti. Tako dobimo poln utežen graf na $2k$ vozliščih. (Za vozlišča vzamemo vozlišča originalnega grafa lihe stopnje, za uteži povezav pa dolžine najkrajših poti med njimi.) Iščemo k povezav tega grafa, ki „pokrijejo“ vsa vozlišča, in katerih vsota uteži je minimalna. Takemu naboru povezav rečemo **minimalno popolno prirejanje**. Izkaže se, da je to nalogo mogoče učinkovito rešiti, vendar so podrobnosti precej zapletene in jih moramo tu izpustiti.[5]

[5]Problem minimalnega popolnega prirejanja je ena od najtežjih nalog kombinatorične optimizacije, ki jo je mogoče rešiti v polinomskem času. Glej: [B.Korte in J.Vygen, Combinatorial Optimization, Theory and Algorithms,

2.4. Potrebni in zadostni pogoji za Hamiltonskost grafov

Na prvi pogled je videti naloga „za dani graf preveri, ali je Hamiltonov ali ne“, zelo podobna nalogi „za dani graf preveri, ali je Eulerjev ali ne“. Zato bi utegnili pričakovati, da obstaja preprost potreben in zadosten pogoj za hamiltonskost grafa, tako kot pogoj Eulerjevega izreka (izrek 2.1.) za Eulerjeve grafe. Vendar ne poznamo nobenega takega pogoja in iskanje potrebnih in zadostnih pogojev za hamiltonskost grafa je pomembno področje proučevanja v današnji teoriji grafov.

Zato je smiselno iskati različne vrste grafov, ki so Hamiltonovi. Na primer: očitno je, da so cikli C_n Hamiltonovi grafi za vse vrednosti n. Tudi K_n so Hamiltonovi, če je $n \geq 3$; če vozlišča označimo z 1, 2, ..., n, potem je Hamiltonov cikel $123 \ldots n1$.

Če vzamemo Hamiltonov graf in mu dodamo nekaj povezav, potem je dobljeni graf še vedno Hamiltonov, saj lahko uporabimo isti Hamiltonov cikel kot prej. Torej je za grafe z veliko povezavami bolj verjetno, da bodo Hamiltonovi, kot za grafe z manj povezavami. To lahko povemo bolj natančno na različne načine. Dva najpomembnejša sta naslednja zadostna pogoja G.A. Diraca in O. Oreja, objavljena leta 1952 in 1960.

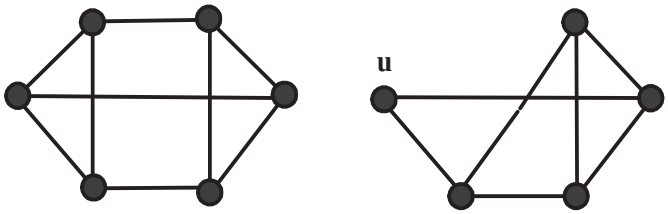
IZREK 2.3. (DIRACOV IZREK). *Naj bo G enostaven graf z n vozlišči in $n \geq 3$. Če je $\mathrm{st}(v) \geq \frac{1}{2}n$ za vsako vozlišče v, potem je G Hamiltonov.*

IZREK 2.4. (OREJEV IZREK). *Naj bo G enostaven graf z n vozlišči in $n \geq 3$. Če je*

$$\mathrm{st}(v) + \mathrm{st}(w) \geq n$$

za vsak par nesosednjih vozlišč v in w, potem je G Hamiltonov.

Uporabo izrekov ponazorimo na grafih na sliki



V grafu (a) je $n = 6$ in $\mathrm{st}(v) = 3$ za vsako vozlišče v, zato je ta graf Hamiltonov po Diracovem izreku.

V grafu (b) je $n = 5$, toda $\mathrm{st}(u) = 2$, zato Diracovega izreka ne moremo uporabiti. Vendar pa je

Springer, Berlin 2000] str. 235, [E.Lawler, Combinatorial optimization, Networks and Matroids, Holt, Rinehart and Winston, New York 1976] str.217, za podrobnosti algoritmov s časovno zahtevnostjo $O(n^3)$ in $O(mn + n^2 \log n)$.

$\mathrm{st}(v) + \mathrm{st}(w) \geq 5$ za vse pare nesosednjih vozlišč v in w (pravzaprav za vse pare vozlišč v in w), zato je graf Hamiltonov po Orejevem izreku.

Če je $\mathrm{st}(v) \geq \frac{1}{2}n$ za vsako vozlišče v, potem velja $\mathrm{st}(v) + \mathrm{st}(w) \geq n$ za vsak par vozlišč v in w. Torej Diracov izrek sledi iz Orejevega izreka, zato bomo dokazali samo Orejev izrek (glej nalogo 2.12.).

Poglejmo si še potreben pogoj, ki ga ni težko dokazati (glej nalogo 2.13.).

IZREK 2.5. *Če iz povezanega grafa G odstranimo k vozlišč in tako dobimo graf z več kot k komponentami, potem graf G ne vsebuje Hamiltonovega cikla; če je komponent več kot $k + 1$, potem graf G ne vsebuje niti Hamiltonove poti.*

2.5. Problemi hamiltonskega tipa

Polhamiltonovi grafi

Podobno kot pri Eulerjevih grafih je tudi tu nekaj različic gornjih idej in rezultatov. Na primer: **polhamiltonovi** grafi so grafi, v katerih obstaja pot, ki obišče vsako vozlišče natanko enkrat; tako pot običajno imenujemo **Hamiltonova pot**.

Primer zanimive naloge iz področja zabavne matematike, ki jo prevedemo na nalogo hamiltonskega tipa, je problem šahovskega konjička (glej nalogo 2.14.).

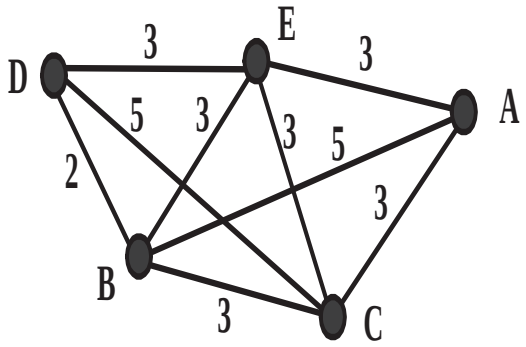
Problem trgovskega potnika

Problem trgovskega potnika je verjetno najbolj raziskovan problem kombinatorične optimizacije[6]. Razlog za tako veliko zanimanje seveda ni samo v tem, da bi bilo računanje optimalnih poti tako pomembno za trgovske potnike. Zanimivo in morda presenetljivo je, da lahko enakovredne naloge najdemo v mnogih drugih, včasih zelo različnih kontekstih. Nekaj preprostih primerov bomo srečali v nadaljevanju.

Trgovski potnik želi obiskati nekaj mest in se vrniti v svoje mesto, tako da bo vsako mesto obiskal natanko enkrat, skupna prevožena razdalja pa bo najkrajša možna. Katero pot naj izbere, če pozna vse razdalje med mesti?

Načeloma lahko problem rešimo s pregledom vseh možnih ciklov in izbiro tistega z najkrajšo skupno razdaljo. Če imamo na primer pet mest A, B, C, D in E in če so razdalje med njimi take kot na sliki spodaj, potem mora trgovski potnik iz A obiskati mesta v zaporedju $ACBDEA$ (ali v obratnem vrstnem redu $AEDBCA$), s skupno razdaljo 14.

[6]Naloga trgovskega potnika (angl. traveling salesman problem, TSP) je tipičen predstavnik težkih problemov kombinatorične optimizacije, zato je pogosto predmet teoretičnih in experimentalnih raziskav. Število člankov in knjig o tem problemu je ogromno, glej npr. [E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan in D.B.Shmoys, The Traveling Salesman Problem, John Wiley & Sons, New York, 1985]. Že nekaj časa je na internetu na voljo knjižnica testnih nalog TSPLIB, glej na primer: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/



Če uteži v grafu ne pomenijo razdalj, ampak čas ali stroške, potrebne za potovanje, potem je rešitev problema trgovskega potnika cikel z najmanjšim časom ali najmanjšimi stroški.

Ko pa število mest povečamo, hitro zabredemo v težave, saj ni znan noben algoritem, ki bi dal enostavno in hitro rešitev problema. Iskanje optimalne rešitve zahteva praktično pregled vseh možnih poti in izbiro najkrajše, saj problem spada med NP-težke probleme (glej poglavje 5). To je še mogoče za primere z desetimi mesti, saj je takrat število vseh možnih ciklov največ $9! = 362880$, in računalnik, ki pregleda milijon poti na sekundo, bo našel najboljšo v približno 0.36 sekunde. Po drugi strani pa za 20 mest obstaja že okoli 1.22×10^{17} možnih poti in računalnik bi pri isti hitrosti pregledovanja potreboval skoraj 4000 let! Zato se moramo običajno zadovoljiti s približnimi rešitvami. To nas pripelje do obravnave algoritmov, s katerimi izračunamo približne rešitve in do metod za ocenjevanje vrednosti optimalne rešitve. (O tem nekaj več v poglavju 5.) Včasih je mogoče nalogo, ki je v splošni formulaciji težka, v posebnih primerih učinkovito rešiti. Nekaj primerov si bomo ogledali v nadaljevanju.

Vrnimo se k formulaciji naloge. Poiskati želimo cikel z najmanjšo utežjo, ki obišče vsako vozlišče – z drugimi besedami, Hamiltonov cikel z najmanjšo utežjo. Problem trgovskega potnika lahko torej v jeziku teorije grafov povemo takole:

Problem trgovskega potnika I: Uteženemu grafu poišči Hamiltonov cikel z najmanjšo utežjo.

V tej formulaciji problema smo privzeli, da je dani graf Hamiltonov. Nalogo lahko zapišemo tudi tako, da ta privzetek ni potreben.

Problem trgovskega potnika II: Uteženemu polnemu grafu poišči Hamiltonov cikel z najmanjšo utežjo.

V drugi formulaciji vsaka permutacija vozlišč grafa določa Hamiltonov cikel[7]. Ta formulacija je torej v nekem smislu enostavnejša, saj nam ni treba iskati Hamiltonovih ciklov grafa, kar je (kot bomo videli kasneje) že samo zase težek problem.

Prepričajmo se, da sta obe formulaciji problema enakovredni. Druga formulacija je poseben primer prve, saj se razlikuje samo v dodatni zahtevi, da je graf poln. Recimo, da smo dobili nalogo

[7]**Permutacija** (reda n) je bijektivna preslikava množice $\{1, 2, \ldots, n\}$ nase. Eden od standardnih kombinatoričnih zapisov permutacije je (3 1 2 4), ki pomeni, da se preslika 3 v 1, 1 v 2, 2 v 4 in 4 v 3. Permutacija (3 1 2 4) nam da Hamiltonov cikel 3 1 2 4 (3).

trgovskega potnika na poljubnem grafu, ki ni poln (naloga v prvi formulaciji). Seštejmo uteži na vseh povezavah in vsoto označimo z S. Dodajmo grafu manjkajoče povezave, jih utežimo z utežjo S, in predpostavimo, da smo dobili optimalno rešitev nove naloge (na polnem grafu, torej v drugi formulaciji). Zdaj sklepamo takole: Če ima optimalna rešitev dolžino (vsoto uteži) večjo kot S, potem smo gotovo morali uporabiti vsaj eno dodano povezavo, prvotni graf torej ni bil Hamiltonov in naloga sploh ni imela rešitve. Če je optimalna rešitev manjša od S, potem na njej ni nobene od dodanih povezav in rešitev je tudi (optimalna) rešitev prvotne naloge. Zato običajno, ko govorimo o problemu trgovskega potnika, mislimo na drugo formulacijo.

Zanimivo je primerjati ta problem s kitajskim problemom poštarja, ki ga lahko razumemo kot Eulerjev analogon tega problema. Pri *kitajskem problemu poštarja* ni nobenih težav, če je graf Eulerjev – enostavno poiščemo Eulerjev obhod s Fleuryjevim algoritmom in katerakoli dobljena rešitev je že rešitev problema. Tudi če graf ni Eulerjev, obstaja standardni algoritem, s katerim lahko poiščemo primeren najkrajši sklenjen sprehod. Pri *problemu trgovskega potnika* privzamemo, da je graf Hamiltonov, vendar lahko obstaja več Hamiltonovih ciklov z različnimi utežmi. Zato potrebujemo algoritem, s katerim lahko poiščemo Hamiltonov cikel z najmanjšo utežjo. Žal pa ne poznamo nobenega takega algoritma. Če število mest ni zelo majhno, je najboljše, kar lahko naredimo, da poskusimo dobiti približne rešitve.

Naloge sorodne problemu trgovskega potnika

Problem trgovskega potnika je pomemben v praksi, saj se lahko pojavi v mnogih preoblekah. Tak primer je problem razporejanja poslov, ki ga definiramo takole:

Problem razporejanja opravil. Nekaj neodvisnih opravil je treba izvesti na enem računalniku. Opravila so zapleteni računalniški programi, zato je treba pred vsakim opravilom nastaviti računalnik posebej za to opravilo. Na začetku je računalnik pripravljen za izvajanje enega od opravil in na koncu mora biti nastavljen v enako stanje kot na začetku. Če so stroški za nastavitev odvisni od zaključenega in začetega opravila, kako je treba razporediti izvajanje opravil, da bodo skupni stroški najmanjši? s Povezavo med problemoma lahko vidimo, če si problem predstavimo grafično. Pri *problemu trgovskega potnika* narišemo utežen graf, v katerem vozlišča ustrezajo mestom, ki jih mora trgovski potnik obiskati, povezave ustrezajo cestam med mesti, uteži na povezavah pa ustrezajo razdaljam med mesti. Pri *problemu razporeditve opravil* narišemo utežen poln graf, v katerem vozlišča ustrezajo opravilom, povezave povezujejo opravila, uteži pa ustrezajo stroškom nastavitve računalnika pri ustreznem paru opravil.

Poglejmo si še en primer praktične naloge, ki jo lahko predstavimo kot problem trgovskega potnika.

Problem rezanja tapet. Recimo, da je treba iz dolgega traku tapete z vzorcem izrezati n kosov. Vzorec tapete je periodičen s periodo p (mm,cm,dm,...). Vemo, da moramo začetek i-tega kosa odrezati na mestu s_i (s_i je razdalja od začetka periode vzorca) in če je dolžina i-tega kosa enaka l_i, lahko izračunamo, da bo konec na mestu $f_i = s_i + l_i (\mathrm{mod}\ p)$. Če odrežemo kos j takoj za kosom i, bomo zavrgli

$$c_{ij} = \begin{cases} s_j - f_i, & \text{če je } f_i \leq s_j \\ p + s_j - f_i, & \text{sicer} \end{cases}$$

Če privzamemo, da vedno dobimo tapeto, ki je odrezana na mestu f_0 in da se spodobi oddati zavoj odrezan enako, si lahko mislimo, da imamo dodaten kos „0“, z $f_0 = s_0$. Hitro se prepričamo, da je naloga ekvivalentna nalogi trgovskega potnika na omrežju, katerega vozlišča ustrezajo kosom

0,1,...,n, uteži na povezavah pa dolžini odpadkov c_{ij}. Optimalna rešitev problema trgovskega potnika na tem omrežju nam da zaporedje rezanja z minimalnim odpadom.[8]

Za konec omenimo še manj znan primer posplošitve naloge.[9]

Problem Verjetnostni trgovski potnik (angl. probabilistic traveling salesman problem, PTSP) je posplošitev klasičnega problema trgovskega potnika, v katerem je množica vozlišč, ki naj jih trgovski potnik obišče, slučajna spremenljivka[10].

Za motivacijo si oglejmo naslednji primer. Predpostavimo, da podjetje želi načrtati obhod svojih n strank in pri tem želi optimirati samo potovalne stroške (ali prevoženo razdaljo). Če je vsako stranko treba obiskati vsak dan je to običajni TSP. Predpostavimo, da bo obhod uporabljan daljši čas in da se v splošnem (pod)množica strank, ki jih je treba tisti dan obiskati, iz dneva v dan spreminja. Razlogi za zadnjo predpostavko so lahko različni, na primer:

- podjetje ne ve, katere stranke je treba obiskati. Ta informacija je dostopna šele tik pred obhodom ali pa celo med njim.
- podjetje ne želi optimirati obhoda vsak dan, ker je to predrago ali pa zato, ker želi, da isti voznik obišče iste postaje. Zadnja zahteva zagotavlja večjo stalnost usluge, pokriva pa tudi primer, ko je voznikovo znanje (izkušnje) pomembno za učinkovito opravljeno uslugo.

Znano je:

- TSP je očitno posebni primer PTSP. (Če so vse verjetnosti vozlišč enake 1, je rešitev PTSP ista kot rešitev TSP.)
- obstajajo primeri, za katere je optimalna TSP rešitev poljubno slaba rešitev za PTSP.[11]

To pomeni, da je naloga vsaj tako težka kot problem trgovskega potnika in da algoritmi in ocene za trgovskega potnika morda niso uporabni za verjetnostnega trgovskega potnika[12].

2.6. Naloge z rešitvami

2.1. Kateri od naslednjih grafov so Eulerjevi? (a) cikli C_n, (b) polni dvodelni grafi $K_{r,s}$, (c) hiperkocke Q_k, (d) polni grafi K_n.

[8] Ta poseben primer je mogoče učinkovito rešiti. Glej [E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan in D.B.Shmoys, The Traveling Salesman Problem, John Wiley & Sons, New York, 1985], stran 117.

[9] Nadalnje aplikacije, posplošitve in variante: glej npr. 2. poglavje [E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan in D.B.Shmoys, The Traveling Salesman Problem, John Wiley & Sons, New York, 1985].

[10] **Slučajna spremenljivka** je spremenljivka, pri kateri je ob zalogi vrednosti dan tudi porazdelitveni zakon. Za spremenljivke s končno zalogo vrednosti je porazdelitveni zakon kar prireditev verjetnosti vsaki od vrednosti iz zaloge vrednosti, torej vsaki možni vrednosti spremenljivke $x_i \in \mathcal{R}_x$ priredimo verjetnost p_i. ($0 \leq p_i \leq 1$, $\sum_i p_i = 1$.) Glej npr. [Jamnik, Verjetnostni račun, Mladinska knjiga, Ljubljana 1971].

[11] [P.Jaillet, A Priori Solution of a Traveling Salesman Problem in which a Random Subset of the Customers are Visited, Operations Research 36 (1988) 929-936]

[12] Glej tudi [J.Žerovnik, Verjetnost v kombinatorični optimizaciji, disertacija, Univerza v Ljubljani, 1990] in dodatne reference tam.

2.2. Poišči Eulerjev obhod grafov: (a) K_3, K_5 in K_7, (b) K_9, (c) K_{2n+1}!

2.3. S koliko potezami lahko narišemo graf s slike 2.5 (na strani 41)?

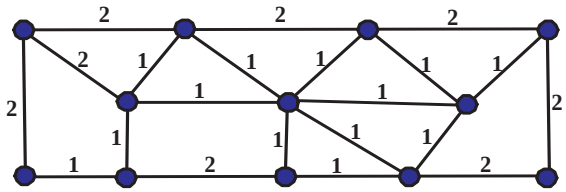
2.4. Dokaži trditev: Če ima v grafu G vsaka točka stopnjo večjo ali enako 2, potem v grafu obstaja cikel.

2.5. Dokaži trditev: Če ima v grafu G vsako vozlišče sodo stopnjo, potem lahko povezave grafa razbijemo na unijo disjunktnih ciklov – torej noben par ciklov nima skupne povezave.

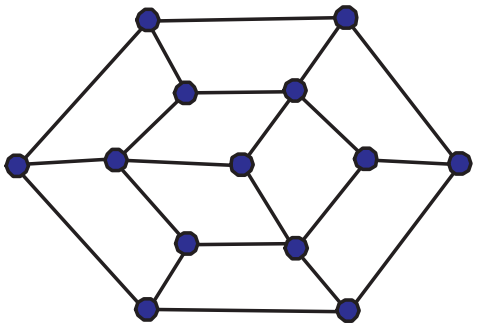
2.6. Dokaži Izrek 2.2.: Naj bo G povezan graf. Potem je G poleulerjev natanko tedaj, ko ima G natanko dve vozlišči lihe stopnje.

2.7. S koliko potezami lahko narišemo graf z $2k$ točkami lihe stopnje?

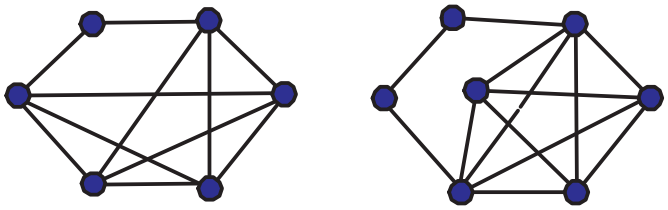
2.8. Reši nalogo kitajskega poštarja na grafu



2.9. Dokaži, da noben dvodelen graf z lihim številom vozlišč ne more biti Hamiltonov. Uporabi ta rezultat in pokaži, da naslednji graf ni Hamiltonov.



2.10. Preveri, ali pogoji Diracovega in Orejevega izreka veljajo za naslednja grafa:



(a) (b)

2.11. Poišči primere za:

(a) Hamiltonov graf, ki ne zadošča pogojem Orejevega izreka;
(b) Graf z n točkami, ki ni Hamiltonov in v katerem je $\mathrm{st}(v) \geq \frac{1}{2}(n-1)$ za vsako točko v.

2.12. Dokaži Orejev izrek.

2.13. Dokaži izrek 2.5.: Če iz povezanega grafa G odstranimo k vozlišč in tako dobimo graf z več kot k komponentami, potem graf G ne vsebuje Hamiltonovega cikla; če je komponent več kot $k+1$, potem graf G ne vsebuje niti Hamiltonove poti.

2.14. Naloga o požrešnem šahovskem konjičku sprašuje, ali na šahovnici velikosti $m \times n$ obstaja tako zaporedje skokov šahovskega konjička, s katerim obišče vsako polje šahovnice natanko enkrat. Nalogo lahko predstavimo tudi takole: definiramo graf, katerega vozlišča so polja šahovnice in povežemo dve polji, če lahko konjiček skoči z enega na drugo polje. Zanimajo nas sprehodi po tem grafu. Naloga ima več verzij: običajno zahtevamo, da se konjiček po obisku vseh polj (lahko) vrne na začetno polje. V tem primeru sprašujemo po Hamiltonovem ciklu grafa. Če zahtevamo samo obisk vseh polj (natanko po enkrat) potem iščemo Hamiltonovo pot. Če predpišemo tudi začetno in(/ali) končno vozlišče, potem iščemo Hamiltonovo pot z dodatnim(a) pogojema.

(a) Ali obstajata konjičkov sprehod in obhod na šahovnici 3×4?

(b) Ali obstaja konjičkov sprehod na šahovnici 3×5?

(c) Dokaži, da na šahovnicah 5×5 in 7×7 ni nobenega konjičkovega obhoda.

(d) Poišči konjičkov obhod na šahovnici 8×8.

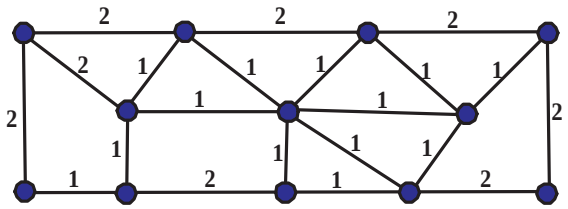
(e) Dokaži, da na šahovnici 8×8 ni konjičkovega sprehoda z začetkom v a1 in koncem v h8.

2.15. Za nalogo trgovskega potnika, dano z razdaljami (London – Birmingham = 116, London – Leeds=193, London – Manchester = 184, Birmingham – Manchester = 81, Birmingham – Leeds = 110, Leeds – Manchester = 41):

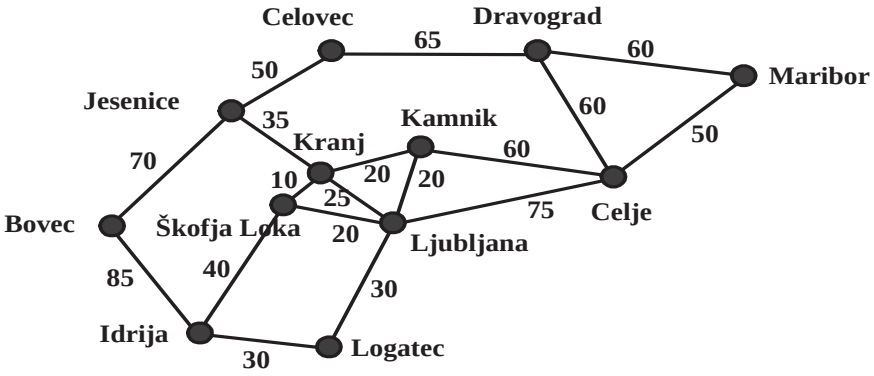
(a) Nariši ustrezni uteženi graf (omrežje)!

(b) Poišči optimalno rešitev!

2.16. Poišči Hamiltonov cikel z minimalno dolžino na grafu



2.17. Poišči rešitev za nalogo trgovskega potnika na sliki



2.18. Stroj lahko pripravimo za izdelovanje petih vrst izdelkov: I_1,I_2,I_3,I_4 in I_5. Označimo z S_i stanje stroja, ko je pripravljen za izdelovanje izdelka I_i. Ko stroj ne obratuje mora biti v mirujočem stanju S_0. Sprememba nastavitve stroja iz stanja S_i v stanje S_j traja $t(i,j)$, $i = 0,1,\ldots,5$, $j = 0,1,\ldots,5$ časa. Tipično dnevno naročilo je seznam K_1, K_2, K_3, K_4, K_5, kjer K_i pomeni število naročenih izdelkov I_i. Želimo čim bolj skrajšati čas obratovanja.

(a) Ali lahko problem predstaviš kot nalogo na usmerjenem omrežju (usmerjenem grafu z utežmi na povezavah? Če je odgovor da, nariši omrežje.

(b) Poišči optimalno rešitev pri podatkih (stolpci in vrstice v matriki so indeksirani od 0 do 5.)

$$t = \begin{bmatrix} 0 & 1 & 1 & 1 & 2 & 1 \\ 1 & 0 & 1 & 1 & 2 & 1 \\ 2 & 2 & 0 & 2 & 2 & 2 \\ 3 & 1 & 3 & 0 & 2 & 1 \\ 2 & 2 & 3 & 1 & 0 & 1 \\ 1 & 1 & 3 & 1 & 2 & 0 \end{bmatrix}$$

2.19. V kakšnem vrstnem redu moramo izrezati štiri kose tapete, da bo odpadek minimalen, če vemo, da ima vzorec tapete periodo 1 in da morajo biti začetki kosov odrezani na razdaljah od začetka vzorca $s_1 = 0.1$, $s_2 = 0.8$, $s_3 = 0.6$, $s_4 = 0.4$ in konci kosov na $f_1 = 0.8$, $f_2 = 0.7$, $f_3 = 0.7$, $f_4 = 0.2$.

(a) Izračunaj stroške c_{ij}!

(b) Nariši ustrezni uteženi graf (omrežje)!

(c) Poišči optimalno rešitev!

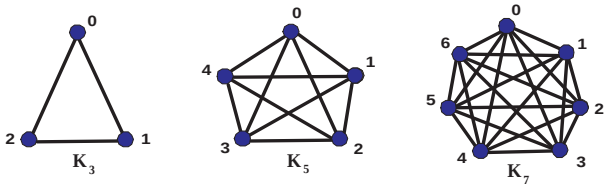
2.20. Požrešna metoda *najbližji sosed* (NN, angl. nearest neighbor) za nalogo trgovskega potnika je definirana takole: začni v poljubnem vozlišču, v_1, dokler niso obiskana vsa vozlišča, izberi najbližjega še neobiskanega soseda vozlišča v_i in ga označi z v_{i+1}. Nazadnje poveži v_n z v_1, da dobiš Hamiltonov obhod.

Poišči primere, ki dokazujejo, da je lahko dobljena rešitev poljubno slaba.

Rešitve

2.1. Po izreku 2.1. je povezan graf Eulerjev natanko tedaj, ko so vsa vozlišča sode stopnje. Zato: (a) Vsi cikli C_n so Eulerjevi. (b) Polni dvodelni grafi $K_{r,s}$ so Eulerjevi natanko tedaj, ko sta r in s sodi števili. (c) Hiperkocke Q_k so Eulerjevi grafi za sode k. (d) Poln graf K_n je $n-1$ regularen, torej je Eulerjev samo za lihe vrednosti n.

2.2. Vemo, da je poln graf K_n je Eulerjev za lihe vrednosti n po izreku 2.1.



(a) Eulerjevi obhodi so: 0 1 2 0 na K_3; 0 1 2 3 4 0 2 4 1 3 0 na K_5; 0 1 2 3 4 5 6 0 2 4 6 1 3 5 0 3 6 2 5 1 4 0 na K_7; (Obstajajo seveda tudi drugi obhodi.)

(b) Konstrukcija iz točke (a) na grafu K_9 ne deluje. (Težava nastopi pri ciklu 0 3 6 ker 3 deli 9. To lahko popravimo tako, da vsakega od trikotnikov 0 3 6 0, 1 4 7 1, 2 5 8 2) obhodimo takoj, ko je mogoče. Eulerjev obhod je na primer:

0 3 6 0 1 4 7 1 2 5 8 2 3 4 5 6 7 8 0 2 4 6 8 1 3 5 7 0 4 8 3 7 2 6 1 5.

(c) Konstrukcije iz točke (a) ne moremo direktno posplošiti, saj že na na grafu K_9 ne deluje. Trik, ki smo ga uporabili v točki (b), pa lahko uporabimo, vendar je zapis rešitve sorazmerno zapleten. (Nasvet: sledimo ideji dokaza izreka 2.1.)[13]

2.3. S tremi potezami. (Glej tudi nalogo 2.7.)

2.4. Naj bo G graf z m povezavami, v katerem ima vsako vozlišče stopnjo večjo ali enako 2. Naj bodo $v_0, \ldots, v_t$ vozlišča poti P z največjo dolžino v G. Ker ima vozlišče v_t sodo stopnjo, je povezano še vsaj z enim vozliščem grafa v, ki ni v_{t-1}. Ker je P pot največje dolžine, mora biti v eno od vozlišč $v_0, \ldots, v_{t-2}$[14] – recimo $v = v_i$. Potem je $v_i, v_{i+1}, \ldots, v_t, v_i$ cikel v grafu G, kar je bilo treba dokazati.

2.5. Trditev lahko dokažemo z metodo *matematične indukcije.* Najprej bomo videli, da trditev velja za grafe z $m = 0$ povezavami. Nato bomo dokazali, da lahko dokažemo trditev za grafe z m povezavami, če privzamemo, da velja trditev za grafe z manj kot m povezavami. Odtod sklepamo, da rezultat velja za grafe s katerimkoli danim številom točk.

Dokaz trditve. Izrek očitno velja, če graf nima nobene povezave.

Predpostavimo zdaj, da izrek velja za grafe z manj kot m povezavami – z drugimi besedami, vsak graf z vozlišči sode stopnje s k povezavami lahko razbijemo na unijo disjunktnih ciklov, če je le $k < m$. Izpeljati moramo trditev za grafe z vozlišči sode stopnje z m povezavami.

Naj bo G graf z m povezavami, v katerem ima vsako vozlišče sodo stopnjo. Vse stopnje vozlišč grafa so torej večje ali enake 2, zato v grafu obstaja cikel, označimo ga s C (glej nalogo 2.4.). Če iz G odstranimo cikel C, dobimo graf G_1 z manj kot m povezavami, v katerem ima vsako vozlišče (spet) sodo stopnjo. Po naši predpostavki lahko povezave grafa G_1 razbijemo na unijo disjunktnih ciklov; skupaj s ciklom C pa dobimo razbitje povezav grafa G na unijo disjunktnih ciklov, kar je bilo treba dokazati.

2.6. Naj bo G poleulerjev graf, v in u pa začetno in končno vozlišče odprtega sprehoda, na katerem so vse povezave grafa. Če dodamo povezavo e med vozliščema v in u, dobimo Eulerjev graf, v katerem morajo po izreku 2.1. biti vsa vozlišča sode stopnje. Če povezavo e odstranimo, vidimo, da sta v G edini vozlišči lihe stopnje ravno v in u.

Zdaj pa recimo, da ima graf G natanko dve vozlišči lihe stopnje, imenujmo ju v in w. Če dodamo povezavo e med vozliščema v in w, dobimo povezan graf, v katerem imajo vsa vozlišča sodo stopnjo. Po izreku 2.1. je dobljeni graf Eulerjev, v njem torej obstaja Eulerjev obhod. Če v dobljenem Eulerjevem obhodu odstranimo povezavo e, dobimo odprt sprehod, na katerem so vse povezave grafa G. Graf G je torej poleulerjev.

[13] Leta 1809 je L. Poinsot pokazal, da je mogoče diagrame z n paroma povezanimi vozlišči narisati z eno potezo samo, če je n liho število in zapisal konstrukcijo za iskanje Eulerjevega obhoda.

[14] Sicer bi dobili novo pot $v_0, \ldots, v_{t-2}, v$, ki bi bila daljša od poti P.

2.7. S k potezami.

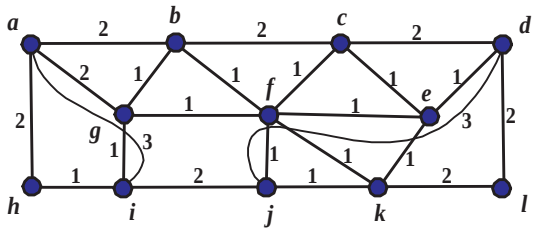
Utemeljimo odgovor: Najprej preverimo, da je k potez vedno dovolj. S k dodatnimi povezavami kakorkoli povežimo po dve vozlišči lihe stopnje v grafu. Tako dobimo Eulerjev graf. Po odstranitvi dodanih k povezav Eulerjev obhod grafa razpade na natanko k sprehodov, na katerih so vse povezave prvotnega grafa.

Povezav grafa ne moremo narisati z manj kot k potezami. Recimo, da bi to bilo mogoče z $\ell < k$ potezami. Konstruirajmo novi graf takole: Končno vozlišče prvega sprehoda (prve poteze) povežimo s prvim vozliščem drugega, in tako dalje, končno vozlišče ℓ-tega sprehoda povežimo s prvim vozliščem prvega. dobili nov graf, ki ima ℓ povezav več kot prvotni in je Eulerjev. Ker je bilo $2k$ vozlišč lihe stopnje, z ℓ povezavami pa smo popravili stopnje kvečjemu $2\ell > 2k$ vozliščem, smo prišli v protislovje z izrekom 2.1.Z manj kot k potezami torej ne gre.

2.8. V grafu so štiri vozlišča lihe stopnje: a, i, j in d. Dolžine najkrajših poti med njimi so: $d(a,i) = 3$, $d(a,j) = 4$, $d(a,d) = 5$, $d(i,j) = 2$, $d(i,d) = 4$, $d(j,d) = 3$. Izbrati moramo dve poti med vozlišči a, i, j in d z minimalno skupno dolžino. (Ali, drugače rečeno, na grafu z vozlišči a, i, j in d in povezavami z utežmi enakimi razdaljam v originalnem grafu moramo poiskati minimalno prirejanje.) Izberemo pot med a in i ter pot med j in d.

Optimalen obhod bo imel strošek enak vsoti vseh uteži povezav grafa plus dolžini dodanih poti. Strošek bo torej enak 33.

Rešitev lahko poiščemo s Fleuryjevim algoritmom na grafu, ki smo mu dodali dve povezavi (ai z utežjo 3 in jd z utežjo 3).



Ena od rešitev je $abcdecfbgahigfjkfekldjia$. Poštarjev obhod na originalnem grafu dobimo tako, da dodani povezavi zamenjamo z najkrajšima potema (na primer z agi in $jfed$), kar nam da poštarjev obhod $abcdecfbgahigfjkfekldefjiga$ na originalnem grafu.

2.9. Vozlišča dvodelnega grafa lahko razdelimo na dve množici A in B, tako da ima vsaka povezava eno krajišče v A in eno krajišče v B. Vsak Hamiltonov cikel mora izmenično obiskovati ti dve množici in končati v množici, v kateri je njegov začetek. Torej morata v dvodelnem Hamiltonovem grafu imeti množici A in B enako mnogo elementov. To ni mogoče, če je število vozlišč grafa liho.

Graf na sliki je dvodelen z lihim številom vozlišč, zato ne more biti Hamiltonov.

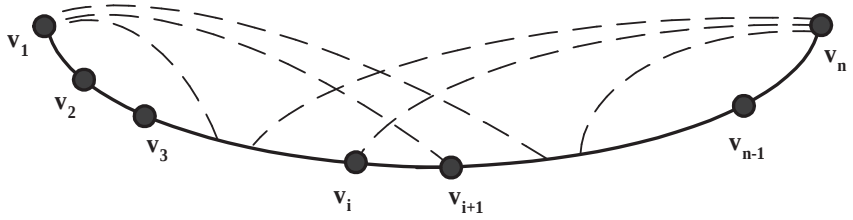
2.10. (a) Graf ne ustreza pogojem Diracovega izreka, ker ima $n = 6$ vozlišč in vozlišče stopnje 2, $2 < \frac{6}{3}$.

Graf ustreza pogojem Orejevega izreka, ker so vsa druga vozlišča stopnje 5, in $5 + 2 > 6$. (b) Graf ne ustreza pogojem Orejevega izreka (in seveda tudi ne ustreza pogojem Diracovega izreka). Vendar je graf kljub temu Hamiltonov, saj izreka govorita o zadostnih, ne pa tudi o potrebnih pogojih za hamiltonskost.

2.11.

(a) Katerikoli cikel C_n za $n \geq 5$. Glej tudi graf (b) iz naloge 2.10..
(b) Polni dvodelni graf $K_{r,s}$, kjer je $r = \frac{1}{2}(n-1)$, $s = \frac{1}{2}(n+1)$ in je n liho število. (Če je n sodo število, potem je $\mathrm{st}(v) \geq \frac{1}{2}n$ za vsako točko v in graf je Hamiltonov po Diracovem izreku[15].)

2.12. Izrek bomo dokazali s protislovjem. Recimo, da obstaja graf G, ki *ni Hamiltonov* in v katerem je $\mathrm{st}(v) + \mathrm{st}(w) \geq n$ za vsak par nesosednjih vozlišč v in w. Lahko privzamemo, da je G „najmanjši“ nehamiltonov graf v naslednjem pomenu: če grafu G dodamo katerokoli povezavo, potem dobimo Hamiltonov graf. (Če to za graf G ne bi veljalo, bi lahko dodali tako povezavo, da graf ne bi postal Hamiltonov, in obravnavali dobljeni graf.) V takem grafu obstaja pot $v_1v_2v_3 \ldots v_n$, na kateri so vsa vozlišča grafa, vozlišči v_1 in v_n pa nista sosednji, kot kaže naslednja slika (če dodamo povezavo v_nv_1, dobimo Hamiltonov cikel):



Ker v_1 in v_n nista sosednji, mora veljati:

$$\mathrm{st}(v_1) + \mathrm{st}(v_n) \geq n$$

ali, če označimo $\mathrm{st}(v_1) = r$,

$$\mathrm{st}(v_n) \geq n - \mathrm{st}(v_1) = n - r \ .$$

Naj bo S_1 množica vozlišč, ki so sosednja z v_n in S_2 množica vozlišč, ki so neposredni predhodniki sosedov vozlišča v_1 na poti. Z drugimi besedami, če je vozlišče v_k povezano z v_1, potem je v_{k-1} v množici S_2.

Množica S_1 ima vsaj $n - r$ elementov, množica S_2 pa r elementov. Ker v_n ni element nobene od množic S_1 in S_2, ima unija $S_1 \cup S_2$ kvečjemu $n - 1$ elementov. Zato mora biti presek $S = S_1 \cap S_2$ neprazen. Torej mora biti v S vsaj eno vozlišče v_i, ki je povezano z v_n in zato obstajata povezavi med v_1 in v_{i+1} in med v_i in v_n, kot je narisano na sliki zgoraj. Potem pa imamo v grafu Hamiltonov cikel:

$$v_1v_2 \ldots v_{i-1}v_iv_nv_{n-1} \ldots v_{i+1}v_1,$$

kar je v nasprotju s predpostavko, da graf G ni Hamiltonov. Iz tega protislovja sledi veljavnost izreka.

[15] Če je n sodo, potem iz $\mathrm{st}(v) \geq \frac{1}{2}(n-1)$ sledi $\mathrm{st}(v) \geq \frac{1}{2}n$, saj mora biti $\mathrm{st}(v)$ celo število.

2.13. Recimo, da v grafu obstaja Hamiltonov cikel. Če iz grafa odstranimo k vozlišč, cikel razpade na največ k (pod)poti. Vsaka od teh (pod)poti je v neki komponenti, na katere razpade graf G. Ker je bil cikel Hamiltonov, vsaka komponenta vsebuje vsaj eno od (pod)poti. Torej Hamiltonov graf ne more razpasti na več kot k komponent.

Podobno dokažemo, da polhamiltonov graf z odstranitvijo $k+1$ vozlišč lahko razpade na največ $k+1$ komponent.

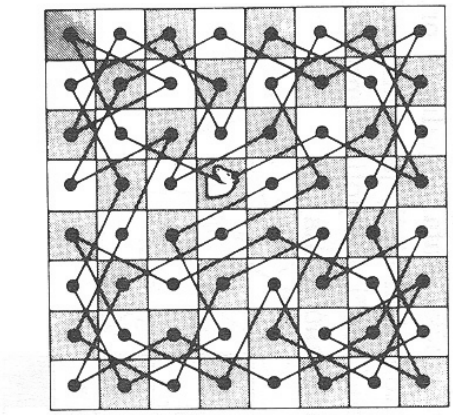
2.14. Označimo polja šahovnice (kot je navada pri šahu) z a1, b1,c1,d1,. . . ,h1,. . . , a2, b2,c2,d2, . . . , h2, . . . , h8 in uporabimo analogne oznake tudi za šahovnice drugih dimenzij. (Nariši šahovnici 3×4 in 3×5 in pripadajoča grafa!)

(a) Na šahovnici 3×4 je sprehod na primer: a1 c2 a3 b1 d2 b3 c1 d3 b2 d1 c3 a2. Obhod na tej šahovnici ni mogoč. V grafu je 6 vozlišč stopnje 2. Vsak Hamiltonov cikel mora obiskati vsa vozlišča, zato morajo vse povezave s krajišči v teh vozliščih biti del vsakega Hamiltonovega cikla. Vendar v našem grafu povezave s krajišči v vozliščih stopnje 2 sestavljajo dva cikla dolžine 6, ki ju seveda ne moremo dopolniti do Hamiltonovega cikla. Torej konjičkov obhod na šahovnici 3×4 ne obstaja.

(b) Če iz grafa odstranimo vozlišča (polja) b1, b3, c2, d1 in d3, graf razpade na 7 komponent. Torej po izreku 2.5. ni polhamiltonov.

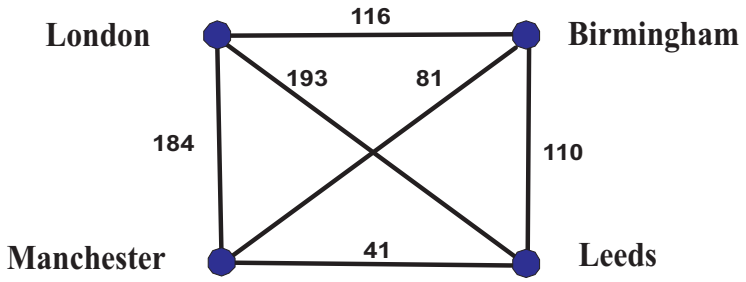
(c) Ustrezni graf je dvodelen, ker skakač vedno skoči na polje druge barve. Razbitje grafa seveda ustreza delitvi šahovnice na bela in črna polja. Odgovor zdaj sledi iz ugotovitve naloge 2.9.. (Vse kar je treba povedati, je: ker skakač vedno skoči iz črnega na belo polje ali *obratno*, mora biti število črnih in belih polj enako. To pa seveda ni mogoče na katerikoli šahovnici, ki ima liho število polj.)

(d)



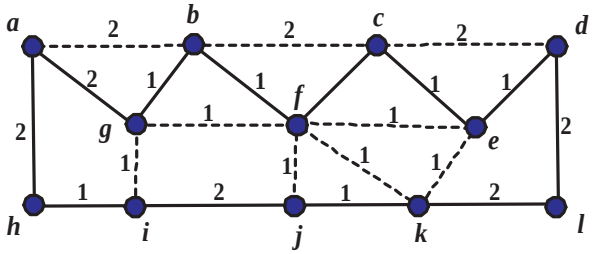
(e) V točki (c) smo videli, da je graf dvodelen. Vozlišči a1 in h8 sta v isti množici razbitja (polji sta iste barve), vsak sprehod med njima je torej sode dolžine. Sprehod med a1 in h8, ki bi obiskal vsa vozlišča pa bi imel dolžino 63, torej liho dolžino. Protislovje, torej takega sprehoda ni.

2.15. (a)



(b) Leeds – Manchester – Birmingham – London – Leeds, dolžina 431.

2.16. Na sliki je Hamiltonov cikel dolžine 17

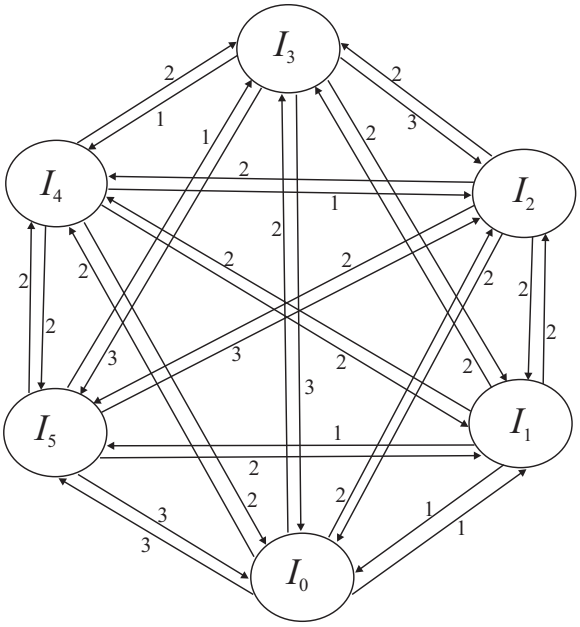


Preverimo optimalnost. Poskusimo najprej metodo z uporabo minimalnega vpetega drevesa. (Glej stran 65.) Vidimo, da sta vozlišči a in ℓ krajišči povezav, ki imajo vse dolžino 2. Za obisk vozlišč a in ℓ bomo morali uporabiti po dve povezavi dolžine 2. Na grafu brez a in ℓ za Hamiltonov cikel potrebujemo unijo dveh poti, ki pokrije vsa vozlišča. Poiščemo minimalno vpeto drevo (z utežjo 9), odstranimo eno povezavo, da dobimo unijo dveh dreves, in dobimo spodnjo mejo $8 + 2 \times 2 + 2 \times 2 = 16$.

Pokažimo, da ni Hamiltonovega cikla z utežjo 16. Če tak cikel obstaja, na njem zagotovo ni povezave ij. Na grafu brez povezave ij so na kateremkoli Hamiltonovem ciklu zagotovo povezave ah, hi in ig, zato zagotovo ni povezave ai, saj bi tako dobili cikel dolžine 4. Potem pa je na Hamiltonovem ciklu zagotovo povezava ab in ni povezav gb in bf. Potem pa mora biti na ciklu povezava bc. Ker ima povezava bc dolžino 2, bo na Hamiltonovem ciklu vsaj pet povezav z dolžino 2, torej bo cikel imel dolžino vsaj $5 \times 2 + 7 = 17$.

2.17. Maribor – Dravograd – Celovec – Jesenice – Bovec – Idrija – Logatec – Ljubljana – Škofja Loka – Kranj – Kamnik – Celje – Maribor, dolžina 550 km.

2.18. (a)



(b) Iz matrike vidimo, da so stroški vseh povezav, ki pridejo v vozlišče I_5, vsaj 2 (ali, da so stroški vseh povezav z začetnim vozliščem I_3 vsaj 2). Zato je strošek vsakega Hamiltonovega obhoda vsaj $5 \times 1 + 2 = 7$. Obhod s stroškom 7 je na primer 0 1 5 3 4 2 0, torej je optimalen.

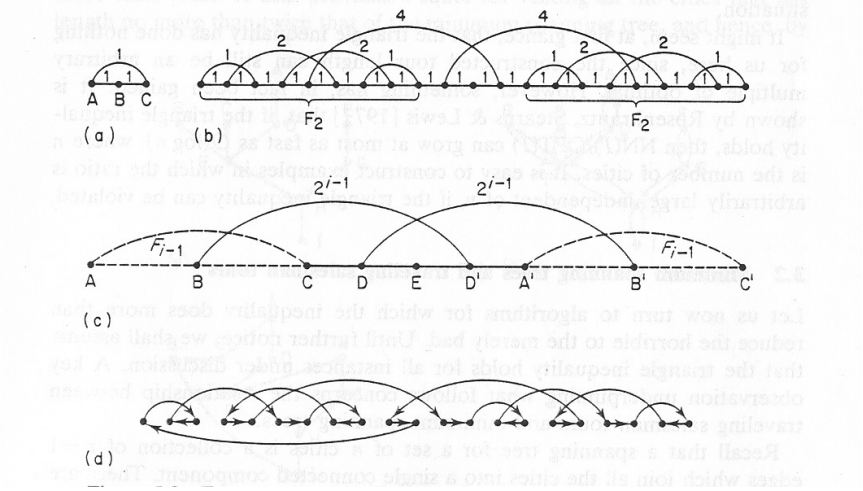
2.19. (c) Optimalno je zaporedje 1 4 3 2 0 s stroškom 1.5.

2.20. Družino uteženih grafov F_i lahko rekurzivno definiramo takole. F_1 je graf (a) na sliki spodaj. Graf F_{i+1} konstruiramo iz dveh kopij F_i in treh novih vozlišč D, E in D', tako kot kaže slika (c). Povezave, ki niso označene, imajo po definiciji dolžino enako dolžini najkrajše poti med krajiščema. Graf (b) je primer konstrukcije F_3 iz dveh F_2.

Algoritem NN lahko ob primerni izbiri začetnega vozlišča izbere obhod na sliki (d), katerega dolžina (na F_i) je $(i+2)2^i - 3$. Dolžina optimalnega obhoda je $\leq 6 \cdot 2^i - 8$.

Odtod sledi, da za poljubno število r obstaja naloga trgovskega potnika I, za katero velja $NN(I) > rOPT(I)$, kjer smo z $OPT(I)$ označili strošek optimalne rešitve.

Podrobnosti prepuščamo bralcu (glej tudi [E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan in D.B.Shmoys, The Traveling Salesman Problem, John Wiley & Sons, New York, 1985], str. 152.).


A B C
(a)
(b)
F_2
F_2
2^{i-1}
2^{i-1}
F_{i-1}
F_{i-1}
A B C D E D' A' B' C'
(c)
(d)

3

DREVESA

3.1. Karakteristične lastnosti dreves

Za matematika so drevesa pomembna in zanimiva, ker so v mnogih pogledih drevesa najenostavnejši netrivialni primeri grafov in imajo nekaj lepih lastnosti. Ko poskušamo v teoriji grafov dokazati kak splošen rezultat ali preveriti kakšno hipotezo, je včasih prikladno začeti s poskusom dokaza ustrezne lastnosti za drevesa. Obstaja nekaj hipotez, ki niso bile dokazane za splošne grafe, znano pa je, da veljajo za drevesa. Drevesa na naraven način srečamo tudi pri nekaterih algoritmih na splošnih grafih, na primer pri pregledovanju grafov.[1]

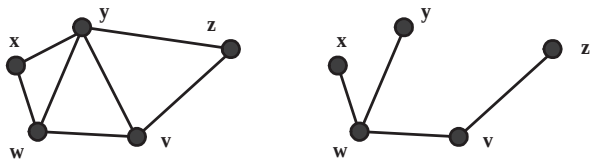
Za začetek se spomnimo definicije drevesa: **Drevo** je povezan graf brez ciklov. V naslednjem izreku je danih šest enakovrednih trditev. Vsaka opisuje eno od karakterističnih lastnosti dreves, zato bi lahko katerokoli uporabili za definicijo. (Za dokaz izreka glej nalogo 3.1..)

IZREK 3.1. *Naj bo T graf z n vozlišči. Naslednje trditve so ekvivalentne:*

a. Graf T je povezan in brez ciklov.
b. Graf T je povezan in ima $n-1$ povezav.
c. Graf T ima $n-1$ povezav in nobenega cikla.
d. Graf T je povezan in vsaka povezava je most.
e. Vsak par vozlišč grafa T je povezan z natanko eno potjo.
f. V grafu T ni nobenega cikla; če dodamo katerokoli povezavo, dobimo natanko en cikel.

Kasneje v tem poglavju bomo potrebovali pojem vpetega drevesa. Spomnimo se definicije vpetega drevesa v povezanem grafu: Naj bo G povezan graf. **Vpeto drevo** v G je podgraf grafa G, na katerem so vsa vozlišča grafa in je drevo.

Na naslednji sliki je graf, ob njem pa eno izmed njegovih vpetih dreves:



[1] Drevesa so ena najpomembnejših družin grafov v kemijski teoriji grafov. Pred mnogimi leti je Artur Cayley (1821-1895) odkril metodo, s katero se da določiti število neizomorfnih označenih dreves na n vozliščih. To ustreza število izomer alkana (nenasičenega acikličnega ogljikovodika). Pojasnitev pojava izomerizma z uporabo teorije grafov je ena prvih in najpomembnejših aplikacij teorije grafov (Alexander Crum Brown, 1838-1922).

V vsakem povezanem grafu G lahko najdemo vpeto drevo z uporabo ene od naslednjih dveh metod.

Metoda odstranjevanja. Izberemo si katerikoli cikel v G in odstranimo eno od povezav cikla. (Če v G ni nobenega cikla, potem je G sam sebi vpeto drevo.) Ker smo odstranili povezavo na ciklu, je dobljeni graf povezan. Postopek ponavljamo, dokler ni nobenega cikla več: dobljeni graf je vpeto drevo.

V zgornjem grafu lahko na primer odstranimo povezave:
vy (ki pretrga cikel $vwyv$),
yz (ki pretrga cikel $vwyzv$),
xy (ki pretrga cikel $wxyw$).

Tako dobimo zgoraj narisano vpeto drevo.

Metoda konstrukcije. Po vrsti izbiramo povezave G tako, da ne dobimo nobenega cikla. Ponavljamo, dokler nismo vključili vseh vozlišč. Tako lahko v gornjem grafu izberemo povezave vz, wx, xy in yz. Na ta način ne ustvarimo nobenega cikla, in dobimo prvo od zgoraj narisanih vpetih dreves.

Podobno metodo bomo uporabili v algoritmu za računanje minimalnega vpetega drevesa na strani 65.

3.2. Centri in centroidi

Ko dokazujemo trditve v zvezi z drevesi, pogosto začnemo v sredini drevesa in se pomikamo navzven. Tak pristop je uporabil Arthur Cayley leta 1870, ko je štel število kemijskih molekul z dano formulo tako, da jih je gradil korak za korakom. V novejšem času so v računalništvu uporabili koncept *uravnoteženega drevesa*. Tu gradimo drevo tako, da so različna poddrevesa, ki se nadaljujejo iz vsakega vozlišča „uravnotežena“ – to pomeni, da imajo (vsaj približno) enako število vozlišč)[2]. Kaj pa pomeni 'sredina' drevesa?

Sredino drevesa lahko definiramo na več načinov. Tu si bomo ogledali dva, oba pa bomo predstavili s konstrukcijsko metodo.

Metoda 1. Odstrani vsa vozlišča stopnje 1, skupaj s povezavami, ki jih imajo za krajišče; ponavljaj tako dolgo, dokler ne dobiš *ali* enega vozlišča (imenovanega **center**) *ali* dveh vozlišč (imenovanih **bicenter**).

Drevo s centrom imenujemo **centrirano drevo**, drevo z bicentrom pa **bicentrirano drevo**. Vsako drevo je centrirano ali bicentrirano, oboje hkrati pa ne more biti.

Opomba. Če je vozlišče c center, potem velja

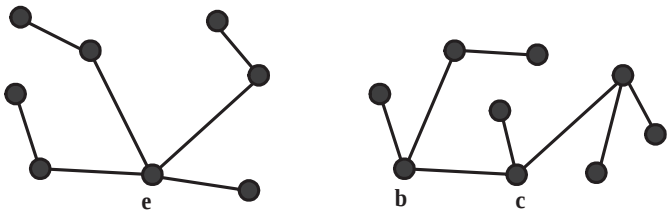
$$\max_{v \in V} d(c, v) = \min_{u \in V} \max_{v \in V} d(u, v).$$

[2]Določanje centrov (in nekaterih drugih na podoben način definiranih točk) na uteženih grafih spada na področje lokacijskih problemov [M. Daskin, Network and Discrete Location, Wiley, New York 1995.] Obstajajo različne variante, s katerimi modeliramo naloge optimalne postavitve reševalnih postaj, trgovskih centrov in na drugi strani nezaželjenih centrov kot so odlagališča odpadkov.

Vozlišče c je center natanko tedaj, ko je edino, za katero to velja. Če sta taki vozlišči dve, potem tvorita bicenter. Na drevesu več kot dveh takih vozlišč ne more biti.[3] (Glej nalogo 3.8..)

Enako lahko definiramo centre na poljubnih grafih, seveda pa se v tem primeru lahko zgodi, da je kandidatov veliko.

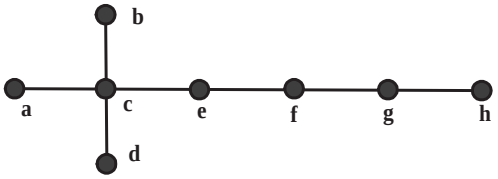
Primer: Na sliki sta *centrirano drevo* s centrom e in *bicentrirano drevo* s centrom bc.



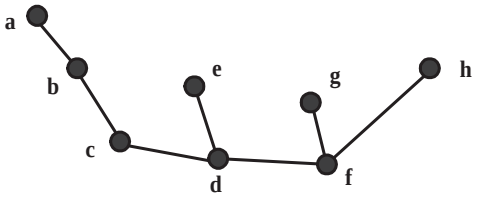
Metoda 2. Za vsako vozlišče v stopnje 2 ali več preštej število vozlišč v vsakem od poddreves, ki izhajajo iz v. Naj bo n_v največje od teh števil. Dokazati se da, da za drevo z n vozlišči velja: *ali* obstaja samo eno vozlišče v, za katero je $n_v \leq \frac{1}{2}(n-1)$ (**centroid**) *ali* pa obstajata dve sosednji vozlišči v in w, za kateri je $n_v = n_w = \frac{1}{2}n$ (**bicentroid**). (Glej nalogo 3.9.)

Centroid ali bicentroid si lahko predstavljamo kot „težišče“ drevesa. Drevo s centroidom imenujemo **centroidno drevo**, drevo z bicentroidom pa **bicentroidno drevo**. Vsako drevo je centroidno ali bicentroidno, oboje hkrati pa ne more biti.

Primeri: $n_c = 4$, $n_e = 4$, $n_f = 5$ in $n_g = 6$, zato je spodnje drevo bicentroidno z bicentroidom ce.



$n_b = 6$, $n_c = 5$, $n_d = 3$ in $n_f = 5$, zato je spodnje drevo centroidno s centroidom d.



[3] **ekscentričnost vozlišča** (ali **izsrednost**) definiramo takole: $e(v) = \max_{v \in V} d(u,v)$ Potem so centri (in bicentri) ravno vozlišča, za katera velja: $e(c) = \min_{u \in V} e(v)$.

Opomba. Zanimivo posplošitev dobimo, če za lokacijo centra dovolimo tudi točke na povezavah. Povezavo dolžine λ v tem primeru interpretiramo kot daljico dolžine λ in za lokacijo lahko izberemo katerokoli točko daljic vključno s krajišči. V tem primeru govorimo o **absolutnih centrih** grafa. Omenimo še eno posplošitev, *nezaželene centre*. Tu iščemo vozlišča (točke), za katere je minimalna razdalja maksimalna.[4]

3.3. Pregledovanje grafov

Pogosto se znajdemo pred nalogo, ki od nas zahteva, da sistematično iščemo po dani drevesni strukturi. Tako je računalniška datoteka včasih organizirana kot drevesna podatkovna struktura, posebno kadar uporablja spomin z direktnim dostopom (RAM, random-access memory). V takem primeru potrebujemo metodo za sistematično iskanje po drevesu, kadarkoli potrebujemo določen del informacije. To pogosto pomeni pregledovanje vseh delov drevesa, dokler ne najdemo iskanega vozlišča ali povezave. Da se izognemo nepotrebnemu zapravljanju časa in računalniških zmogljivosti, potrebujemo tehniko iskanja, ki bo zagotovo obiskala vsak del drevesa, ne da bi se prevečkrat zadrževala pri kakšnem vozlišču.

Obstajata dve dobro znani metodi pregledovanja, ki se razlikujeta v vrstnem redu obiskovanja vozlišč. Običajno ju imenujemo **pregledovanje v globino** (DFS, depth-first search) in **pregledovanje v širino** (BFS, breadth-first search). Pri vsaki od teh metod zapisujemo seznam že obiskanih vozlišč drevesa in označimo, v katero smer je bila povezava že uporabljena. Metodi se razlikujeta samo v načinu, kako konstruirata zaporedje obiskanih vozlišč.

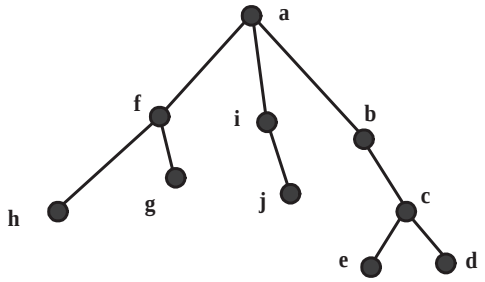
Obe metodi je mogoče uporabiti tudi na bolj splošnih tipih grafov. V takih primerih uporabe metodi dejansko pregledujeta vozlišča primernega v grafu vpetega drevesa. Dobljeni vpeti drevesi ponavadi imenujemo BFS- in DFS-vpeto drevo. Izkaže se, da je BFS-drevo enolično določeno z izbiro začetnega vozlišča. DFS dreves je praviloma tudi pri istem začetnem vozlišču lahko več.

Pregledovanje v globino

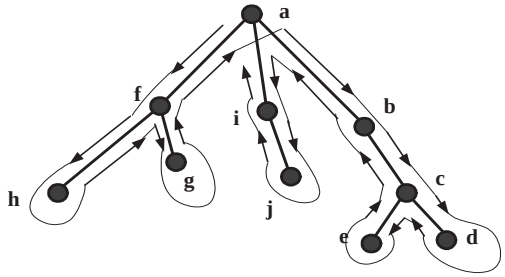
Osnovna ideja pregledovanja v globino je prodiranje kar se da v globino, preden se pot pregledovanja razveji.

Primer: Opišimo postopek na drevesu na sliki 3.1. Začnimo v vozlišču a. Izberemo poljubno sosednje vozlišče, recimo vozlišče f. V naslednjem koraku izberemo poljubnega soseda vozlišča f, ki še ni bil obiskan, na primer h. V naslednjem koraku bi izbrali poljubnega soseda vozlišča h, ki še ni bil obiskan. Vendar takega vozlišča ni, zato se vrnemo nazaj v vozlišče f in pogledamo, če je še kakšno neobiskano sosednje vozlišče vozlišča f. Izberemo g. Ker iz g ne moremo nadaljevati v globino, se vrnemo v f in v a. Vozlišče a ima neobiskane sosede, i in b. Izberemo enega od njiju in nadaljujemo. Eno od možnih pregledov je s puščicami označeno na naslednji sliki. Ker smo vedno izbirali enega od neobiskanih sosedov vozlišča, v katerem smo, je večkrat na voljo več enakovrednih

[4] Obe nalogi sta rešljivi v linearnem času na drevesih (in kaktusih), na poljubnih grafih pa sta problema še vedno rešljiva v polinomskem času. Naloge postanejo NP-težke, če namesto enega želimo postaviti dva ali več centrov na grafu. Glej tudi [B. Zmazek in J.Žerovnik, Nezaželeni centri v grafu, Zbornik posvetovanja DSI99, Slovensko društvo Informatika, 1999, str. 777-784, B. Zmazek in J.Žerovnik, Algoritem za računanje absolutnega centra na uteženih drevesih in kaktusih v linearnem času. Zbornik posvetovanja DSI02, Slovensko društvo Informatika 2002, str. 360-365.]



Slika 3.1: Drevo T.

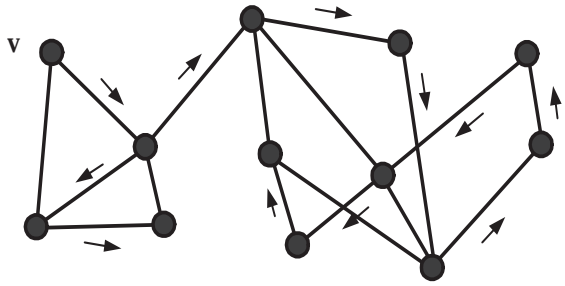


Slika 3.2: Drevo T z enim od pregledov v globino.

izbir, in zato zaporedje obiskovanja vozlišč ni enolično določeno.

Zgornje postopke lahko razširimo na poljubne grafe, kot pokažemo v naslednjem primeru.

Primer: Vzemimo graf na sliki 3.3 in začnimo v vozlišču v.



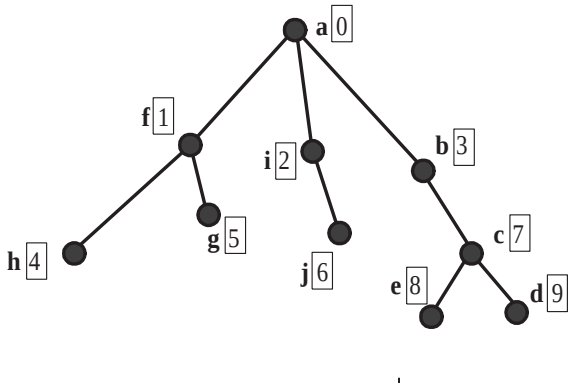
Slika 3.3: Graf z enim od pregledov v globino.

S puščicami smo označili tiste povezave, ki smo jih uporabili pri prvem obisku novega vozlišča. Te povezave tvorijo vpeto drevo. Tako drevo imenujemo **DFS vpeto drevo**. Ker bi lahko že v prvem koraku izbrali drugega soseda vozlišča v, in s tem drugo povezavo označili s puščico, vidimo, da DFS vpeto drevo z začetnim vozliščem na tem grafu ni enolično določeno.

Pregledovanje v širino

Osnovna ideja iskanja v širino je obiskati čim več vozlišč, preden se spustimo bolj globoko v drevo. To pomeni, da obiščemo najprej vsa vozlišča, ki so sosednja trenutno obravnavanemu, preden gremo v naslednje vozlišče.

Primer: Opišimo postopek na drevesu na sliki 3.1. Začnimo v vozlišču a. Izberemo poljubno sosednje vozlišče, recimo vozlišče f. V naslednjem koraku izberemo enega od preostalih sosedov vozlišča a, na primer i. V naslednjem koraku izberemo zadnjega od neobiskanih sosedov vozlišča a, torej b. Ker je sosedov vozlišča a zmanjkalo, se premaknemo v prvo od vozlišč, ki smo ga že obiskali, vendar še nismo pregledali vseh njegovih sosedov. V našem primeru je to vozlišče f. Izberemo enega od sosedov, na primer h. V naslednjem koraku izberemo edinega preostalega neobiskanega soseda g in ker f nima več nobenega neobiskanega soseda, se premaknemo v naslednje vozlišče i in nadaljujemo s pregledom sosedov. Na naslednji sliki smo z oznakami v okvirčkih označili enega od možnih zaporedij pregledovanja v širino z začetkom v vozlišču a. Ker smo vedno izbirali enega



Slika 3.4: Drevo T z enim od pregledov v širino.

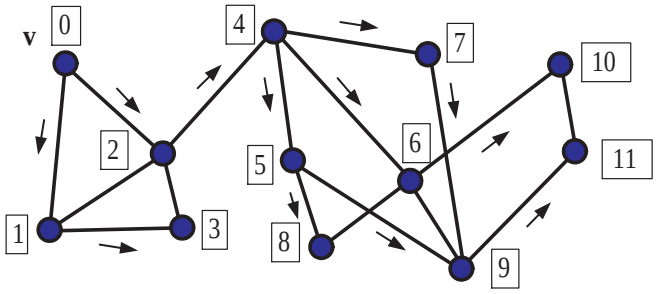
od neobiskanih sosedov vozlišča, v katerem smo, je večkrat na voljo več enakovrednih izbir, in zato zaporedje obiskovanja vozlišč ni enolično določeno.

Če so vozlišča drevesa narisana v vodoravnih „nivojih“ kot v našem primeru, potem pri pregledovanju v širino vsakič obiščemo vsa vozlišča enega nivoja, preden se premaknemo za en nivo globje.

Zaporedje pregledovanja drevesa v širino je torej usklajeno z razdaljami: vedno pridejo najprej na vrsto vozlišča na razdalji 1 od začetnega vozlišča, potem vozlišča na razdalji 2 in tako dalje.

Naslednji primer kaže, kako lahko iskanje v širino uporabimo na poljubnem grafu.

Primer: Vzemimo graf na sliki 3.5. Označen je pregled v širino z začetkom v vozlišču v. V okvirjih je zapisan vrstni red, v katerem smo obiskovali vozlišča. Spet smo s puščicami označili povezave, po katerih smo prvič obiskali neko vozlišče. Te povezave tvorijo vpeto drevo, ki ga imenujemo **BFS vpeto drevo**. Kot smo ugotovili že pri pregledu drevesa, najprej pridejo na vrsto vozlišča na razdalji 1 od vozlišča v, potem vozlišča na razdalji 2 od vozlišča v in tako dalje. Vrstni red



Slika 3.5: Graf z enim od pregledov v širino.

obiskovanja vozlišč, kot smo že videli, ni enolično določen. BFS vpeto drevo z začetkom v danem vozlišču pa je. (Za razliko od DFS vpetega drevesa).

Praktičen primer uporabe iskanja v širino je algoritem za iskanje najkrajših poti (glej stran 112).

3.4. Konstrukcije dreves

Algoritmi za konstrukcijo dreves so koristni vsaj pri dveh vrstah nalog. Prva vrsta zajema naloge, pri katerih želimo sestaviti velika drevesa iz manjših. Kot smo že omenili v tem poglavju, lahko vsa drevesa z izbrano lastnostjo dobimo tako, da manjšemu drevesu postopoma dodajamo vozlišče in povezavo, pri tem pa pazimo, da sproti odstranjujemo dvojnike. Druga vrsta nalog, pri katerih moramo poiskati vpeto poddrevo posebne vrste, je bolj zapletena. Značilen primer take naloge je *problem minimalnega vpetega drevesa*, ki si ga bomo zdaj ogledali.

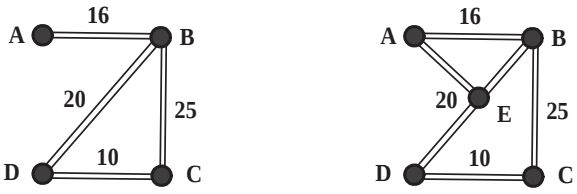
Problem minimalnega vpetega drevesa

Recimo, da želimo zgraditi sistem kanalov za izsuševanje, ki naj bo povezan z danimi lokacijami. Stroški izgradnje in vzdrževanja vsakega od možnih kanalov so znani vnaprej, vemo pa tudi, da nekateri pari lokacij ne morejo biti neposredno povezani s kanalom zaradi geografskih ali političnih razlogov (na primer hribovit teren ali politično nestabilno področje). Kako naj zgradimo sistem kanalov, ki bo povezoval vse lokacije s čim manjšimi stroški?

Praktičnih primerov za nalogo minimalnega vpetega drevesa je veliko. Podobno kot o kanalih za namakanje bi lahko premišljali o naftovodih, o komunikacijskih povezavah (telefon, internet) ali o energetskem omrežju[5].

Nalogo si lahko zastavimo na dva načina, glede na to, ali dovolimo dodatne „lokacije“, v katerih se kanali smejo sekati. V spodnjem primeru morda lahko zmanjšamo stroške z dodatkom lokacije E, ki jo povežemo z A.

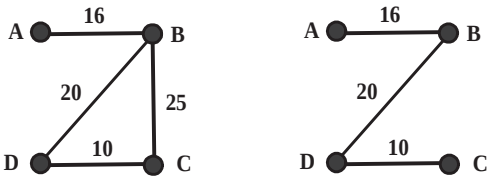
[5]Osnovna zahteva pri teh aplikacijah je, da dobimo povezan sistem z minimalnimi stroški. Zanesljivost sistema lahko povečamo tako, da dodamo dodatne povezave, tako da je vsako vozlišče na vsaj enem ciklu. Tako ob izpadu ene povezave sistem ne razpade. Smiselne so tudi druge naravne posplošitve, ki jih lahko prav tako dobro modeliramo z grafi. Graf, pri katerem je vsaka povezava na največ enem ciklu, imenujemo *kaktus*.



Slika 3.6: Dva sistema kanalov.

Za mnoge probleme te vrste velja, da lahko strošek dodajanja nove lokacije (ki je lahko telefonska centrala ali transformator) bistveno presega morebitni prihranek zaradi cenejših povezav, zato potrebna matematična analiza postane precej zapletena. Zaradi tega bomo tu privzeli, da vsaka povezava povezuje dve obstoječi lokaciji.

Problem minimalnega vpetega drevesa lahko predstavimo z uteženim grafom, katerega vozlišča so lokacije, povezave z utežmi pa so kanali z danimi stroški. Poiskati moramo povezan podgraf z najmanjšo skupno utežjo, na katerem so vsa vozlišča grafa. *Tak graf mora biti vedno vpeto drevo*, saj če bi podgraf vseboval cikel, bi lahko vedno zmanjšali stroške z odstranitvijo ene od povezav cikla[6].



Slika 3.7: Utežen graf sistema kanalov in minimalno vpeto drevo.

V našem primeru ima graf skupno utež 71. Odstranitev ene od povezav cikla BCD zmanjša skupno utež in nam da vpeto drevo. Očitno dobimo vpeto drevo z najmanjšo utežjo, če odstranimo povezavo BC. Najmanjši skupni strošek je torej $16 + 20 + 10 = 46$. Naj bo T vpeto drevo z najmanjšo skupno utežjo v povezanem uteženem grafu G. Potem T imenujemo **minimalno vpeto drevo** grafa G, njegovo težo pa označimo z $W(G)$. Problem lahko zdaj zapišemo še v jeziku teorije grafov.

Problem minimalnega vpetega drevesa: Za dani uteženi graf poišči minimalno vpeto drevo.

Pristop, ki deluje v tem primeru, je znan kot *požrešna metoda*. Tu si bomo ogledali dve različici požrešne metode, najprej Kruskalov[7] algoritem:

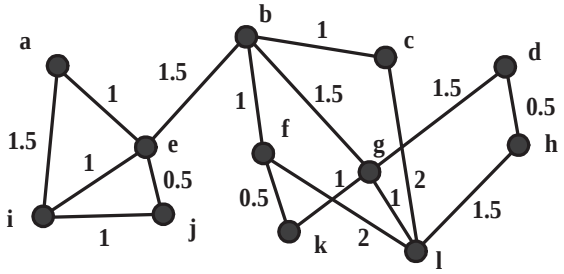
> **KRUSKALOV ALGORITEM**. Sestavi minimalno vpeto drevo grafa G tako, da po vrsti izbiraš povezave z naraščajočimi utežmi in povezavo dodaš, če z njo ne dobiš cikla.

[6] To velja, če so vsi stroški pozitivna števila.

[7] Kruskal v svojem članku citira članek češkega pionirja teorije grafov Borůvke iz leta 1928, v katerem je Borůvka opisal splošno metodo, katere poseben primer sta Kruskalov in Primov algoritem. Bolj pravilno bi bilo torej algoritem imenovati po Borůvki.

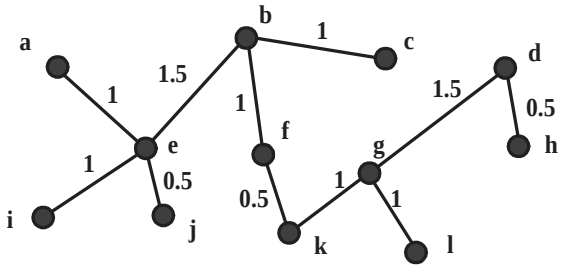
Ime „požrešna metoda“ je dobil algoritem zato, ker na vsakem koraku naredimo najbolj požrešno potezo, ki je na voljo (izberemo povezavo z najmanjšo utežjo), ne glede na to, kaj se dogaja drugje v grafu. Algoritmi te vrste običajno niso uspešni v praksi, ta pa deluje, kot bomo pokazali v nalogi 3.10.

Uporabo algoritma bomo ponazorili na primeru. Vzemimo graf na sliki 3.8: Uredimo povezave po



Slika 3.8: Kruskalov algoritem na grafu.

naraščajočih utežeh: ej, fk, dh (povezave z utežjo 0.5), ae, ie, ij, bf, bc, gk, gl (povezave z utežjo 1), ai, be, bg, dg, hl (povezave z utežjo 1.5), cl, fl (z utežjo 2). Konstruirajmo vpeto drevo na vozliščih $V(G)$: Dodajajmo povezave v tem vrstnem redu, razen če bi dodana povezava povezala že povezani vozlišči. V zaporedju ej, fk, dh, ae, ie, pridemo do povezave ij. Če bi dodali ij, bi dobili cikel eij, zato povezave ij ne dodamo. Nadaljujemo z bf, bc, gk, gl, izpustimo ai zaradi cikla aei, in nadaljujemo z dodajanjem be, izpustimo bg, in dodamo dg. Dobili smo minimalno vpeto drevo z utežjo 10.5 na sliki 3.9. (Povezav hl, cl in fl ne smemo dodati, saj bi dobili cikle.)



Slika 3.9: Minimalno vpeto drevo, dobljeno s Kruskalovim algoritmom.

Požrešno metodo je enostavno uporabiti za konstrukcijo najmanjšega vpetega drevesa, če računamo „peš“ na majhnih grafih. Za programiranje na računalniku je primernejši naslednji algoritem, ki ima nekoliko boljšo oceno časovne zahtevnosti (to bomo obravnavali v poglavju 5). V Kruskalovem algoritmu je potrebno najprej urediti povezave po naraščajočih utežeh in sproti preverjati, ali smo dobili kak cikel. Konstrukcijo minimalnega vpetega drevesa lahko naredimo tudi z drugo različico požrešne metode, ki jo imenujemo *Primov algoritem.*[8]

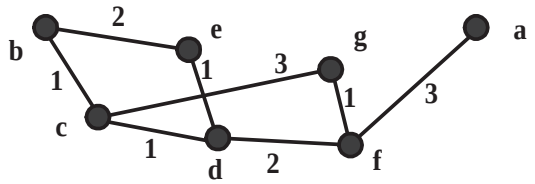
[8]Leta 1930 je Jarnik objavil poenostavljeno različico algoritma, ki jo zdaj običajno imenujemo Primov algoritem. Ker je bil članek napisal v češkem jeziku, je njegovo delo še manj znano kot delo Borůvke. Glej [J.Nešetřil, E.Milková, H.Nešetřilova, Otakar Borůvka on minimum spanning tree problem, Discrete Mathematics 233 (2001) 3-36].

PRIMOV ALGORITEM. Najmanjše vpeto drevo T povezanega uteženega grafa G konstruiramo s postopkom:

a. dodaj poljubno vozlišče v prazno drevo T;
b. ponavljaj: dodaj v T povezavo z najmanjšo utežjo, ki povezuje neko vozlišče iz T z nekim vozliščem, ki še ni v T.

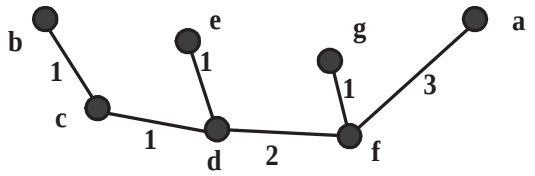
Prednost Primovega algoritma je, da lahko deluje neposredno na tabeli uteži namesto na grafu. Če je graf velik, je zato metoda bolj primerna za programiranje. Vse, kar moramo narediti, je, da zbrišemo vrstico tabele, kadarkoli ustrezno vozlišče dodamo v T in potem poiščemo najmanjši element v stolpcu, ki ustreza vozliščem v T. Metodo ponazorimo z naslednjim primerom.

Primer: Izvedimo Primov algoritem za iskanje najmanjšega vpetega drevesa uteženega grafa na naslednjem grafu:



Slika 3.10: Primer uporabe Primovega algoritma.

Začnimo (na primer) v vozlišču a, dodajmo ga v drevo T. Nadaljujemo z dodajanjem vozlišča, ki je najbližje že konstruiranemu drevesu. Najprej je to vozlišče f, ker je edino sosednje dosedaj zgrajenemu drevesu. Dodamo f in povezavo af. V naslednjem koraku sta sosedi drevesa g in d, in ker je g bližja, dodamo g in povezavo fg. Dodamo d in fd. V naslednjem koraku sta sosedi drevesa c in e, in ker sta obe od drevesa oddaljeni za 1, izberemo poljubno od njiju. Tako nadaljujemo in dobimo drevo na sliki 3.11. Pri konstrukciji smo imeli dve enakovredni izbiri, vendar v tem primeru vsakič dobimo isto minimalno vpeto drevo. V splošnem seveda lahko obstaja več minimalnih vpetih dreves.



Slika 3.11: Minimalno vpeto drevo, dobljeno s Primovim algoritmom.

Primov algoritem lahko izvajamo na tabeli (matriki sosednosti uteženega grafa), katere elementi so dolžine povezav. Primer, ki smo ga obdelali pravkar, zapišimo v tabelo. (Tam, kjer povezav ni, smo zapisali znak $-$ in s tem poudarili, da ni povezave. Lahko bi namesto minus zapisali tudi ∞ ali pa kakšno zelo veliko število, ki bi bilo zagotovo večje od vseh uteži grafa.)

	a	b	c	d	e	f	g
a	0	-	-	-	-	3	-
b	-	0	1	-	2	-	-
c	-	1	0	1	-	-	3
d	-	-	1	0	1	2	-
e	-	2	-	1	0	-	-
f	3	-	-	2	-	0	1
g	-	-	3	-	-	1	0

Izberemo vrstico začetnega vozlišča. Recimo, da smo izbrali, tako kot zgoraj, vozlišče a. Zbrišemo vrstico a in označimo stolpec a. V stolpcu izberemo minimalni element, v našem primeru je to 3, v vrstici vozlišča f.

	$\boxed{a}$	b	c	d	e	f	g
b	-	0	1	-	2	-	-
c	-	1	0	1	-	-	3
d	-	-	1	0	1	2	-
e	-	2	-	1	0	-	-
f	$\boxed{3}$	-	-	2	-	0	1
g	-	-	3	-	-	1	0

V drevo dodamo vozlišče f in povezavo af (dolžine 3). Iz tabele zbrišemo vrstico f in označimo stolpec vozlišča f. Zdaj označimo minimalni element (ali enega od njih, če jih je več) v označenih stolpcih (a in f).

	$\boxed{a}$	b	c	d	e	$\boxed{f}$	g
b	-	0	1	-	2	-	-
c	-	1	0	1	-	-	3
d	-	-	1	0	1	2	-
e	-	2	-	1	0	-	-
g	-	-	3	-	-	$\boxed{1}$	0

V drevo dodamo vozlišče g in povezavo gf (dolžine 1). Iz tabele zbrišemo vrstico g in označimo stolpec vozlišča g.

	$\boxed{a}$	b	c	d	e	$\boxed{f}$	$\boxed{g}$
b	-	0	1	-	2	-	-
c	-	1	0	1	-	-	3
d	-	-	1	0	1	$\boxed{2}$	-
e	-	2	-	1	0	-	-

V označenih stolpcih smo označili minimalni element v vrstici d. Zato v drevo dodamo vozlišče d in povezavo df, zbrišemo vrstico d in označimo stolpec d.

	$\boxed{a}$	b	c	$\boxed{d}$	e	$\boxed{f}$	$\boxed{g}$
b	-	0	1	-	2	-	-
c	-	1	0	1	-	-	3
e	-	2	-	1	0	-	-

V označenih stolpcih je minimalni element enak 1, pojavi se na dveh mestih. Zato lahko izberemo vozlišče c in povezavo cd ali pa vozlišče e in povezavo de. V primeru, da smo izbrali prvo, dobimo:

	$\boxed{a}$	b	$\boxed{c}$	$\boxed{d}$	e	$\boxed{f}$	$\boxed{g}$
b	-	0	1	-	2	-	-
e	-	2	-	1	0	-	-

Spet imamo dve izbiri, bc ali de. Če izberemo bc, nam ostane

	$\boxed{a}$	$\boxed{b}$	$\boxed{c}$	$\boxed{d}$	e	$\boxed{f}$	$\boxed{g}$
e	-	2	-	1	0	-	-

zato moramo nazadnje dodati še vozlišče e in povezavo de.

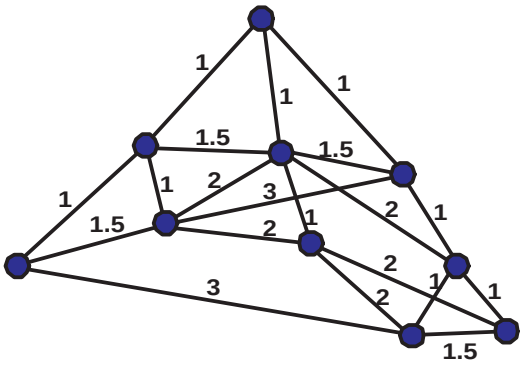
Uporaba na problemu trgovskega potnika

Glede na preprostost požrešne metode za problem iskanja najmanjšega vpetega drevesa bi lahko upali, da bo obstajal preprost algoritem tudi za reševanje problema trgovskega potnika. Kot smo že omenili, na žalost ne poznamo nobenega takega algoritma in zelo verjetno je, da takega algoritma sploh ni.

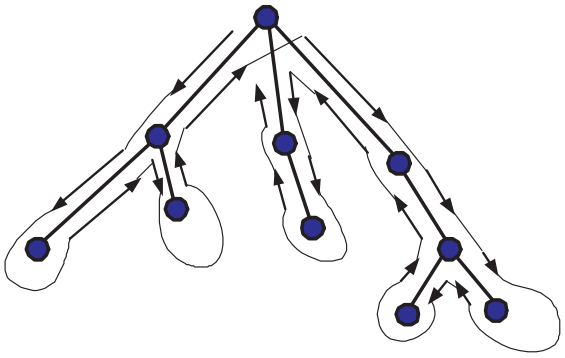
Zato smo pogosto prisiljeni iskati približne rešitve problema. Včasih si pri iskanju približnih rešitev pomagamo tako, da poiščemo meje za optimalno rešitev. Tako lahko približno ocenimo kako dobro rešitev smo dobili ne da bi poznali strošek optimalne rešitve. *Spodnjo mejo* za problem trgovskega potnika lahko dobimo tako, da rešimo problem minimalnega vpetega drevesa! Metodo opravičimo z naslednjim argumentom: Če vzamemo Hamiltonov cikel z najmanjšo utežjo v uteženem polnem grafu in odstranimo vozlišče A in povezave s krajišči v A, dobimo pot, ki gre skozi ostala vozlišča. Taka pot je vpeto drevo za polni graf na preostalih vozliščih, utež Hamiltonovega cikla pa dobimo, če dodamo teži poti uteži odstranjenih dveh povezav. *Spodnjo mejo za rešitev problema trgovskega potnika torej dobimo, če seštejemo utež najmanjšega vpetega drevesa in dve najmanjši uteži povezav s krajiščem v A.*

Za utež najmanjšega Hamiltonovega cikla včasih potrebujemo tudi *zgornjo mejo*. Morda najbolj preprosta metoda je, da izberemo nek Hamiltonov cikel in izračunamo njegovo skupno utež. Na osnovi snovi tega poglavja temelji naslednja metoda. Naredimo iskanje v globino na najmanjšem vpetem drevesu, tako da dobimo sklenjen sprehod, ki obišče vsako vozlišče vsaj enkrat, zato mora njegova skupna utež biti enaka ali večja od rešitve problema trgovskega potnika. Toda, če naredimo iskanje v globino na najmanjšem vpetem drevesu, potem prehodimo vsako povezavo natanko dvakrat in prehojena pot je torej enaka dvojni skupni teži najmanjšega vpetega drevesa. Torej je vrednost rešitve problema trgovskega potnika največ dvakrat večja od teže najmanjšega vpetega drevesa.

To zgornjo mejo lahko precej izboljšamo tako, da uberemo „bližnjice“, kadarkoli je to mogoče. Poglejmo, kako dobimo zgornjo mejo na naslednjem primeru.

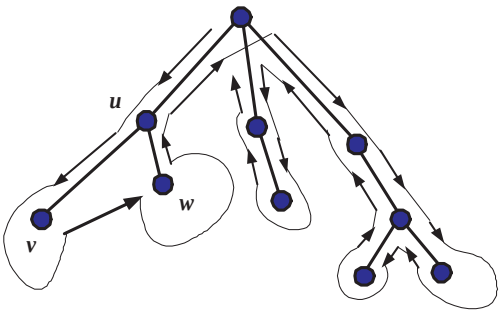


Najprej poiščimo minimalno vpeto drevo

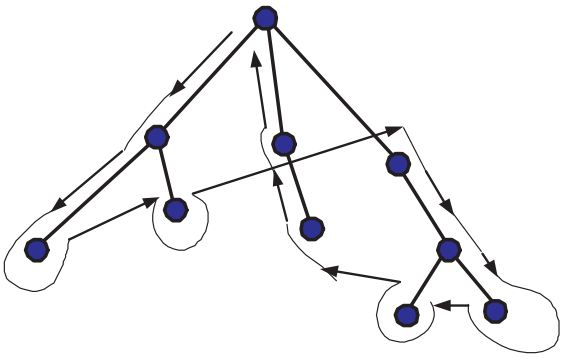


Na drevesu smo označili obhod, na katerem so vsa vozlišča grafa, katerega dolžina je dvakratna vrednost vsote uteži minimalnega vpetega drevesa.

Če za uteži na povezavah velja trikotniška neenakost, lahko v konkretnem primeru zgornjo mejo še izboljšamo. (Trikotniška neeenakost za uteži velja, če je za poljuben trikotnik u, v, w izpolnjena neenakost $\lambda(uv) \leq \lambda(uw) + \lambda(wv)$). To pomeni, da lahko vedno izberemo „bližnjice“.



Dobimo Hamiltonov cikel dolžine 14.



Ta obhod ni optimalen, saj hitro najdemo krajše Hamiltonove cikle.

Spodnjo mejo za ta primer lahko dobimo tako, da izberemo vozlišče, ki je krajišče povezav z utežmi 1,2,2,2 in poiščemo minimalno drevo preostanka grafa. Dobimo $3 + 8 = 11$.

3.5. Naloge z rešitvami

3.1. Dokaži izrek 3.1.: Naj bo T graf z n vozlišči. Naslednje trditve so ekvivalentne:

a. Graf T je povezan in brez ciklov.
b. Graf T je povezan in ima $n - 1$ povezav.
c. Graf T ima $n - 1$ povezav in nobenega cikla.
d. Graf T je povezan in vsaka povezava je most.
e. Vsak par vozlišč grafa T je povezan z natanko eno potjo.
f. V grafu T ni nobenega cikla; če dodamo katerokoli povezavo, dobimo natanko en cikel.

3.2. Konstruiraj vsa neizomorfna drevesa na 6 vozliščih!

3.3. Koliko je neizomorfnih dreves s šest ali manj vozlišči?

3.4. Nariši vsa neizomorfna drevesa s sedmimi vozlišči.

3.5. Dokaži, da so drevesa dvodelni grafi.

3.6. Določi centre in bicentre drevesom iz naloge 3.2.

3.7. Določi centroide in bicentroide drevesom iz naloge 3.2.

3.8. Dokaži, da na drevesu obstajata največ dve vozlišči, za kateri je $e(c) = \min_{u \in V} e(v)$.

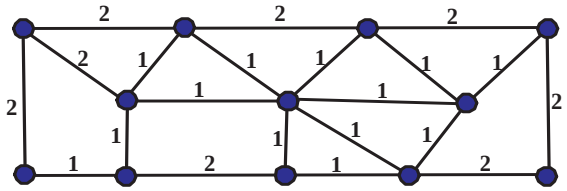
3.9. Dokaži, da na drevesu obstajata največ dve vozlišči, za kateri je $n_v \leq \frac{1}{2}(n-1)$.

3.10. Dokaži, da je Kruskalov algoritem optimalen.

3.11. Ali je rešitev s strani 67 (graf s slike 3.10) edina, ki jo lahko dobimo s Primovim algoritmom?

3.12. Poišči primer grafa, na katerem bo Primov algoritem lahko generiral različna minimalna vpeta drevesa.

3.13. Poišči minimalno vpeto drevo grafa



Ali je rešitev enolično določena?

3.14. Definirajmo utež povezave $e = uv$ z

$$w(e) = n_{e,u} n_{e,v} \ ,$$

kjer smo z $n_{e,x}$ označili število vozlišč v povezani komponenti, v kateri je vozlišče x. Dokaži, da je Wienerjevo število vsota uteži vseh povezav drevesa:

$$W(T) = \sum_{e \in E(T)} w(e) \ . \tag{3.1}$$

3.15. Definirajmo utež povezave $e = uv$ z $w(e) = n_{e,u} n_{e,v}$, kjer smo z $n_{e,x}$ označili število vozlišč, ki so bližje x kot drugemu krajišču povezave e. Izkaže se[9], da je Wienerjevo število enako vsoti uteži vseh povezav drevesa: $W(G) = \sum_{e \in E(G)} w(e)$.

Izračunaj Wienerjevo število grafa iz naloge 1.22. s pomočjo tako definiranih uteži na povezavah.

[9] Glej [J.Žerovnik, Topološki indeksi v kemijski teoriji grafov, Obzornik za matematiko in fiziko 44 (1997) 33-39.]

Rešitve

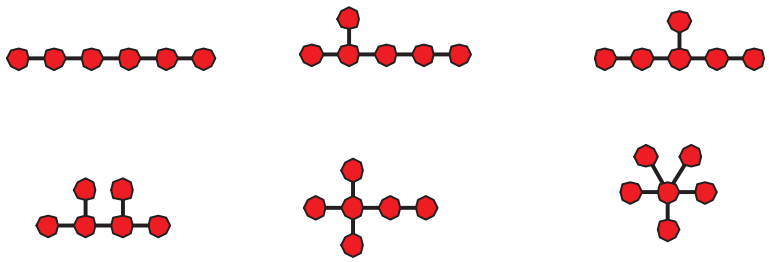
3.1. Ekvivalentnost trditev lahko pokažemo z dokazom implikacij (a) $\Longrightarrow$ (b) $\Longrightarrow$ (c) $\Longrightarrow$ (d) $\Longrightarrow$ (e) $\Longrightarrow$ (f) $\Longrightarrow$ (a)

(a) $\Longrightarrow$ (b): Z indukcijo na število vozlišč. Za majhne grafe trditev očitno velja. Privzemimo, da trditev velja za grafe z n vozlišči in naj bo G povezan graf brez ciklov z $n+1$ vozliščem. Vemo, da v vsakem grafu z minimalno stopnjo vsaj 2 obstaja cikel. (Naloga 2.4.. Ker v grafu ni nobenega cikla, v njem obstaja vsaj eno vozlišče stopnje 1. Odstranimo iz grafa G to vozlišče in povezavo, katere krajišče je. Dobimo povezan graf z n vozlišči in brez ciklov. Po indukcijski predpostavki ima ta graf $n-1$ povezav. Torej je graf G povezan in ima n povezav, kar smo želeli dokazati.

(b) $\Longrightarrow$ (c): Naj bo G povezan graf z $n-1$ povezavami. Po lemi o rokovanju je $2(n-1) = \sum \mathrm{st}(v)$, zato v grafu obstaja vsaj eno vozlišče s stopnjo manjšo kot 2. Ker je graf povezan, v njem ni vozlišč stopnje 0, zato obstaja vozlišče s stopnjo 1. To vozlišče gotovo ni na nobenem ciklu. Odstranimo to vozlišče in ponovimo sklep na dobljenem grafu. Vidimo, da nobeno vozlišče grafa G ne more biti na ciklu, torej v G ni nobenega cikla.

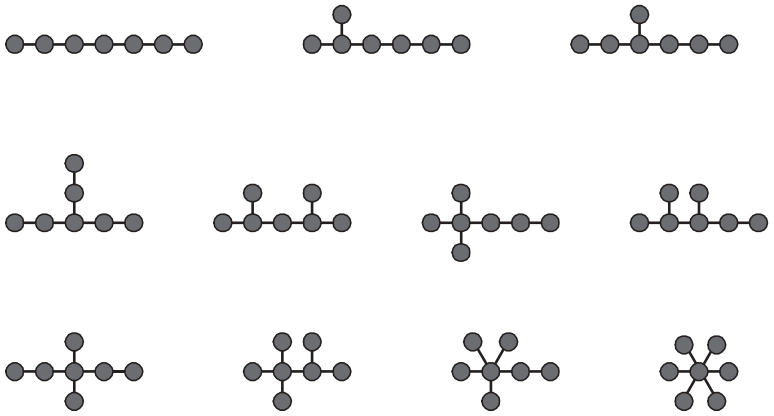
Dokaze preostalih implikacij prepuščamo bralcu.

3.2.



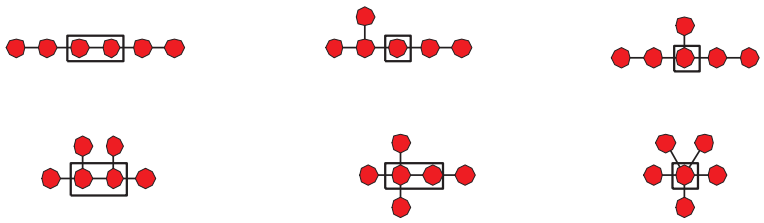
3.3. Obstaja 14 dreves s šest ali manj vozlišči.

3.4. Obstaja 11 dreves s sedmimi vozlišči.

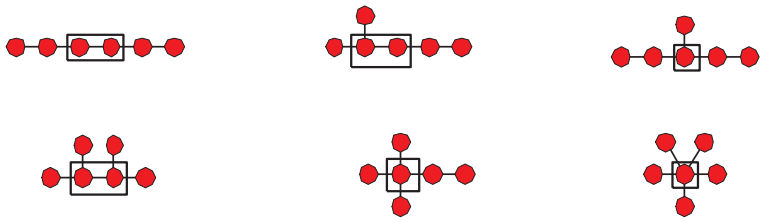


3.5. Namig: Izberemo poljubno vozlišče, recimo v, in ga damo v prvo množico V_1. Vse sosede vozlišča v damo v drugo množico V_2, sosede sosed spet v V_1 In tako dalje.

3.6. Centri in bicentri so označeni na sliki:



3.7. Centroidi in bicentroidi so označeni na sliki:



3.8. Če za vozlišče c velja $e(c) = \min_{u \in V} e(v)$, potem morata obstajati vsaj dve različni vozlišči u in v, tako da je $d(c, v) = e(c)$ in $e(c) - 1 \leq d(c, u) \leq e(c)$. (Če to ne bi bilo res, potem $e(c)$ ne bi bilo minimalno, saj bi imelo sosednje vozlišče v smeri proti v zagotovo manjšo ekscentričnost.)

Naj bo $w \neq c$ poljubno drugo vozlišče, za katero tudi velja $e(w) = \min_{u \in V} e(v)$. Potem mora biti w sosed c v smeri v. (Če to ne bi bilo res, bi veljalo $d(w, v) = d(w, c) + d(c, v) > 1 + e(c)$, kar je v nasprotju z minimalnostjo $e(w)$.) Če obstaja, je torej vozlišče w natanko določeno in v tem primeru sta w in c bicenter.

3.9. Naj bo c poljubno vozlišče drevesa T.

Označimo z MAX število vozlišč v največji komponenti povezanosti grafa $T \setminus c$.

Če je $MAX < \frac{n}{2}$ potem je c centroid.

Če je $MAX \geq \frac{n}{2}$ potem označimo s c' vozlišče iz največje komponente povezanosti, ki je sosednje s c. Za vozlišče c' velja: komponenta povezanosti, ki vsebuje c ima $n - MAX \leq \frac{n}{2}$ vozlišč. Vse druge komponente skupaj imajo MAX-1 vozlišč.

Označimo z MAX' število vozlišč v maksimalni komponenti povezanosti grafa $T \setminus c'$.

Če je $MAX' < \frac{n}{2}$, je c' centroid, ali pa sta c in c' skupaj bicentroid.

Če je $MAX' \geq \frac{n}{2}$, ponovimo sklep. Ker je $MAX' < MAX$, se velikost največje komponente povezanosti vsakič zmanjša, torej zagotovo v končno mnogo korakih pridemo do centroida ali bicentroida.

3.10. *Dokaz optimalnosti Kruskalovega algoritma.* Naj bo G povezan graf z n vozlišči. Naj bo T graf, ki ga dobimo s požrešnim algoritmom. Iz konstrukcije T sledi, da v T ni nobenega cikla. Prav tako vemo, da mora biti T povezan graf, sicer bi lahko dodali še eno povezavo, ne da bi dobili cikel. T tudi vsebuje vsa vozlišča grafa G. Če to ne bi bilo res, potem bi lahko dodali povezavo s krajiščem v takem vozlišču in ne bi dobili cikla. Torej je T vpeto drevo grafa G.

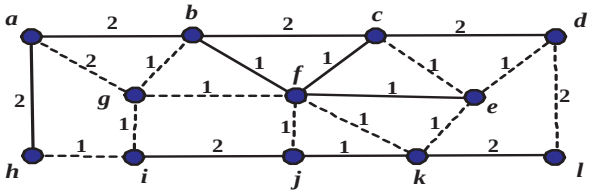
Dokazati moramo, da je T najmanjše vpeto drevo grafa G. To bomo dokazali s protislovjem. Recimo, da je S vpeto drevo grafa G z manjšo skupno težo kot jo ima T, torej $W(S) < W(T)$. Bodi e povezava z najmanjšo utežjo, ki leži v T in ne leži v S, in glejmo graf, ki ga dobimo, če grafu S dodamo povezavo e.

Ko smo S dodali povezavo e, smo dobili cikel C, na katerem leži povezava e. Na tem ciklu mora ležati vsaj ena povezava e', ki ni v T. Če v S zamenjamo e z e', spet dobimo vpeto drevo; označimo ga z S'. Iz konstrukcije T sledi, da utež e ne more presegati uteži e', zato je $W(S') \leq W(S)$. Vpeto drevo S' ima eno povezavo več skupno s T. Torej lahko S pretvorimo v T tako, da zamenjujemo po eno povezavo in vsakič se utež kvečjemu zmanjša. Torej je $W(T) \leq W(S)$, kar je v nasprotju z definicijo S. Dokaz je končan.

3.11. Da. Graf ima enolično določeno minimalno vpeto drevo grafa. (Vse povezave z utežjo 1 so gotovo na minimalnem vpetem drevesu, saj ne inducirajo nobenega cikla. Povezava z utežjo 2 je najcenejša povezava obeh komponent, induciranih z množicama $\{b, c, d, e\}$ in $\{a, f, g\}$. Vozlišče a lahko dosežemo samo z eno povezavo, ki mora biti na vsakem vpetem drevesu.)

3.12. Takih primerov je veliko. Najpreprostejši je graf trikotnik z enakimi utežmi na vseh treh povezavah. Začetno vozlišče lahko izberemo na tri načine in dobimo tri različna minimalna vpeta drevesa.

3.13. Uporabimo, na primer, Kruskalov algoritem. Eno od minimalnih vpetih dreves določajo črtkane povezave na spodnji sliki



Minimalno vpeto drevo ni enolično določeno. Če na primer povezavo ag zamenjamo s povezavo ab, dobimo novo vpeto drevo z minimalno utežjo 13.

3.14. Kot vemo, je drevo povezan graf brez ciklov. Če izberemo katerokoli povezavo drevesa in jo odstranimo, dobimo dve povezani komponenti.

Najprej je

$$\sum_{u\in V(T)}\sum_{v\in V(T)} d(u,v) = 2\sum_{\{u,v\}} d(u,v)\ ,$$

kjer vsota na desni strani teče po vseh (neurejenih) parih vozlišč $u, v \in V(T)$. Na levi smo vsak tak par $u \neq v$ šteli natanko dvakrat, vse razdalje $d(u,u)$ pa so seveda enake 0.

Veljavnost enakosti

$$\sum_{\{u,v\}} d(u,v) = \sum_{e} w(e)$$

pa preverimo tako, da na dva načina preštejemo povezave na vseh najkrajših poteh in se spomnimo pomena uteži $w(e)$. Torej je res

$$\frac{1}{2}\sum_{u\in V(T)}\sum_{v\in V(T)} d(u,v) = \sum_{e} w(e)\ .$$

Če je graf drevo, lahko torej Wienerjevo število izrazimo tudi malo drugače. (Zvezo $W(T) = \sum_{e\in E(T)} w(e)$ je poznal že Wiener[10]).

3.15. Izračunajmo razdalje in število najkrajših poti v grafu in jih zapišimo v tabeli.

par vozlišč	$\{v_1,v_2\}$	$\{v_1,v_3\}$	$\{v_1,v_4\}$	$\{v_1,v_5\}$	$\{v_2,v_3\}$
razdalja	1	2	3	2	1
število najkrajših poti	1	1	2	1	1

par vozlišč	$\{v_2,v_4\}$	$\{v_2,v_5\}$	$\{v_3,v_4\}$	$\{v_3,v_5\}$	$\{v_4,v_5\}$
razdalja	2	1	1	2	1
število najkrajših poti	2	1	1	2	1

Izračunajmo še uteži povezav. Povezava $e_1 = v_1v_2$ ima utež

$$w(e_1) = w(v_1v_2) = 1/n_{1,2} + 1/n_{1,3} + 2/n_{1,4} + 1/n_{1,5} = 1 + 1 + 2\left(\frac{1}{2}\right) + 1 = 4\ .$$

[10]H. Wiener, Correlation of Heats of Isomerization, and Diferences in Heats of Vaporization of Isomers, Among the Paraffin Hydrocarbons, Journal of the American Chemical Society 69 (1947) 2636-2638.

Podobno povezavi $e_2 = v_2v_3$ pripada utež

$$w(e_2) = w(v_2v_3) = 1/n_{1,3} + 1/n_{1,4} + 1/n_{2,3} + 1/n_{2,4} + 1/n_{3,5} = 1 + \frac{1}{2} + 1 + \frac{1}{2} + \frac{1}{2} = 3.5 \ .$$

Povezavam $e_3 = v_3v_4$, $e_4 = v_4v_5$ in $e_5 = v_1v_5$ pripadajo uteži $\frac{1}{2} + \frac{1}{2} + 1 + \frac{1}{2} = 2.5$, $\frac{1}{2} + \frac{1}{2} + \frac{1}{2} + 1 = 2.5$ in $\frac{1}{2} + 1 + \frac{1}{2} + 1 + \frac{1}{2} = 3.5$.

Če izračunamo vsoto vseh razdalj v grafu iz prve tabele (kot v nalogi 1.22.) in vsoto vseh uteži, obakrat dobimo 16, Wienerjevo število grafa.

Opomba. Omenimo še eno interpretacijo Wienerjevega števila in uteži na grafu. Mislimo si, da naš graf predstavlja mesta, povezave pa ceste med njimi. Recimo, da je promet med vsakim parom vozlišč enak. Če je graf drevo, potem uteži $w(e)$ na povezavah preštejejo promet, uteži torej povedo, koliko je katera od povezav obremenjena. Kaj pa, če graf ni drevo? Potem si mislimo, da se bo promet enakomerno razdelil na vse najkrajše poti, kadarkoli jih je na razpolago več. Spet uteži $w(e)$ merijo obremenjenost povezav.

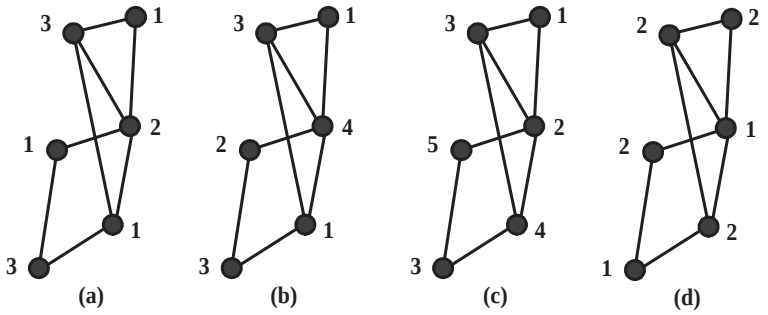
4

BARVANJE GRAFOV

4.1. Barvanja vozlišč

Začnimo z definicijami. Naj bo G graf brez zank. **Dobro k-barvanje** grafa G je dodelitev k barv vozliščem grafa G, tako da sosednja vozlišča dobijo različne barve. Če ima graf G dobro k-barvanje, rečemo, da je **k-pobarvljiv**. **Kromatično število** ali **barvnost** grafa G, $\chi(G)$, je najmanjše število k, za katero je G k-pobarvljiv.

V bolj matematičnem jeziku je k-barvanje preslikava, ki množici vozlišč priredi elemente iz množice barv. Barve lahko brez škode za splošnost označimo z zaporednimi števili 1,2,...,k. Torej, vsaka preslikava $c : V(G) \to \{1, 2, \ldots, k\}$ je k-barvanje (vozlišč) grafa G. Barvanje c je **dobro**, če velja: $v \sim u \Rightarrow c(u) \neq c(v)$.



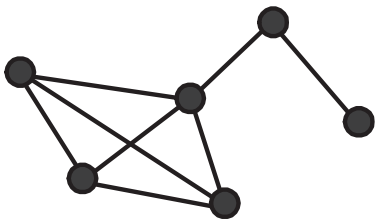
Slika 4.1: Različna barvanja.

Običajno k-barvanje predstavimo tako, da ob točke grafa napišemo števila 1, 2, 3, ..., k. Na slikah (a), (b) in (c) zgoraj so: dobro 3-barvanje, dobro 4-barvanje in dobro 5-barvanje grafa s šestimi vozlišči; barvanje na sliki (d) pa ni dobro, saj ena od povezav povezuje dve točki z barvo 2. Torej je $\chi(G) \leq 3$, ker ima G dobro 3-barvanje (slika 4.1(a)). Po drugi strani pa je $\chi(G) \geq 3$, ker so v G tri paroma sosednja vozlišča (trikotnik), ki morajo biti pobarvana različno. Torej $\chi(G) = 3$.

Gornja definicija je smiselna samo za grafe brez zank, saj krajišči nobene zanke ne moreta biti različnih barv. Lahko tudi predpostavimo, da v grafu ni večkratnih povezav, saj že ena povezava zadošča, da morata biti krajišči pobarvani različno in ostale povezave ne vplivajo več na barvanje. Zato se smemo pri obravnavi barvanj vozlišč omejiti na enostavne grafe.

Primer 1. Določimo barvnost grafa na sliki 4.2. Ker ima graf samo šest vozlišč, lahko pregledamo vse možnosti, in vidimo, da je $\chi(G) = 4$. Hitreje pa lahko sklepamo takole: V grafu so štiri vozlišča, med katerimi so vse povezave, zato za ta štiri vozlišča potrebujemo štiri različne barve.

Torej $\chi(G) \geq 4$. Ker za preostali dve vozlišči ne potrebujemo novih barv, lahko hitro najdemo 4-barvanje, torej $\chi(G) \leq 4$ in zato $\chi(G) = 4$.



Slika 4.2: Graf s kromatičnim številom 4.

Primer 2. Naj bo G poljuben dvodelen graf. Po definiciji obstajata množici vozlišč, tako da imajo vse povezave grafa po eno krajišče v eni, drugo pa v drugi množici. Torej lahko pobarvamo vozlišča prve množice z eno, vozlišča druge pa z drugo barvo in dobimo dobro barvanje. Dvodelni grafi (z vsaj eno povezavo) imajo torej barvnost 2.

V primerih zgoraj smo že uporabili naslednjo trditev:

IZREK 4.1. *Barvnost podgrafa je manjša ali enaka barvnosti grafa.*

Poln podgraf imenujemo **klika**. Moč največje klike v grafu G imenujemo **kličnо število** grafa G in ga označimo z $\omega(G)$.[1] Ker je vsaka klika podgraf, seveda za vsak graf velja:

IZREK 4.2. $\omega(G) \leq \chi(G)$.

Za mnoge grafe velja enakost $\omega(G) = \chi(G)$. Za polne grafe je seveda $\omega(K_n) = \chi(K_n) = n$. Naslednji primer je graf na sliki 4.2, ki vsebuje podgraf K_4, zato je $\chi(G) = \omega(G) = 4$.

Za lihe cikle pa velja $\omega(C_{2k+1}) = 2$ in $\chi(C_{2k+1}) = 3$, torej $\omega(G) < \chi(G)$. Izkaže se, da je razlika med kličnim številom in barvnostjo lahko poljubno velika. Obstajajo namreč družine grafov brez trikotnikov s poljubno velikim kromatičnim številom (glej nalogo 4.7.).

Iz izreka 4.2. sledi enostavna metoda za določitev spodnje meje za $\chi(G)$, pri kateri poiščemo največji poln podgraf grafa G. Metodo smo že nekajkrat uporabili na primerih.

Zgornja meja za $\chi(G)$ grafa z n vozlišči je seveda $\chi(G) \leq n$. Vendar je ta meja običajno zelo slaba, precej pa jo lahko izboljšamo, če poznamo največjo stopnjo vozlišča grafa, $\Delta(G)$, kot vidimo iz naslednjega izreka. (Glej tudi nalogo 4.8.)

IZREK 4.3. *Za enostaven graf G z največjo stopnjo točke $\Delta(G)$ je $\chi(G) \leq \Delta(G) + 1$.*

S precej več truda lahko dokažemo naslednji nekoliko močnejši izrek, ki ga je prvi dokazal L. Brooks

[1] Kliki rečemo tudi **skupek**. Nekateri avtorji poimenovanje klike uporabljajo samo za maksimalne polne podgrafe.

leta 1941.

IZREK 4.4. (BROOKSOV IZREK) *Naj bo G enostaven povezan graf z največjo stopnjo vozlišča $\Delta(G)$. Če G ni cikel z liho mnogo vozlišči in če G ni poln graf, potem je $\chi(G) \leq \Delta(G)$.*

Za ponazoritev Brooksovega izreka uporabimo graf s slike 4.2. Ker je v G podgraf K_4, smo že ugotovili $\chi(G) \geq 4$. Ker graf ustreza pogojem Brooksovega izreka (ni lih cikel, ni poln graf in $\Delta(G) = 4$), je po Brooksovem izreku $\chi(G) \leq 4$. Torej je $\chi(G) = 4$.

Na žalost Brooksov izrek ni vedno tako uporaben. Če graf G vsebuje majhno število vozlišč visoke stopnje, potem je lahko zgornja meja iz Brooksovega izreka poljubno slaba. Za polni dvodelni graf $K_{1,100}$ na primer vemo, da je $\chi(G) = 2$, Brooksov izrek pa nam da zgornjo mejo $\chi(G) \leq 100$.

4.2. Algoritmi za barvanje vozlišč

Znano je, da je problem barvanja vozlišč grafa NP-težek. Verjetno zato ne obstaja noben učinkovit algoritem za optimalno barvanje.

V prejšnjih razdelkih smo spoznali nekaj trikov, s katerimi lahko določimo barvnost grafa v posebnih primerih. Zdaj si bomo pogledali nekaj algoritmov. Najprej algoritem, ki izračuna kromatično število, vendar je njegova časovna zahtevnost eksponentna.[2] Za večje grafe ni smiselno uporabljati algoritma z eksponentno časovno zahtevnostjo, zato si bomo ogledali še primere algoritmov, ki so hitrejši, vendar je rezultat, ki ga dobimo, samo ocena (spodnja ali zgornja meja) za kromatično število.

Algoritem Zykova

Za ne prevelike grafe lahko uporabimo algoritem Zykova, ki urejeno pregleda vse zanimive možnosti. Možnosti je seveda veliko, a za grafe z ne preveč vozlišči je algoritem mogoče izvesti. Algoritem Zykova temelji na naslednjem. Če je graf poln, potem je njegova barvnost enaka številu vozlišč n. Če graf ni poln, potem sta v njem vsaj dve nepovezani vozlišči, označimo ju z u in v. Zagotovo velja vsaj ena od naslednjih dveh možnosti:

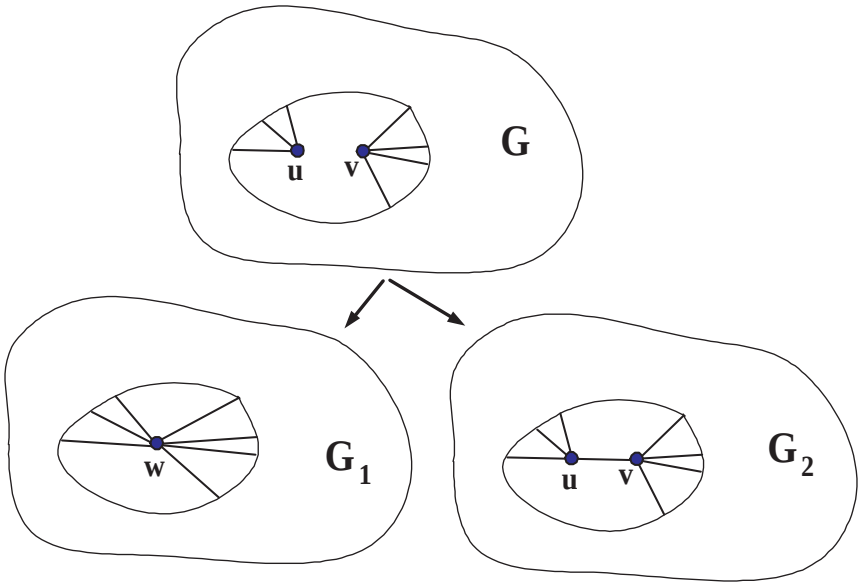
(1) V optimalnem barvanju sta ti dve vozlišči enako pobarvani.

(2) V optimalnem barvanju sta ti dve vozlišči različno pobarvani.

Naredimo dva nova grafa.

V prvem primeru lahko vozlišči identificiramo. Formalno to naredimo takole: naredimo nov graf G_1 z množico vozlišč $V(G_1) = V(G) - \{u, v\} \cup \{w\}$ (namesto vozlišč u in v grafu dodamo novo vozlišče w). Sosedi vozlišča w v novem grafu so vsi sosedi vozlišč u in v, torej $N(w) = N(u) \cup N(v)$. Vsaj eno

[2] Takim algoritmom rečemo tudi **eksaktni** ali **optimalni**. Ker je naloga NP-težka, seveda ni presenetljivo, da ima ta algoritem nepolinomsko časovno zahtevnost. Glej tudi poglavje 5.



Slika 4.3: Algoritem Zykova: G, G_1 in G_2.

optimalno barvanje grafa G je tudi optimalno barvanje grafa G_1; če definiramo $c(w) = c(u) = c(v)$, ali natančneje $c_1 : V(G_1) \to C$, $c_1(w) := c(u) = c(v)$ in $c_1(x) = c(x)$ za $x \in V(G_1) - w$. Torej $\chi(G) = \chi(G_1)$.

V drugem primeru vozlišči povežemo, dobimo torej nov graf z isto množico vozlišč $V(G_2) = V(G)$ in eno dodano povezavo $E(G_2) = E(G) \cup \{uv\}$. V tem primeru je vsaj eno optimalno barvanje grafa G tudi optimalno barvanje grafa G_2, ker je $c(u) \neq c(v)$. (Tokrat lahko za c_2 vzamemo kar isto preslikavo, c.) Torej $\chi(G) = \chi(G_2)$.

Ker ne vemo, katera možnost je prava, postopamo takole. V G izberemo dve nepovezani vozlišči. Naredimo grafa G_1 in G_2. Ker velja

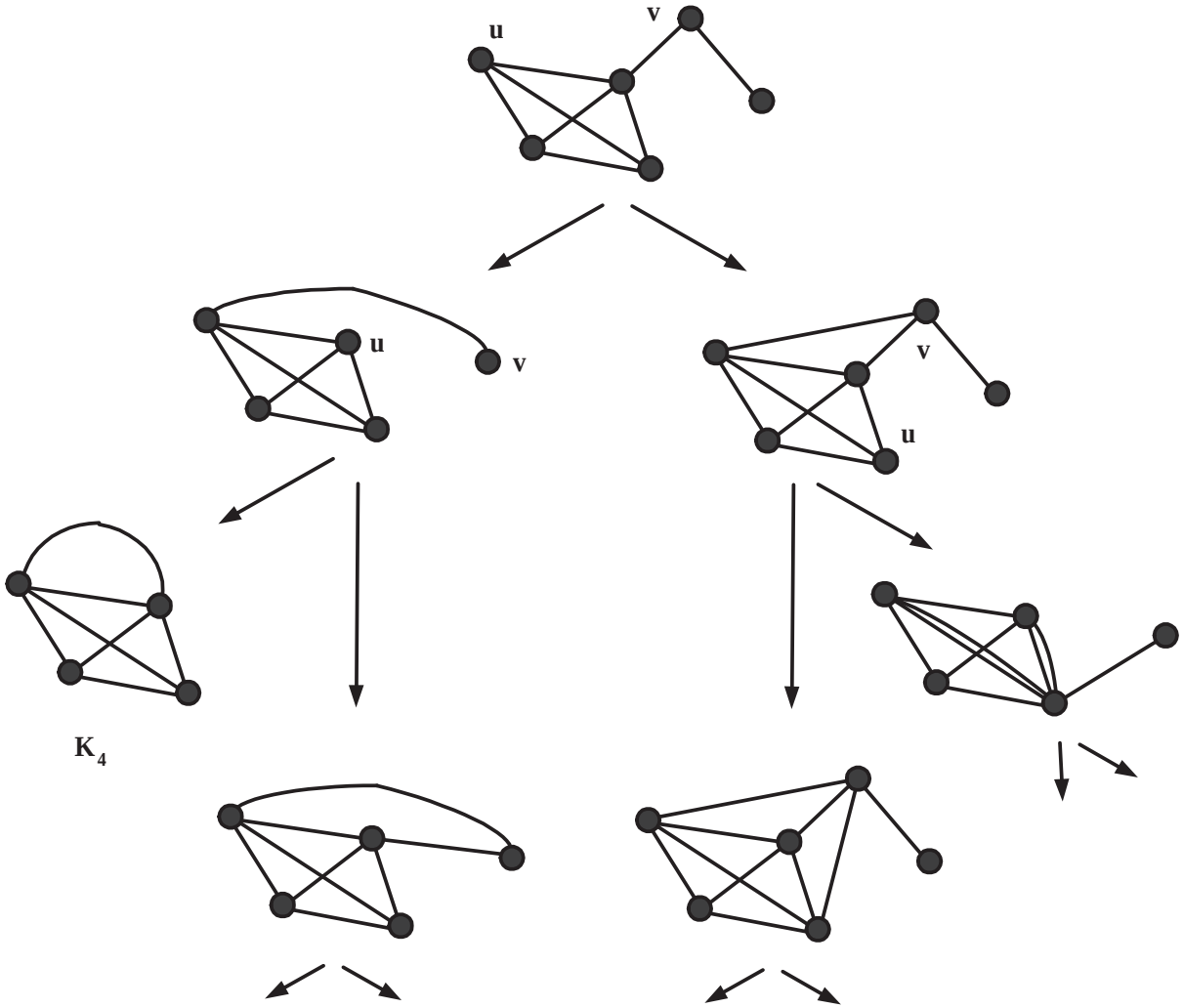
$$\chi(G) = \min\{\chi(G_1), \chi(G_2)\} \ ,$$

nadaljujemo z določanjem barvnosti G_1 in G_2. Če je graf G_i poln, potem vemo, da je $\chi(G_i) = |V(G_i)|$, sicer pa sta v grafu G_i dve nepovezani točki, in korak ponovimo (spet se računanje lahko razveji na dve veji!)[3].

Poglejmo delovanje algoritma na primeru grafa na 6 vozliščih (s slike 4.2). V eni od vej smo naleteli na K_4, v tej veji je izvajanje zaključeno. Ostale tri veje bi lahko razvejevali naprej. Na osnovi narisanega na sliki 4.4 lahko sklepamo takole: V vsaki od vej imamo graf s polnim podgrafom na

[3]Število korakov (razvejitev) je lahko v najslabšem primeru skoraj toliko kot je nepovezanih parov točk, torej $O(n^2)$ (glej definicijo $O(.)$ na strani 100.) Ob vsaki pravi razvejitvi moramo pogledati dva grafa, vseh obravnavanih grafov je zato v najslabšem primeru reda $O(2^{n^2})$. Algoritem ima torej eksponentno časovno zahtevnost, glej poglavje 5. Pri vsaki razvejitvi sta nova grafa na nek način *enostavnejša*: ali se je zmanjšalo število vozlišč ali pa je graf bolj poln, barvnost polnega grafa pa je preprosto določiti. Algoritem Zykova je primer splošne metode, ki jo lahko poimenujemo **Deli in vladaj**. (Glej tudi [J.Kozak, Deli in vladaj z objektnim programiranjem, Obzornik za matematiko in fiziko 10 (1993) 49-61.]

4 vozliščih, zato je zagotovo $\chi(G) \geq 4$. Ker je ena od vej zaključena s K_4 vemo, da je $\chi(G) \leq 4$. Torej $\chi(G) = 4$. (To seveda znamo videti tudi hitreje, glej primer na strani 78.)



Slika 4.4: Nekaj korakov algoritma Zykova.

Postopki zaporednega barvanja

V primerih, ko je graf prevelik za iskanje optimalne rešitve, se moramo zadovoljiti z iskanjem približnih rešitev. Tu si poglejmo preprost postopek, pravzaprav družino algoritmov, ki jih imenujemo **postopki zaporednega barvanja**. Osnovna ideja je zelo enostavna:

ponavljaj: izberi vozlišče, ki še ni pobarvano, in ga pobarvaj z eno od barv, ki je še prosta.

Da dobimo dobro definiran algoritem, moramo določiti, kako izbiramo naslednje vozlišče, in kako izberemo barvo zanj. Barva je v vozlišču v **prosta**, če nobeno vozlišče, sosednje v, ni bilo doslej pobarvano z njo. Ni težko videti (glej nalogo 4.8.), da velja

IZREK 4.5. *S postopkom zaporednega barvanja graf G z največjo stopnjo točke $\Delta(G)$ dobro pobarvamo z največ $A(G) \leq \Delta(G) + 1$ barvami.*

Opomba. Posledica izreka 4.5. je izrek 4.3.

Za konec si oglejmo dve verziji postopka

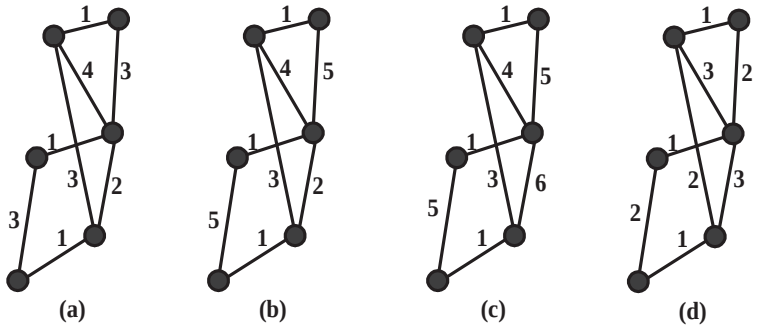
Welsh-Powellov algoritem. Uredimo vozlišča po padajočih stopnjah. Množica barv: $C = \{1, 2, \ldots, \Delta(G) + 1\}$. Vsakič izberemo minimalno prosto barvo.

Postopek zaporednega barvanja z naključno izbiro. Izberemo katerokoli ureditev vozlišč. Množica barv: $C = \{1, 2, \ldots, \Delta(G) + 1\}$. Vsakič izberemo (z enako verjetnostjo) katerokoli prosto barvo.

Opomba. Za katerikoli graf obstaja vsaj en vrstni red zaporednega barvanja vozlišč, pri katerem postopek zaporednega barvanja uporabi $\chi(G)$ barv. Seveda pa ni zelo verjetno, da bomo med $n!$ različnimi vrstnimi redi izbrali ravno enega od optimalnih.

4.3. Barvanja povezav

Začnimo z definicijami. Naj bo G graf brez zank. **Dobro k-barvanje povezav** grafa G je dodelitev k barv povezavam grafa G, tako da noben par povezav s skupnim krajiščem ni pobarvan z isto barvo. Če ima graf G dobro k-barvanje povezav, potem rečemo, da je **po povezavah k-pobarvljiv**. Najmanjše število k, za katerega obstaja dobro k-barvanje povezav grafa G, imenujemo **kromatični indeks** grafa G in ga označimo z $\chi'(G)$.



Slika 4.5: Graf s kromatičnim indeksom 4.

Barvanje povezav običajno predstavimo z risbo, na kateri ob povezavah zapišemo števila 1, 2, 3,

$\ldots, k$. Na slikah (a), (b) in (c) so dobra 4-, 5- in 6-barvanja povezav grafa G. Barvanje na sliki (d) ni dobro, ker imata dve povezavi barve 3 skupno krajišče. Torej je $\chi'(G) \leq 4$, ker ima G dobro 4-barvanje povezav (slika 4.5(a)). Po drugi strani pa mora biti $\chi'(G) \geq 4$, ker so v G štiri povezave z istim krajiščem (vozliščem stopnje 4), ki morajo vse dobiti različne barve. Torej $\chi'(G) = 4$.

V definiciji smo zahtevali, da graf nima zank, saj morajo biti vse povezave, ki se srečajo v istem vozlišču, različno pobarvane. Po drugi strani pa pogosto želimo obravnavati grafe z večkratnimi povezavami, ker te pomembno vplivajo na vrednost kromatičnega indeksa.

Obstaja očitna spodnja meja za $\chi'(G)$: $\chi'(G) \geq d$. Če ima G m povezav, potem je očitno $\chi'(G) \leq m$. Vendar pa je ta zgornja meja zelo slaba. Bistveno boljši meji sta našla V.G. Vizing in C.E. Shannon. Za enostavne grafe je Vizing leta 1963 dokazal naslednji zelo močan rezultat, ki ga tu navajamo brez dokaza:

IZREK 4.6. (VIZINGOV IZREK) *Za enostavni graf G z največjo stopnjo točke $\Delta(G)$ je*

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1 \ .$$

Ta znameniti rezultat pove, da je za enostavni graf G kromatični indeks enak $\Delta(G)$ ali $\Delta(G) + 1$. Na ta način lahko razdelimo vse enostavne grafe v dva razreda – tiste, za katere je $\chi'(G) = \Delta(G)$ in tiste z $\chi'(G) = \Delta(G)+1$. Obstajajo eni in drugi grafi, v splošnem pa ni znano, kateri grafi spadajo v kateri razred[4]. Za nekatere vrste grafov je odgovor na dlani. Za cikle C_n ($n \geq 3$) na primer velja:

$\chi'(C_n) = 2$, *če je* n *sod in* $\chi'(C_n) = 3$, *če je* n *lih.*

Podoben rezultat velja za polne grafe K_n: Za poln graf K_n je

$\chi'(K_n) = n - 1$, *če je* n *sod, in* $\chi'(K_n) = n$, *če je* n *lih.*

(Glej naloge 4.11. in 4.12.)

Opomba. Zanimiva je tudi naslednja kombinacija kromatičnega števila in kromatičnega indeksa, ki jo imenujemo totalno kromatično število grafa (angl. total chromatic number), in definiramo takole: **totalno barvanje grafa** je preslikava, ki elementom množice $V(G) \cup E(G)$ priredi barve; **dobro totalno barvanje grafa** sosednjim elementom priredi različne barve. Povezavi sta sosednji, če imata skupno krajišče. Vozlišče in povezava sta sosednji, če je vozlišče krajišče povezave. (In seveda: vozlišči sta sosednji, če sta krajišči iste povezave.) **Totalno kromatično število grafa** $\chi''(G)$ je minimalno število barv, za katerega obstaja dobro totalno barvanje. Očitno velja $\Delta(G) \leq \chi''(G)$, za zgornjo mejo pa je že nekaj časa odprta *Behzad-Vizingova domneva*

$$\chi''(G) \leq \Delta(G) + 2 \ ?$$

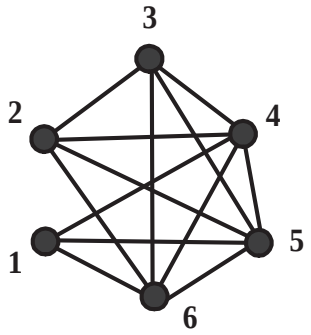
[4]Odločitveni problem, ali je za dani graf $\chi'(G) = \Delta(G)$ ali $\chi'(G) = \Delta(G) + 1$ je NP-poln.

4.4. Primeri uporabe barvanja

Problemi barvanja grafov so med najbolj raziskovanimi nalogami diskretne matematike. Razlogov za to je več, podobno kot za problem trgovskega potnika velja, da so naloge barvanja teoretično zanimive, ker so težke, kljub temu, da jih je mogoče preprosto zastaviti[5]. Po drugi strani so naloge barvanja praktično zanimive zaradi mnogih uporab. Tu si bomo ogledali nekaj preprostih primerov.

Skladišče kemikalij. Zamislimo si naslednjo nalogo, s katero se utegnemo srečati pri načrtovanju skladišča kemikalij. Vemo, da nekaterih parov kemikalij zaradi varnosti ne smemo skladiščiti v istem prostoru. Recimo, da imamo seznam kemikalij, ki jih bo treba skladiščiti. Najmanj koliko prostorov moramo predvideti v načrtu, da bomo vedno lahko varno skladiščili vse kemikalije?

Preprosta naloga je na primer takšna. Vemo, da naslednje skupine kemikalij nikakor ne smemo skladiščiti v skupnem prostoru zaradi nevarnosti burne reakcije. Zapišimo kar oznake (številke), ki so napisane na sodih: $\{1, 4, 5, 6\}$, $\{2, 3, 6\}$, $\{2, 3, 4, 5\}$. Na sliki 4.6 je graf z vozlišči $\{1, 2, 3, 4, 5, 6\}$, povežemo pa vse pare kemikalij, ki so vsaj v eni od prepovedanih kombinacij. Za vsako kemikalijo



Slika 4.6: Graf s skupinami nevarnih kombinacij kemikalij.

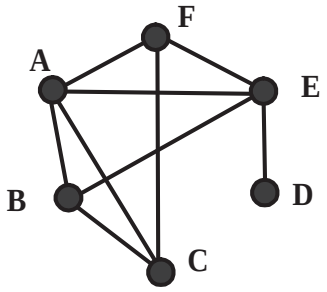
moramo povedati, v kateri prostor skladišča ga bomo shranili. Imenujmo prostore rdeča soba, modra soba, in tako naprej. Koliko sob (barv) potrebujemo? Ker je barvnost grafa na sliki enaka pet (bralec naj to sam preveri), potrebujemo (vsaj) pet sob.

Naloge razporejanja. Druga zelo pogosta uporaba barvanja so naloge razporejanja, na primer sestavljanje urnika ali razporeda izpitov.

Urnik. Poglejmo tale primer. V kreditnem sistemu študija so na voljo predavanja iz predmetov A, B, C, D, E in F. Zaradi enostavnosti predpostavimo, da imamo samo pet študentov, ali bolj realno, da imamo pet skupin (smeri): Študent (skupina) 1 bi želel (ali mora po programu) poslušati predmete A, B in C. Študent (skupina) 2 bi želel (ali mora po programu) poslušati predmete A, C in F. Študent (skupina) 3 bi želel (ali mora po programu) poslušati predmete E in F. Študent (skupina) 4 bi želel (ali mora po programu) poslušati predmete E in D. Študent (skupina) 5 bi želel (ali mora po programu) poslušati predmete A, E in F. Recimo, da so predavanja enkrat na teden popoldan. Narediti je treba urnik. (Določiti, kateri dan bo predavanje iz predmeta X.) Ali je pet dni dovolj? Kaj pa če predavanj ne moremo ali nočemo organizirati v petek in v ponedeljek?

[5] Tako je na primer „problem štirih barv“, ki smo ga že omenili na strani 25 postal eden od najslavnejših odprtih problemov matematike 20. stoletja.

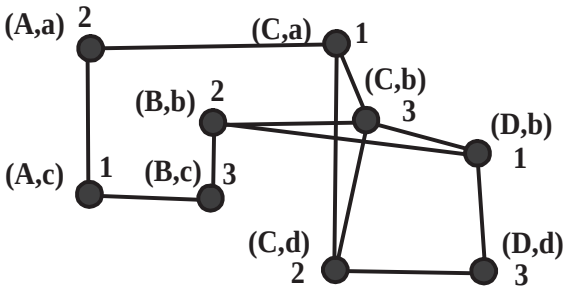
Narišimo graf z vozlišči A, B, C, D, E in F (slika 4.7). Povežimo tiste pare vozlišč, za katere nočemo, da so na urniku hkrati, ker obstaja študent (skupina), ki želi poslušati oba predmeta. Zdaj lahko sklepamo takole: vozlišča A, B in C so paroma povezana, zato morajo biti na urniku



Slika 4.7: Primer urnika s šestimi predmeti.

tri različne dneve. Na primer A prvi dan (recimo, v ponedeljek), B drugi dan (torek) in C tretji dan. F ne sme biti na urniku niti prvi niti tretji dan (zaradi povezav AF in CF), lahko pa je drugi dan. E lahko damo (na primer) na četrti dan, D pa na peti dan. Kaj smo naredili: vozliščem smo priredili dneve. Ali drugače povedano, poiskali smo 5-barvanje grafa. Ker je barvnost grafa v tem primeru enaka 3, lahko urnik naredimo tudi če imamo na voljo samo tri dneve.

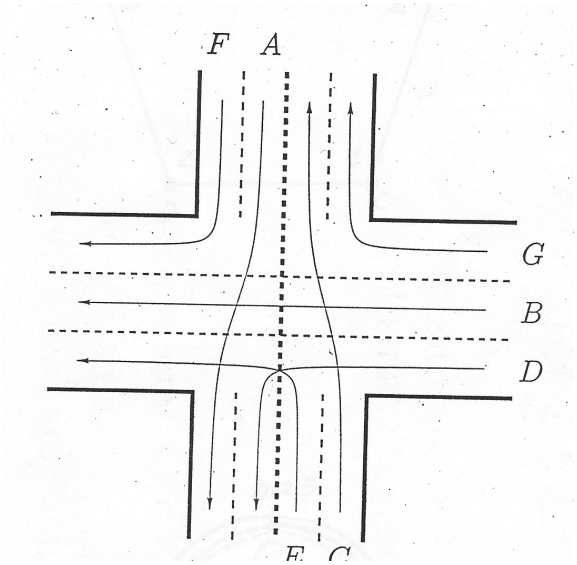
Razporeditev strojev na gradbišča. Tudi drugje najdemo podobne naloge. Na primer, gradbeno podjetje ima hkrati aktivnih več gradbišč, in omejeno število zelo dragih specializiranih strojev. Zaradi težav z izpeljavo prevoza lahko stroj isti dan dela največ na enem gradbišču. Na vsakem od gradbišč je omejeno število usposobljenih operaterjev, zato lahko učinkovito uporabijo v istem dnevu na vsakem gradbišču samo en stroj. Vemo, da stroj A potrebujejo na gradbiščih a in c, stroj B na gradbiščih b in c, stroj C na gradbiščih a, b in d, in stroj D na gradbiščih b in d. Koliko dni najmanj je potrebno in kako je treba razporediti stroje in gradbišča po dnevih. Zdaj postopajmo takole: vozlišča grafa naj bodo pari (X, y), kjer je X stroj, y pa gradbišče. Vozlišči povežimo, če je ena koordinata enaka. Graf je na sliki 4.8. Ker je barvnost grafa enaka 3, vemo, da lahko vsa dela izvedemo v treh dneh. (Eno od dobrih 3-barvanj je označeno na sliki.)



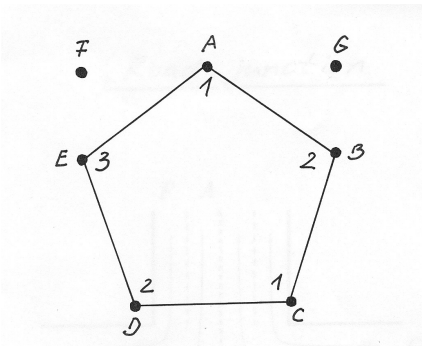
Slika 4.8: Stroji in gradbišča.

Semaforizacija križišča. Skozi križišče na sliki poteka promet v smereh A, B, C, D, E, F in G. V nekaterih parih smeri lahko promet poteka hkrati brez nevarnosti, nekateri pari smeri pa niso

neodvisni. Pri semaforizaciji križišča taki pari ne smejo imeti hkrati zelene luči.

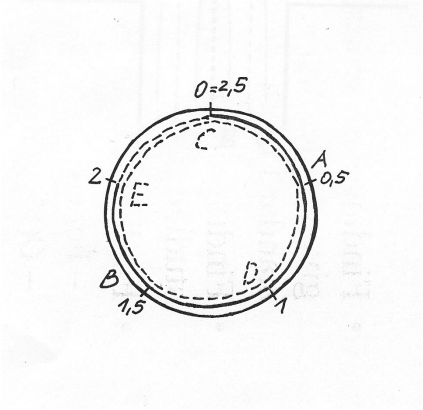


V našem primeru dobimo graf:



ki ga lahko pobarvamo s tremi barvami. Vozliščem, pobarvanim z isto barvo, hkrati dovolimo vožnjo skozi križišče. Vsako 3-barvanje nam da eno možno rešitev. Boljše rešitve dobimo z bolj natančnim modeliranjem problema. Tako na primer lahko upoštevamo prometnost posameznih smeri in bolj prometnim smerem dodelimo daljše časovne intervale. Če privzamemo, da so vsi intervali enako dolgi, dobimo iz 3-barvanja rešitev, v kateri ima vsaka od smeri zeleno luč tretjino časa. V našem primeru lahko dobimo boljši izkoristek časa, na primer z naslednjo delitvijo časa, kjer je vsaka od smeri odprta vsaj 2/5 časa[6]. (Krožnico z obsegom 5/2 razdelimo na 5 enakih delov. Smer A je potem odprta prvi dve enoti (od 0 do 1), smer B (od 1 do 2), smer C (od 2 do 0.5), smer D (od 0.5 do 1.5), in smer E (od 1.5 do 2.5). Smeri F in G sta neodvisni od ostalih in sta lahko odprti ves čas.)

[6]Ta rešitev ustreza optimalnemu cikličnemu barvanju grafa (angl. circular chromatic number), saj je cikl. kromatično število cikla C_5 enako $\frac{5}{2}$.



Dodeljevanje frekvenc. Omejenost frekvenčnega spektra je bila v času najhitrejšega povečevanja števila uporabnikov ena od glavnih težav pri načrtovanju omrežij v brezžični telefoniji. V sodobnih mobilnih sistemih se uporablja celični sistem, pri katerem so velika področja razdeljena na več manjših, imenovanih *celice*. Visoko zmogljiv oddajnik je nadomeščen z več oddajniki manjše moči, ki pokrivajo le majhen del celotnega območja danega sistema, kar omogoča večkratno uporabo istih frekvenc. Vsaka celica vsebuje eno glavno postajo, ki za komunikacijo z mobilnimi telefoni v svoji celici uporablja del sistemu dodeljenih kanalov. Kanali so bili prvotno kar frekvence, zaradi velikih zahtev pa so kasneje začeli uporabljati boljše metode, na primer dodeljevanje iste frekvence več pogovorom z deljenjem časa. Sosednjim postajam so dodeljene množice kanalov, ki so med seboj disjunktne, s čimer se izognemo nedovoljeni interferenci. Če je v celici na voljo prost kanal, je z mobilnim telefonom mogoče telefonirati, sicer mora sistem klic zavrniti. Torej je zaželjeno, da je dovolj prostih kanalov v celicah, v katerih pričakujemo več uporabnikov.

Dobre modele za problem dodeljevanja kanalov v celični telefoniji dobimo s pomočjo teorije grafov. Celične mreže so ponavadi predstavljene kot uteženi grafi, kjer točke predstavljajo območja oziroma *celice*, ki jih pokrivajo posamezne glavne postaje, povezave predstavljajo možnosti frekvenčne interference, vektor uteži pa predstavlja zahteve posamezne celice oziroma število klicev, ki morajo biti oskrbljeni v celici.

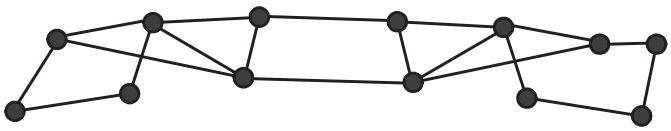
Poenostavljeni model lahko z vpeljavo dodatnih zahtev in definicijo primernih namenskih funkcij približamo nalogam iz prakse. Ker je že osnovna naloga posplošitev naloge barvanja grafov, so optimizacijske naloge NP-težki problemi. V literaturi tako najdemo mnoge različice nalog in mnoge hevristične algoritme za njihovo reševanje. (Glej tudi naslednja poglavja.)

Po drugi strani je poenostavljeni model motiviral zanimive matematične probleme. Zaradi enostavnosti, v nekaterih primerih pa tudi praktične uporabnosti, so mnogi matematiki raziskovali lastnosti grafov, ki jih dobimo iz pokritja ravnine s šestkotnimi celicami. Grafom, katerih točke so šestkotne celice, povezave pa povezujejo sosednje celice, rečemo **heksagonalni grafi**. Na videz preprosta naloga je dodeliti vsaki celici množico kanalov zahtevane moči tako, da ne smemo enakih barv dodeliti dvema sosednjima točkama. Za celice, ki niso sosednje, nimamo nobenih omejitev glede barv. Nalogo lahko interpretiramo kot nalogo barvanja vozlišč grafa, v katerem vozlišče z utežjo w nadomestimo s polnim grafom z w vozlišči in utežmi 1, ali pa govorimo o večbarvanju, to je barvanju vozlišč grafa z množicami barv (velikosti $w(v)$). Izkaže se, da je tudi ta problem večbarvanja uteženih heksagonalnih grafov NP-poln[7].

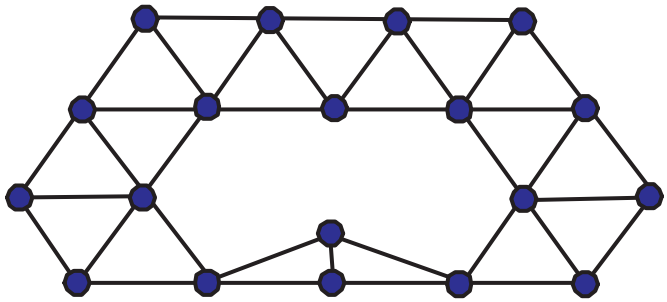
[7]Podrobnosti glej npr. v [P.Šparl, J.Žerovnik, Primer uporabe teorije grafov v mobilni telefoniji, Obz. mat. fiz.

4.5. Naloge z rešitvami

4.1. Določi barvnost (kromatično število) naslednjega grafa

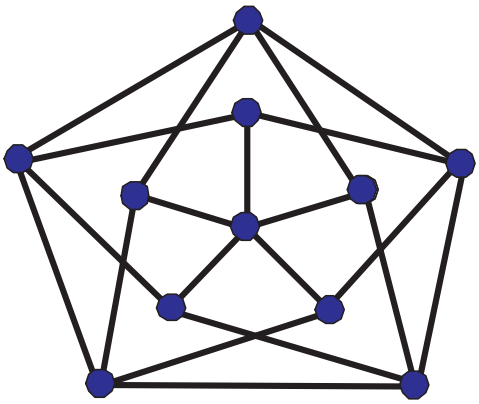


4.2. Določi barvnost (kromatično število) naslednjega grafa



4.3. Določi barvnost Petersenovega grafa.

4.4. Določi barvnost Grötzschevega grafa (slika 4.9).



Slika 4.9: Grötzschev graf.

49 (2002) 10-20.] in reference tam.

4.5. Dokaži, da je barvnost podgrafa manjša ali enaka barvnosti grafa.

4.6. Določi barvnost kartezičnega produkta (a) $P_3 \Box P_4$, (b) $K_3 \Box K_4$, (c) $C_3 \Box C_5$, (d) $C_4 \Box C_6$, (e) $K_4 \Box C_5$.

4.7. Ali obstajajo grafi brez trikotnikov s poljubno velikim kromatičnim številom?

4.8. Dokaži izrek: S postopkom zaporednega barvanja graf G z največjo stopnjo točke $\Delta(G)$ dobro pobarvamo z največ $A(G) \leq \Delta(G) + 1$ barvami.

4.9. Izberi poljuben graf in poišči vrstni red vozlišč, pri katerem bo Welsh-Powellov postopek zaporednega barvanja:

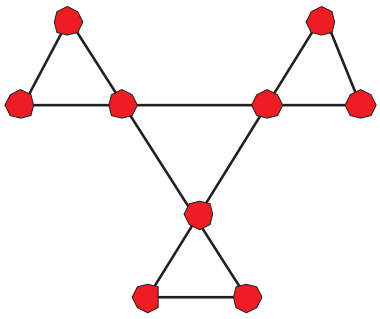
(a) našel optimalno barvanje,

(b) uporabil $\Delta(G) + 1$ barv.

4.10. Dokaži Brooksov izrek.

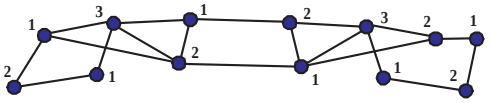
4.11. Dokaži, da za poln graf K_n velja: $\chi'(K_n) = n - 1$, če je n sod, in $\chi'(K_n) = n$, če je n lih.

4.12. Določi kromatični indeks grafa

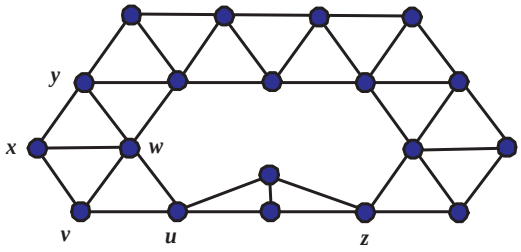


Rešitve

4.1. Ker so v grafu trikotniki, je $\chi(G) \geq 3$. dobro barvanje s tremi barvami na sliki je torej optimalno.



4.2. V grafu so trikotniki, torej $\chi(G) \geq 3$. Pokažimo, da ni nobenega dobrega barvanja s tremi barvami. Izberimo si trikotnik, na primer trikotnik uvw, zanj potrebujemo tri različne barve, recimo 1,2 in 3. Če imamo na voljo samo tri barve in če hočemo, da bo barvanje dobro, potem mora biti barva vozlišča x enaka barvi u, barva vozlišča y enaka barvi v, in tako dalje. Tako vidimo, da morata v vsakem dobrem barvanju s tremi barvami imeti vozlišči u in z *različni* barvi. Takega barvanja pa ne moremo dopolniti do dobrega 3-barvanja zaradi obeh skupnih sosed vozlišč u in z. Zato sklepamo, da ni nobenega dobrega 3-barvanja. Ker ni težko najti 4-barvanja, je $\chi(G) = 4$.



4.3. Petersenov graf (slika 1.18) ima podgraf C_5, zato je barvnost vsaj enaka 3. Po Brooksovem izreku je barvnost manjša ali enaka 3. Torej je barvnost Petersenovega grafa enaka 3.

4.4. Za zunanji 5-cikel potrebujemo vsaj 3 barve. Če za hip odstranimo vozlišče na sredini, vidimo, da tudi za notranjih pet vozlišč potrebujemo vsaj 3 barve. (V grafu brez vozlišča na sredini ima vsako vozlišče na zunanjem ciklu po en „par“, vozlišče v sredini grafa, ki ima iste sosede. Če bi notranja vozlišča lahko (dobro) pobarvali z manj kot tremi barvami, bi lahko iste barve uporabili za pare na zunanjem ciklu.)

Za vozlišče na sredini zato potrebujemo četrto barvo..

4.5. Naj bo H poljuben podgraf grafa G in $c : V(G) \to \{1, 2, \ldots, \chi(G)\}$ dobro barvanje grafa G. Definirajmo preslikavo $\tilde{c} : V(H) \to \{1, 2, \ldots, \chi(G)\}$ s predpisom $\tilde{c}(v) = c(v)$ za $v \in V(H)$. Očitno je $\tilde{c}$ dobro barvanje (saj $v \sim_H u \Longrightarrow v \sim_G u \Longrightarrow c(u) \neq c(v) \Longrightarrow \tilde{c}(u) \neq \tilde{c}(v)$). torej je $\chi(H) \leq \chi(G)$.

4.6. (a) 2, (b) 4, (c) 3, (d) 2, (e) 4.

4.7. Obstajajo. Eno od družin dobimo iz K_2 z naslednjo konstrukcijo Mycielskega. Naj bo G_i graf brez trikotnikov z barvnostjo i. Graf G_{i+1} konstruiramo takole: podvojimo vsako vozlišče grafa G_i in povežemo vsako novo vozlišče z vsemi sosedi originala. Novih vozlišč med sabo ne povežemo z nobeno povezavo. (Ker so nova vozlišča kopije, potrebujemo za barvanje novih vozlišč vse barve, torej vseh i barv.) Dodamo še

eno vozlišče, ki ga povežemo z vsemi novimi vozlišči. Za novo vozlišče očitno potrebujemo novo barvo, zato je barvnost G_{i+1} enaka $i+1$. Ni težko videti, da se klično število pri konstrukciji ne poveča. Če je bil G_i brez trikotnikov, potem je brez trikotnikov tudi G_{i+1}. (Podrobnosti tu opuščamo.)

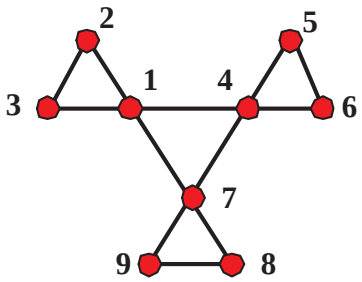
Če začnemo z $G_1 = K_2$, dobimo $G_2 = C_5$, G_3 Grötzschev graf (slika 4.9), in tako naprej, grafe brez trikotnikov s poljubno velikim kromatičnim številom.

4.8. Vzemimo poljuben postopek zaporednega barvanja (poljuben vrstni red izbire vozlišč, poljubna izbira barve), za množico barv pa izberimo $C = \{1, 2, \ldots, \Delta(G)+1\}$.

Trdimo, da po vsakem koraku algoritma velja: *graf G je dobro delno pobarvan*[8]. Pred prvim izvajanjem zanke trditev velja. Recimo, da je graf dobro delno pobarvan in da smo izbrali novo vozlišče v. Sosedov v je največ $\Delta(G)$, (nekateri med njimi so že pobarvani, drugi ne,) zato je med $\Delta(G)+1$ barvami vsaj ena prosta. Izbira katerekoli od njih in razširitev barvanja z njo očitno ohrani gornjo lastnost.

4.9. Prvi primer: Sodi cikel. Vrstni red, pri katerem Welsh-Powellov postopek uporabi dve barvi, je kar zaporedno barvanje vzdolž cikla. Če začnemo barvanje z dvema točkama na razdalji tri, pa Welsh-Powellov postopek uporabi tri barve.

Drugi primer. Vozlišča grafa na sliki barvamo



prvič v zaporedju 1, 2, 3, 4, 5, 6, 7, 8, 9 (in pobarvamo graf optimalno, s tremi barvami) drugič v zaporedju 9, 6 in 3 (vsa tri vozlišča dobijo barvo 1), 8, 5 in 2 (vsa tri vozlišča dobijo barvo 2), 7, 4 in 1 (ta tri vozlišča moramo pobarvati z različnimi barvami: 3,4,5).

4.10.

Za $\Delta(G) = \Delta = 0, 1, 2$ izrek očitno velja. Zato predpostavimo, da velja $\Delta \geq 3$. Recimo, da izrek ne velja. Potem obstaja graf, ki ustreza pogojem izreka, za katerega je $\chi(G) = \Delta(G)+1$. Naj bo G tak graf z minimalnim številom vozlišč.

V grafu G izberimo poljubno vozlišče v_0 in označimo $G' = G \setminus v_0$. Stopnja v_0 mora biti enaka Δ, saj bi v nasprotnem primeru lahko zaradi minimalnosti G' lahko dobro pobarvali G' z Δ barvami, zaradi $\mathrm{st}(v_0) < \Delta$ pa bi obstajala prosta barva, s katero bi lahko pobarvali v_0, v nasprotju s predpostavko, da je $\chi(G) = \Delta + 1$.

[8] **Dobro delno barvanje** je preslikava c iz podmnožice $U \subseteq V(G)$ v množico barv C, pri čemer velja $u \sim v \Rightarrow c(u) \neq c(v)$ za vse pare vozlišč u, v iz U.

V grafu G' veljajo naslednje trditve:

Trditev 1: V vsakem dobrem Δ-barvanju grafa G' so vsi sosedje vozlišča v_0 pobarvani različno. Dokaz: Če sta dva soseda pobarvana enako, potem obstaja prosta barva za v_0 in Δ-barvanje lahko razširimo na G. Protislovje. □

Označimo sosede vozlišča v_0 z $v_1, v_2, \ldots, v_\Delta$ in recimo, da je G' pobarvan tako, da je barva v_k enaka k. Označimo z $G(i,j)$ podgraf v G' induciran z vozlišči, pobarvanimi z barvama i in j.

Trditev 2: Vozlišči v_i in v_j sta v isti komponenti povezanosti grafa $G(i,j)$. Dokaz: Če to ne bi bilo res, bi lahko v komponenti, ki vsebuje v_i zamenjali barvi i in j in tako dobili dobro Δ-barvanje grafa G', v katerem bi imeli vozlišči v_i in v_j isto barvo. To pa po Trditvi 1 ni mogoče. □

Označimo s C_{ij} komponento povezanosti grafa $G(i,j)$, v kateri sta v_i in v_j.

Trditev 3: C_{ij} je pot. Dokaz: v_i ima v C_{ij} natanko enega soseda barve j. V nasprotnem primeru bi lahko v_i zamenjali barvo in prišli v nasprotje s Trditvijo 1. Označimo soseda v_i v C_{ij} z u_1. u_1 ima v C_{ij} natanko dva soseda barve i (v_i in u_2). V nasprotnem primeru bi lahko u_1 in v_i zamenjali barvi in prišli v nasprotje s Trditvijo 1. In tako dalje. u_k ima v C_{ij} natanko dva soseda druge barve (v_{k-1} in u_{k+1}). V nasprotnem primeru bi lahko u_k, $u_{k-1} \ldots v_i$ zamenjali barve in prišli v nasprotje s Trditvijo 1. Slej ko prej (zaradi Trditve 2) s potjo v_i, u_1, $u_2 \ldots$ pridemo do v_j. Torej je C_{ij} pot med v_i in v_j. □

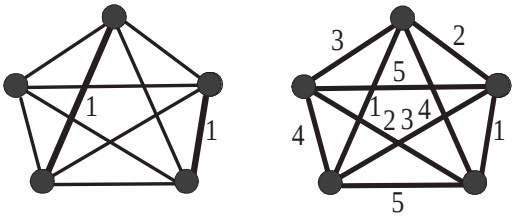
Trditev 4: $V(C_{ij}) \cap V(C_{ik}) = \{v_i\}$ Dokaz: Recimo, da se poti C_{ij} in C_{ik} sekata še v kakšnem vozlišču $u_k \neq v_i$. Potem ima v Δ-barvanju G' vozlišče u_k barvo i in po dve sosedi pobarvani z barvama j in k. Torej obstaja prosta barva v soseščini vozlišča u_k. Potem pa lahko vozlišče u_k prebarvamo s to prosto barvo in dobimo dobro Δ-barvanje grafa G', v katerem C_{ij} ni pot, kar nasprotuje Trditvi 3. □

Zaključimo zdaj dokaz izreka. Ker G ni poln graf, obstajata vsaj dva soseda vozlišča v_0, ki nista sosednja v G (niti v G'). Označimo ju z v_1 in v_2. Na poti C_{12} obstaja vozlišče u, ki je sosed vozlišča v_1, in $u \neq v_2$. Ker je $\Delta \geq 3$ v grafu G' obstaja C_{13}. Zamenjajmo barvi (1 in 3) na poti C_{13}. V tem novem barvanju je v_1 pobarvano z barvo 3 in v_3 pobarvano z barvo 1. V tem novem barvanju novi komponenti C'_{12} in C'_{23} obe vsebujeta vozlišče $u \neq v_2$, kar je v nasprotju s Trditvijo 4. S tem je izrek dokazan.

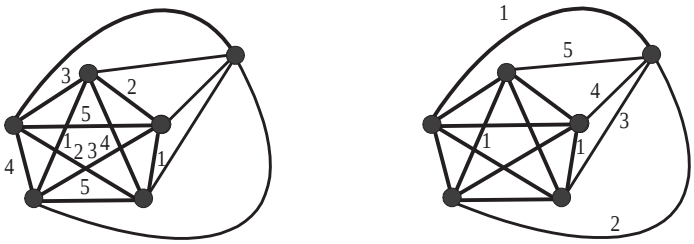
4.11. **Dokaz.** Ker je stopnja vsake točke enaka $n-1$, je po Vizingovem izreku $\chi'(G) = n-1$ ali $\chi'(G) = n$. V primeru polnih grafov lahko kromatične indekse določimo brez uporabe Vizingovega izreka. Sklepamo lahko takole:

Če je n liho število, potem lahko isto barvo dodelimo največ $\frac{1}{2}(n-1)$ povezavam, ker bi sicer vsaj dve od njih imeli skupno krajišče. Ker ima K_n $\frac{1}{2}n(n-1)$ povezav, mora biti barv vsaj n. Zares lahko najdemo n-barvanje povezav grafa K_n, če postopamo takole. Narišemo graf K_n kot pravilen n-kotnik. Povezave na robu pobarvamo z n različnimi barvami. Ostale povezave (diagonale) pa pobarvamo z isto barvo, kot jo ima vzporedna povezava na robu n-kotnika. Dobimo dobro barvanje povezav z n barvami, torej je kromatični indeks enak n, $\chi'(G) = n$. Na sliki 4.10 je narisana konstrukcija za primer $n = 5$.

Če je n sodo število, lahko dokažemo, da velja $\chi'(G) = n-1$. Ker je stopnja vozlišč enaka $n-1$, mora biti $\chi'(G) \geq n-1$. Enakost dokažemo tako, da poiščemo dobro $(n-1)$-barvanje povezav. Za $n = 2$ je naloga trivialna. Za $n > 2$ vzemimo katerokoli vozlišče in ga odstranimo (skupaj s povezavami, ki ga imajo za krajišče). Tako dobimo poln graf K_{n-1} z lihim številom vozlišč, ki mu povezave lahko dobro pobarvamo z $n-1$ barvami s prejšnjo konstrukcijo. V vsakem vozlišču manjka natanko ena barva in te barve so vse različne. Povezave K_n, ki smo jih prej odstranili, lahko torej pobarvamo s temi manjkajočimi barvami. Slika 4.11 kaže dobljeno barvanje v primeru $n = 6$.

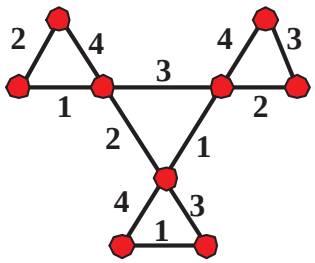


Slika 4.10: Barvanje povezav K_5.



Slika 4.11: Barvanje povezav K_6.

4.12.



5

ALGORITMI

Natančna definicija algoritma zahteva sorazmerno zahteven vpogled v matematično logiko in teoretično računalništvo[1]. K sreči je pojem algoritma tako naraven, da bo za naš namen zadoščala intuitivna definicija algoritma, ki jo bomo dodatno pojasnili z zgledi. Prvotno so besedo algoritem[2] uporabljali za elementarne računske operacije, kot je na primer seštevanje ali množenje večmestnih števil v desetiškem zapisu. Za izvajanje takega postopka je potrebno znanje seštevanja in množenja enomestnih števil in njihova uporaba v predpisanem vrstnem redu. Za pravilno izvajanje algoritma ni potrebno razumevanje ali dokaz pravilnosti postopka.

Algoritem je zaporednje ukazov, ki so enolično razumljivi izvajalcu algoritma. Ponavadi zahtevamo še, da je algoritem končen postopek, torej da se izvajanje vedno zaključi v končno mnogo korakih.

Primer: Poglejmo si algoritem za seštevanje desetiških števil, ki smo se ga naučili v prvih razredih osnovne šole. Zapišimo števili a in b v desetiškem zapisu. Seštevamo najprej enice, potem desetice, stotice, ... in seveda deseticam prištejemo ena, če je bila vsota enic večja od devet, stoticam prištejemo ena, če je bila vsota desetic večja od devet, in tako dalje. Pod črto zapisujemo števke vsote s. Na primer, če seštevamo 78162 + 25452, zapišemo

	7	8	1	6	2	število a
+	2	5	4	5	2	število b
	1	1	0	1	0	
1	0	3	6	1	4	vsota s

Naj bo n število decimalnih mest večjega od sumandov. Označimo cifre (števke) števila a z a_{n-1} $a_{n-2} \ldots a_2\ a_1\ a_0$, cifre b z $b_{n-1}\ b_{n-2} \ldots b_2\ b_1\ b_0$ in vsote s z $s_n\ s_{n-1}\ s_{n-2} \ldots s_2\ s_1\ s_0$. S c_i označimo bit[3], ki pove, ali je treba prišteti ena. V splošnem je račun

[1] Klasična definicija algoritma uporabi formalno definiran Turingov stroj (imenovan po Alanu Turingu 1912-1954) [T.Pisanski, Izračunljivost in rešljivost, Obz. mat. fiz., 25 (1978), let. 41-54, ali DMFA 1983]. Po Churchevi tezi so formalni modeli, ki vključujejo Turingov stroj in teorijo rekurzivnih funkcij, enakovredni intuitivnemu pojmu izračunljivosti. Vemo tudi, da je čas izvajanja programov v večini standardnih programskih jezikov polinomska funkcija časa, ki bi ga enakovreden program zahteval na Turingovem stroju. Zato lahko pri obravnavi zahtevnosti algoritmov uporabimo model(e), ki so bližji algoritmičnim programskim jezikom, kot so pascal, FORTRAN, C, ... Dokaze ekvivalentnosti računskih modelov najdemo na primer v [A.V. Aho, J.E. Hopcroft, J.D.Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974].

[2] Beseda izhaja iz latinske verzije imena učenjaka z imenom Muhamed Abu Džafar ibn Musa Al-Hwarázmi iz 9.stoletja. Po indijskih virih je priredil delo *Hisab algabr valmukabalah*, od koder izvira algebra.

[3] **Bit** je dvojiška števka. V teoriji informacije je bit osnovna enota. V diskretni matematiki in računalništvu so dvojiški logaritmi najpogostejši, zato običajno velja dogovor $\log n = \log_2 n$. (In ne $\log n = \log_{10} n$.)

	a_{n-1}	a_{n-2}	$\ldots$	a_2	a_1	a_0	število a
$+$	b_{n-1}	b_{n-2}	$\ldots$	b_2	b_1	b_0	število b
	c_n	c_{n-1}			c_2	c_1	
s_n	s_{n-1}	s_{n-2}	$\ldots$	s_2	s_1	s_0	vsota s

Postopek lahko opišemo takole: Najprej seštejemo $a_0 + b_0$, in priredimo s_0 število enic vsote, c_0 pa število desetic. (Torej, $a_0 + b_0 = 10c_1 + s_0$.) V naslednjem koraku seštejemo $a_1 + b_1 + c_1$, in priredimo s_1 število enic vsote, c_2 pa število desetic. (Torej, $a_1 + b_1 + c_1 = 10c_2 + s_1$.) V splošnem seštejemo $a_i + b_i + c_i$, in priredimo s_i število enic vsote, c_{i+1} pa število desetic. (Torej, $a_i + b_i + c_i = 10c_{i+1} + s_i$.) Nazadnje $s_n = c_n$, saj je $a_n = 0$ in $b_n = 0$. (Vidimo, da je s_n je lahko samo 0 ali 1.)

5.1. Algoritmi in programski jeziki

Algoritmi morajo biti zapisani nedvoumno. Tako v prej obravnavanem primeru seštevanja ali množenja naravnih števil privzamemo, da učenec v osnovni šoli obvlada seštevanje do dvajset in zna poštevanko, preden ga učimo seštevanja in množenja velikih števil. Če bo algoritem izvajal računalnik, ga zapišemo v enem od programskih jezikov in seveda privzamemo nabor ukazov programskega jezika. Ker je programskih jezikov veliko, vsak programski jezik pa ima več ali manj posebnosti, ki ponavadi niso bistvene za zapis algoritma, bomo v nadaljevanju zapisovali algoritme v jeziku, ki ga bomo imenovali **psevdopascal**[4]. V tem jeziku uporabljamo nekaj standardnih ukazov, ki jih v nekoliko drugačnem zapisu poznajo vsi postopkovno usmerjeni programski jeziki (kot so pascal, FORTRAN, C, ...), vmes pa zaradi enostavnosti zapišemo tudi kak ukaz kar v prostem jeziku. Nekaj algoritmov smo zapisali že v prejšnjih poglavjih. Odslej bodo naši zapisi algoritmov nekoliko bolj podrobni, ker nas bo zanimala tudi analiza algoritmov, predvsem njihove časovne zahtevnosti.

Najprej si poglejmo osnovne ukaze pseudopascala.

prireditveni stavek.

$x :=$ Navodilo za račun;

Zapis preberemo „vrednost spremenljivke x postane ...“. Na levi strani prireditve je vedno spremenljivka, na desni pa je lahko še kaj drugega, na primer enostaven aritmetični izraz ali pa kaj bolj zapletenega. Tako naslednji ukazi pomenijo:

$x := y$; vrednost spremenljivke x postane enaka vrednosti spremenljivke y,

$x := x + 1$; vrednost spremenljivke x se poveča za 1,

$x := z * y$; vrednost spremenljivke x postane enaka vrednosti produkta vrednosti spremenljivke z z vrednostjo spremenljivke y, ali krajše, x dobi vrednost produkta z z y,

$x :=$ z div 2; x je rezultat celoštevilskega deljenja z z 2,

[4]Jezik poimenujemo tako zato, ker je sintaksa še najbolj podobna programskemu jeziku pascal. Nabor ukazov je zaradi enostavnosti in univerzalnosti skromen, a vendar dovolj bogat, da lahko zapišemo katerikoli algoritem.

$x := z \bmod 2$; x postane ostanek celoštevilskega deljenja z z 2,

$x :=\gcd(y, z)$; x je največji skupni delitelj števil y in z,

Za izvedbo prireditvenega stavka je potreben en „korak“, če gre za preprosto prireditev vrednosti. Kadar je na desni strani zapleten izraz, je mogoče, da bo za izvedbo prireditvenega stavka potrebno veliko časa.

Za primer uporabe prireditvenega stavka si poglejmo zaporedje stavkov, s katerimi dosežemo zamenjavo vrednosti dveh spremenljivk, imenujmo ju a in b. Potrebujemo pomožno spremenljivko, recimo x.

$x := b$; $b := a$; $a := x$;

Delovanje zapisanih stavkov si lahko ponazorimo s tabelo vrednosti. Recimo, da ima pred izvajanjem a vrednost 100, b pa vrednost 33, vrednost x pa je nedefinirana. Po vsakem prireditvenem stavku zapišimo vrednosti v novo vrstico.

a	b	x komentar	
100	33	?	začetek
100	33	33	po $x := b$;
100	100	33	po $b := a$;
33	100	33	po $a := x$;

Pred opisom naslednjih ukazov jezika definirajmo **blok**:

begin $S_1; S_2; \ldots, S_n$ **end**

Tu so $S_1, S_2, \ldots, S_n$ poljubni ukazi našega jezika, zapis pa razumemo tako, da je treba najprej izvesti ukaz S_1, potem ukaz S_2, in tako dalje, dokler ne pride na vrsto ukaz S_n.

Pogojni stavek (stavek IF) ima obliko

if pogoj **then** C_1 **else** C_2;

Pogoj je logični izraz[5]. Če je pogoj izpolnjen, potem se izvede stavek C_1, sicer se izvede C_2. C_1 in C_2 sta lahko katerakoli stavka našega jezika.

Primer. Stavek

if $x < 0$ **then** $a := -x$ **else** $a := x$;

spremenljivki a priredi absolutno vrednost spremenljivke x, $a := |x|$.

Stavek FOR. (DO zanka v FORTRANU.) Stavek uporabljamo takrat, ko vnaprej vemo, kolikokrat bo treba ponoviti enak ali „skoraj enak“ del programa.

for $i := m$ **to** M **do** C;

[5]**Logični** ali **Boolov** izraz je izjava, ki ima lahko vrednost *resnično* ali *neresnično* (*true* ali *false*, 1 ali 0).

i, m, M so naravna števila, i je spremenljivka, ki jo imenujemo števec zanke,[6] m in M pa sta meji. Smiselno je zahtevati $m \leq M$. Stavek izvedemo tako, da $(M-m+1)$-krat izvedemo stavek C. Pri tem v C lahko uporabimo vrednost spremenljivke i.

Primer. S spodnjima ukazoma

$p := 1;$
for $i := 1$ **to** n **do** $p := p * a;$

izračunamo n-to potenco števila a, $p = a^n$.

S tabelo ponazorimo delovanje programa pri začetnih vrednostih $a = 3$ in $n = 4$.

a	n	i	p	komentar
3	4	?	1	po stavku $p := 1$
3	4	1	3	po prvi izvedbi for zanke
3	4	2	9	po drugi izvedbi for zanke
3	4	3	27	po tretji izvedbi for zanke
3	4	4	81	po četrti izvedbi for zanke

V spremenljivki p je na koncu res vrednost 81, torej $p = a^4 = 3^4 = 81$. Vidimo, da vrednosti a in n ne bi bilo treba tabelirati, saj se med izvajanjem ne spreminjata. Ko postanejo zapisi algoritmov bolj podrobni in zapleteni, se naravno pojavi potreba po dokazovanju pravilnosti programov, torej dokazovanju, da je rezultat zapisanega postopka res tisto, kar zahtevamo. Znane so formalne metode dokazovanja pravilnosti programov, vendar se tu z njimi ne bomo podrobneje ukvarjali.[7]

WHILE stavek. Stavek uporabljamo takrat, ko je treba ponavljati del programa, vendar vnaprej ni znano, koliko ponovitev bo potrebnih.

while pogoj **do** C;

Zanko izvajamo takole: če pogoj ni izpolnjen, ne naredimo ničesar; če je pogoj izpolnjen, potem izvedemo stavek C in ponovno preverimo veljavnost pogoja. To ponavljamo, dokler se ne zgodi, da pogoj ni več veljaven. Število ponovitev zanke torej ni znano vnaprej. V posebnih primerih lahko število ponovitev izračunamo ali ocenimo iz vsebine naloge.

Primer. Oglejmo si učinek delovanja naslednjega zaporedja ukazov:

$x := n;\ y := 0;\ z := 0;$
while $y < x$ **do**
  begin $y := y + 1;\ z := z + y$ **end**;

Hitro vidimo, da algoritem sešteje vsoto $1+2+3+\ldots+n$, vsota je na koncu shranjena v vrednosti spremenljivke z. Zapišimo v tabelo, kako se spreminjajo vrednosti spremenljivk, če je v začetku $n = 5$.

[6] Stavkom (ukazom) FOR, WHILE in REPEAT rečemo **zanke**.
[7] Glej na primer [S.Alagić, Principi programiranja, Svjetlost, Sarajevo 1983].

x	y	z
5	0	0
5	1	1
5	2	3
5	3	6
5	4	10
5	5	15

REPEAT stavek. Stavek podobno kot while stavek uporabljamo takrat, ko je treba ponavljati del programa, vendar vnaprej ni znano, koliko ponovitev bo potrebnih. Za razliko od while stavka tu pogoj preverjamo na koncu zanke, in zato se zanka v vsakem primeru izvede vsaj enkrat.

repeat C **until** pogoj;

KOMENTARJI. Delom programov bomo včasih dodali komentarje, besede ali stavke v zavitih oklepajih, ki po definiciji nimajo vpliva na izvajanje, lahko pa pomagajo pri razumevanju zapisa. Čeprav komentarji ne vplivajo na izvajanje postopka, so zelo pomembni pri razumevanju zapisa algoritma.

Učinkovitost algoritmov

Učinkovitost algoritmov običajno merimo tako, da ocenimo njihovo časovno (včasih tudi prostorsko) zahtevnost. Časovna zahtevnost algoritma A je število $f(n)$ operacij, ki jih zahteva izvajanje algoritma pri podatkih velikosti n v najslabšem primeru. Ponavadi ne štejemo vseh operacij, ampak samo najpomembnejše. To so tiste, za katere vemo, da je od njih najbolj odvisen čas izvajanja. Pri oceni dolžine podatkov moramo izbrati primeren način zapisa, tako na primer naravno število zapišemo z $\log_2 n$ biti in ne z n biti. Splošen graf običajno podamo z matriko z n^2 elementi, lahko pa tudi s seznami sosedov, za kar potrebujemo približno $n + 2m$ prostora.[8]

Na računalniku, ki naredi milijon operacij v sekundi, primerjajmo, koliko časa bi porabili za algoritme, ki imajo časovno zahtevnost $f(n)$ za nekaj tipičnih funkcij.

$f(n)$	$n = 20$	$n = 40$	$n = 60$	$n = 100$
n	0.00002 s	0.00004 s	0.00006 s	0.0001 s
n^2	0.0004 s	0.0016 s	0.0036 s	0.01 s
n^3	0.008 s	0.064 s	0.216 s	1 s
1.1^n	0.000007 s	0.000045 s	0.0003 s	0.014 s
2^n	1.0 s	12.7 dni	366000 let	4×10^{16} let

Če dobimo računalnik, ki je 1000 krat hitrejši, torej če naredi milijardo operacij v sekundi, dobimo:

[8] Vendar moramo v primeru, ko so vsi grafi, ki so lahko podatek za nalogo, določeno dodatno strukturo, to upoštevati in jih zapisati v najbolj ekonomičnem načinu. Če na primer vemo, da so vsi naši grafi cikli, za cikel C_n, porabimo samo $\log n$ bitov za zapis števila n in ne n^2 elementov za zapis matrike ali $n + m = 2n$ elementov za zapis seznama sosedov.

$f(n)$	$n = 20$	$n = 40$	$n = 60$	$n = 100$
n	0.00000002 s	0.00000004 s	0.00000006 s	0.0000001 s
n^2	0.0000004 s	0.0000016 s	0.0000036 s	0.00001 s
n^3	0.000008 s	0.000064 s	0.000216 s	0.001 s
1.1^n	0.000000007 s	0.000000045 s	0.0000003 s	0.000014 s
2^n	0.001 s	18.3 min	37 let	4×10^{13} let

Vidimo, da je hitrost rasti eksponentnih funkcij mnogo večja kot je hitrost rasti polinomskih funkcij. Ker se hitrost računskih strojev hitro povečuje, je mogoče reševati vedno večje naloge. Vendar, kot vidimo iz tabel, pri polinomskih funkcijah pohitritev računanja pomeni, da bomo zares lahko reševali večje naloge, pri eksponentnih funkcijah pa se izkaže, da tudi na 1000-krat hitrejšem stroju lahko izračunamo v primerno kratkem času samo malo večje naloge.

Zato se je izoblikovala delitev na algoritme z največ polinomsko časovno zahtevnostjo in algoritme z nepolinomsko[9] časovno zahtevnostjo. Primer algoritma, ki potrebuje 2^n operacij, je algoritem, ki pregleda vse podmnožice množice z n elementi. Tak algoritem očitno ni uporaben za večje množice. Mnogo bolj praktično uporabni algoritmi so algoritmi, pri katerih naredimo na primer n, n^2 ali n^3 operacij. Algoritem, ki zahteva v najslabšem primeru n^c operacij za neko konstanto c imenujemo **algoritem s polinomsko časovno zahtevnostjo**, ali **polinomski** algoritem. Algoritem, ki zahteva v najslabšem primeru c^n operacij za neko konstanto c pa imenujemo **algoritem z eksponentno časovno zahtevnostjo**, ali **eksponentni** algoritem. Polinomskim algoritmom rečemo tudi **učinkoviti** algoritmi, nepolinomskim algoritmom pa **neučinkoviti** algoritmi.

Veliki O

Naj bo f funkcija iz $\mathbb{N}$ v $\mathbb{N}$. Rečemo, da je f **(kvečjemu) reda** $O(g)$, če obstaja pozitivna konstanta k, tako da je $f(n) \leq kg(n)$ za vse $n \in \mathbb{N}$. Včasih zapišemo $f = O(g)$ ali, točneje, $f \in O(g)$.

Omenimo še dve oznaki: če je $f \in O(g)$, potem zapišemo tudi $g \in \Omega(f)$ in rečemo, da je g **(vsaj) reda** $O(f)$. Če je $f \in O(g)$ in $f \in \Omega(g)$ in rečemo, da sta f in g **istega reda** in zapišemo $f \in \Theta(g)$ ali $f = \Theta(g)$.

Zapišimo brez dokazov nekaj lastnosti O:

1. Konstantni faktor lahko ignoriramo: Za vse $k > 0$, je kf reda $O(f)$. (Tako sta izraza an^2 in bn^2 oba reda $O(n^2)$.)
2. Višje potence rastejo hitreje kot nižje: n^r je $\mathrm{O}(n^s)$ če je $0 \leq r \leq s$.
3. Hitrost rasti vsote je hitrost najhitreje rastočega sumanda: Če je f reda $O(g)$, potem je $f + g$ reda $O(g)$. (Primer: Izraz $6n^3 + 9n^2$ je $O(n^3)$.)
4. Red polinoma je enak redu vodilnega člena: Polinom f stopnje d je $O(n^d)$.

[9] Rečemo tudi superpolinomsko, kar ponavadi pomeni z eksponentno.

5. Če f raste hitreje kot g in ta raste hitreje kot h, potem f raste hitreje kot h.

6. Red rasti produkta je produkt: Če je f kvečjemu reda $O(g)$ in h kvečjemu reda $O(r)$, potem je fh kvečjemu reda $O(gr)$. (Primer: Če je f kvečjemu reda $O(n^2)$ in g kvečjemu reda $O(\log n)$ je fg kvečjemu reda $O(n^2 \log n)$.)

7. Eksponentne funkcije rastejo hitreje kot potence: n^k je kvečjemu reda $O(b^n)$, za vse $b > 1$; $k \geq 0$, (Na primer, n^4 je $O(2^n)$ pa tudi $O(e^n)$.)

8. Logaritmi rastejo počasneje kot potence: $\log_b n$ je kvečjemu reda $O(n^k)$ za vse $b > 1$; $k > 0$. (Na primer, $\log_2 n$ je kvečjemu reda $O(n^{1/2})$.)

9. Vsi logaritmi rastejo enako hitro: $\log_b n$ je $O(\log_d n)$ in $\log_d n$ je $O(\log_b n)$ za poljubni števili $b, d > 1$.

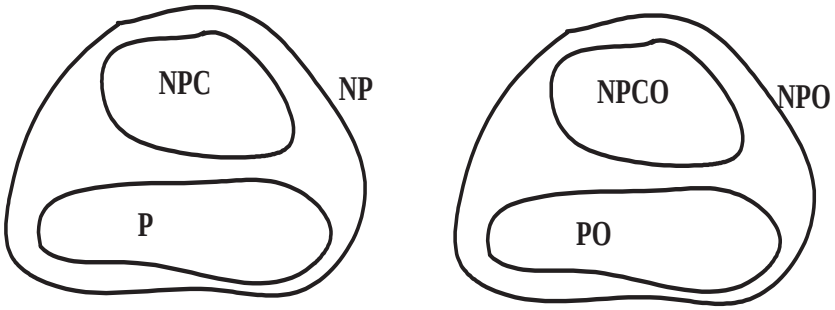
Algoritem je *polinomski*, če je njegova časovna zahtevnost $f(n)$ kvečjemu reda $O(n^k)$ za neki $k \in \mathbb{N}$, sicer je algoritem *nepolinomski* (superpolinomski, neučinkovit).

Primer. Ker je za $n \in \mathbb{N}$

$$n^2 + 1000n + 5 \leq (1 + 1000 + 5)n^2$$

lahko za funkcijo $f(n) = n^2 + 1000n + 5$ rečemo, da je reda $O(n^2)$. Podobno je $5n^4 + 20n^3 + n^2$ reda $O(n^4)$, $2^n + 3n^5 + n^{99}$ pa je reda $O(2^n)$.

5.2. Razredi problemov



Slika 5.1: Razredi odločitvenih in optimizacijskih problemov.

V teoriji algoritmov definiramo razrede problemov glede na to, kako hitro jih je mogoče rešiti. Probleme ponavadi formalno zapišemo v obliki:

PROBLEM:

PODATKI:

VPRAŠANJE, CILJ:

Nekaj primerov:

PROBLEM: k-barvanje
PODATKI: enostaven graf G
VPRAŠANJE: Ali obstaja dobro k-barvanje grafa G?

PROBLEM: barvnost grafa
PODATKI: enostaven graf G
VPRAŠANJE: Kolikšno je kromatično število grafa G?

PROBLEM: minimalno vpeto drevo
PODATKI: enostaven graf G z utežmi na povezavah
CILJ: Poišči vpeto drevo z minimalno utežjo!

PROBLEM: trgovski potnik
PODATKI: enostaven graf G z utežmi na povezavah
CILJ: Poišči Hamiltonov cikel z minimalno utežjo!

Iz formalnih razlogov ločimo odločitvene in optimizacijske probleme.

Odločitveni problem je določen z množico **nalog** (ali **primerkov**), torej z množico vseh možnih podatkov. Množica nalog razpade na dva razreda: tiste, za katere je odgovor na vprašanje pritrdilen, in tiste, za katere je odgovor NE.

Na primer, za problem k-barvanja so naloge ali primerki vsi enostavni grafi. Za vsak graf dobro k-barvanje obstaja ali pa ne.

Optimizacijski problem je določen z množico nalog in z **namensko** (**ciljno**, **stroškovno**) funkcijo. Lahko privzamemo, da je vedno treba poiskati minimum namenske funkcije.[10] Optimizacijske probleme bi lahko nadalje ločili še po tem, ali nas zanima samo optimalna vrednost namenske funkcije, ali pa želimo imeti tudi vsaj eno od **optimalnih** rešitev.[11]

Obravnavo zahtevnosti problemov običajno začnemo z odločitvenimi problemi. Odločitveni problem je v **razredu P**, če ostaja polinomski algoritem za njegovo reševanje. V **razredu NP** so odločitveni problemi, ki jih je mogoče rešiti v polinomskem času z *nedeterminističnim algoritmom*. Pojem nedeterminističnega algoritma je za našo obravnavo tu prezahteven[12], zato samo omenimo ekvivalentno definicijo: problem je v razredu NP, če obstaja polinomski algoritem, ki v primeru pozitivnega odgovora preveri pravilnost odgovora. Poglejmo, kaj to pomeni na primeru zgoraj:

[10] Če nas zanima maksimum, pač definiramo problem s funkcijo $-f$.

[11] Včasih je za zapis optimalne rešitve potrebno veliko časa (ali prostora) in ta na videz nedolžna zahteva lahko vpliva na zahtevnost problema.

[12] Glej na primer [B.Robič, Aproksimacijski algoritmi, Založba FRI, Ljubljana 2002], ali pa ustrezno poglavje v (skoraj) vsaki knjigi na temo diskretne optimizacije.

če je odgovor pritrdilen, potem obstaja dobro barvanje. Preveriti, ali je dano barvanje dobro, je seveda preprosto in mogoče v polinomskem času. Zato je odločitveni problem k-barvanja v razredu NP.

Očitno je razred P vsebovan v razredu NP. Verjetno velja domneva

$$P \neq NP \ ,$$

ki pa je odprta že od leta 1972.[13]

V razredu NP je pomembna podmnožica problemov, imenovanih **NP-polni problemi**. (Oznaka NPC zaradi angl. NP complete.) Za katerikoli NP-poln problem velja naslednje: če bi obstajal polinomski algoritem zanj, potem bi lahko v polinomskem času rešili vse probleme iz NP, torej bi bilo P = NP. In obratno, če bi dokazali, da za enega od problemov iz razreda NPC ni mogoče zapisati polinomskega algoritma, potem bi to veljalo za vse probleme iz NPC. V razredu NPC so mnogi problemi[14], zato je zelo verjetno, da velja P $\neq$ NP.

Med problemi, ki smo jih že obravnavali v tej knjigi sta polinomska problema odločitveni problem 2-barvanja in odločitveni problem: „Ali je graf Eulerjev?“, NP-polna pa sta problem k-barvanja za $k > 3$ in odločitveni problem: „Ali je graf Hamiltonov?“.

Podobno definiramo NP optimizacijske probleme (razred NPCO), v polinomskem času polinomskem rešljive optimizacijske probleme (razred PO) in NP-polne optimizacijske probleme (razred NPCO).[15] Optimizacijski problem je v razredu PO, če obstaja polinomski algoritem za njegovo reševanje. Izkaže se, da je to natanko tedaj, ko je odločitvena različica problema v razredu P.

Optimizacijskemu problemu lahko vedno priredimo odločitveno obliko takole: Podatkom dodamo še eno število m in namesto iskanja minimuma namenske funkcije postavimo vprašanje: „Ali je minimum namenske funkcije manjši od m?“.

Optimizacijski problem je **NP-težek**, če je njegova odločitvena različica NP-poln problem.

Primeri polinomsko rešljivih optimizacijskih problemov so: problem minimalnega vpetega drevesa, iskanje Eulerjevega obhoda, problem kitajskega poštarja.

NP-težki optimizacijski problemi pa so: problem trgovskega potnika, barvnost grafa.

5.3. Primerjava algoritmov

Za reševanje danega problema običajno izberemo algoritem, ki je najbolj učinkovit pri uporabi virov. Pogosto je najpomembnejši vir čas, zato si poglejmo nekaj primerjav algoritmov glede na časovno zahtevnost.

[13] [R.M.Karp: Reducibility among combinatorial problems, v Complexity of Computer Computations (ur. Miller, Thatcher), Plenum Press, New York, 85-103, (1972)] Domneva je najbolj znan odprt problem teoretičnega računalništva.

[14] Glej seznam v [M.Garey, D.Johnson, Computers and Intractability, Freeman, San Francisco 1979] ki se stalno povečuje.

[15] Za podrobnosti glej npr. [B.Robič, Aproksimacijski algoritmi, Založba FRI, Ljubljana 2002].

Najprej primerjajmo dva algoritma za računanje potence u^m. Potenco lahko izračunamo s preprosto zanko

```
power := 1;
for i = 1 to  m do  power := power * q;
```

ki ima časovno zahtevnost $O(m)$.

Naslednji algoritem z računanje u^m je nekoliko bolj zapleten

```
q := m; bot := 1; top := u;
while q > 0 do
  begin
    if q mod 2 = 1 then bot := top * bot;
    top := top * top;
    q := q div 2;
  end;
power := bot;
```

vendar ima bistveno boljšo časovno zahtevnost, $O(\log m)$. Glej nalogo 5.3.

Množenje matrik. Po definiciji lahko matriki velikosti $n \times n$ zmnožimo s preprosto zanko, ki zahteva $O(n^3)$ množenj. (Izračunati moramo n^2 elementov, za vsakega pa en skalarni produkt dveh vektorjev s po n komponentami, torej $n-1$ seštevanj in n množenj.

```
for i = 1 to  n do
  for j = 1 to  n do
    begin
      C[i,j] := A[i,1] * B[1,j];
      for k = 2 to  m do  C[i,j] := C[i,j] + A[i,k] * B[k,j];
    end;
```

Množenje matrik lahko realiziramo v času $O(n^{\log_2 n}) = O(n^{2.81})$.

Če pomnožimo dve matriki velikosti 2×2

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

po definiciji, potrebujemo osem množenj in štiri seštevanja. Za množenj matrik po definiciji torej potrebujemo n^3 množenj in $n^2(n-1)$ seštevanj.

Strassen[16] je leta 1969 presenetil z odkritjem, da se da matrike pomnožiti hitreje. Najprej je izračunal:

$$\begin{aligned} x_1 &= (a+d)(e+h) \\ x_2 &= (b-d)(g+h) \\ x_3 &= (a-c)(e+f) \\ x_4 &= (a+b)h \\ x_5 &= (c+d)e \\ x_6 &= a(f-h) \\ x_7 &= d(-e+g) \end{aligned}$$

potem pa uporabil enakosti

$$\begin{aligned} ae+bg &= x_1+x_2-x_4+x_7 \\ af+bh &= x_4+x_6 \\ ce+dg &= x_5+x_7 \\ cf+dh &= x_1-x_3-x_5+x_6 \end{aligned}$$

Za tak izračun potrebujemo 18 seštevanj (in odštevanj) in samo sedem množenj. (To je dobro, ker so seštevanja cenejša kot množenja.)

Če želimo dobiti algoritem za množenje matrik večje velikosti, matriki razdelimo na 4 bloke in izvedemo račun z matrikami polovične velikosti. Zaradi enostavnosti lahko privzamemo, da je n potenca števila 2. Število vseh množenj je potem enako

$$T(n) = 7 \cdot T\left(\frac{n}{2}\right) \qquad in \qquad T(1) = 1 \ , \tag{5.1}$$

od koder sledi

$$T(n) = O(n^{\log_2 7}) = O(n^{2.81}) \ .$$

(Upoštevali smo rezultat iz naloge 5.7. in $\log_2 7 < 2.81$.)

[16] V.Strassen, Gaussian elimination is not optimal, Numerische Mathematik 13 (1969) 354-356. Matrike se da množiti še hitreje, v času $O(n^{2.376})$, glej [D.Coppersmith, S.Winograd. Matrix multiplication via arithmetic progressions. J. Symbolic Computation, 9 (1990) 251–280].

5.4. Naloge z rešitvami

5.1. Katerega reda je rast funkcij (a) $f(n) = n + n^{100} + 2^n$, (b) $g(n) = 10000000n3^n$, (c) $h(n) = 9n + 8n^2 + 7\log^2 n + 6n^2 \log n$?

5.2. Oceni število operacij v naslednjem algoritmu:

```
a := 0; b := n;
while b > 0 do
  begin
    a := a + b * b; b := b - 1
  end;
s := a;
```

5.3. Oceni število operacij v naslednjem algoritmu in tabeliraj vrednosti spremenljivk pri vrednostih podatkov ($m = 10$, $u = 2$) in pri ($m = 6$, $u = 3$).

```
q := m; bot := 1; top := u;
while q > 0 do
  begin
    if q mod 2 = 1 then bot := top * bot;
    top := top * top;
    q := q div 2;
  end;
power := bot;
```

5.4.

```
a := 0; b := n; c := m;
while b > 0 do
  begin
    a := a + c; b := b - 1
  end;
q := a;
```

(a) Tabeliraj vrednosti spremenljivk a, b, in c pri vrednostih podatkov ($n = 13$, $m = 7$) in pri ($n = 9$, $m = 107$)!

(b) Za kateri „problem“ bi lahko uporabili ta algoritem (kaj bi bili podatki, kaj vprašanje)?

(c) Oceni časovno zahtevnost algoritma!

5.5. Naslednji algoritem ima dva podatka, naravni števili x in y.

$q := 0$; $r := x$;
while $r \geq y$ **do**
  begin
    $r := r - y$; $q := q + 1$
  end;

1. Tabeliraj vrednosti spremenljivk pri podatkih (a) $x = 15$, $y = 7$ in (b) $x = 122$, $y = 20$.
2. Dokaži, da zanka ohranja vrednost izraza $q * y + r$.
3. Kaj izračuna algoritem? (Za rezultat lahko štejemo vrednosti v spremenljivkah q in r.)

5.6. Analiziraj delovanje naslednjega algoritma.

$a := 0$; $b := n$; $c := m$;
while $b > 0$ **do**
  begin
    while $b \bmod 2 = 0$ **do**
      begin $b := b \operatorname{div} 2$; $c := 2 * c$ **end**;
    $a := a + c$; $b := b - 1$
  end;
$q := a$;

5.7. Dokaži, da je rešitev rekurzije

$$\begin{aligned} T(1) &= 0 \\ T(n) &= aT\left(\frac{n}{c}\right) + bn \end{aligned} \tag{5.2}$$

enaka

$$T(n) = \begin{cases} O(n), & \text{če je } a < c \\ O(n \log n), & \text{če je } a = c \\ O(n^{\log_c a}), & \text{če je } a > c\ . \end{cases} \tag{5.3}$$

5.8. Analiziraj delovanje naslednjega algoritma. (Algoritem je del delujočega programa, zato je zapisan v sintaksi jezika pascal in ima zato nekaj dodatkov, ki jih tu nismo definirali.)

```
{del programa bisekcija}
function f(x:real):real; begin f := 2 * x*sqr(x) - 3*sqr(x) + 7*x-10 end;
begin
  a := 1; b := 2; eps:= 0.0001;
  while (f(a) * f(b) < 0) and (b - a ≥ eps) do begin
    c := (a + b)/2;
    if (f(c) <> 0) then
      begin if (f(c) * f(b) < 0) then a := c else b := c end;
  end;
end.
```

Opomba. S stavkom **function** v pascalu definiramo funkcijo. Stavek function je smiselno uporabiti, kadar večkrat izračunamo vrednosti iste funkcije, saj tako povečamo razumljivost zapisa. (sqrt(x) je standardna funkcija v pascalu, ki izračuna vrednost x^2.)

Rešitve

5.1. (a) $O(2^n)$,

(b) $O(n3^n)$,

(c) $O(n^2 \log n)$.

5.2. Vseh operacij je reda $O(n)$. Bolj natančno: $2n$ seštevanj (in odštevanj), n množenj, $2n+3$ prirejanj in n preverjanj pogoja.

5.3. Število ponovitev zanke je sorazmerno dvojiškemu logaritmu q, v zanki pa se izvede eno testiranje pogoja, eno deljenje in eno (ali dve) množenji. Število operacij je torej $O(\log m)$.

q	*top*	*bot*	
10	1	2	začetek
5	1	4	po prvi izvedbi zanke
2	4	16	po drugi izvedbi zanke
1	4	256	po tretji izvedbi zanke
0	1024	256^2	po četrti izvedbi zanke

q	*top*	*bot*	
6	1	3	začetek
3	1	9	po prvi izvedbi zanke
1	9	81	po drugi izvedbi zanke
0	729	81^2	po tretji izvedbi zanke

Opomba. Algoritem izračuna potenco u^m.

5.4. (a)

a	b	c	
0	13	7	začetek
7	12	7	po prvi izvedbi zanke
14	11	7	po drugi izvedbi zanke
21	10	7	
28	9	7	
35	8	7	
42	7	7	
49	6	7	
56	5	7	
63	4	7	
70	3	7	
77	2	7	
84	1	7	
91	0	7	

a	b	c	
0	9	107	začetek
107	8	107	po prvi izvedbi zanke
214	7	107	po drugi izvedbi zanke
321	6	107	
428	5	107	
535	5	107	
642	3	107	
749	2	107	
856	1	107	
963	0	107	

(b) Algoritem izračuna produkt $n \cdot m$, ki je ob koncu izvajanja zanke shranjen v spremenljivki q.

(c) Časovna zahtevnost je $O(n)$.

5.5. (a)

q	r	y	
0	15	7	začetek
1	8	7	po prvi izvedbi zanke
2	1	7	po drugi izvedbi zanke

q	r	y	
0	122	20	začetek
1	102	20	po prvi izvedbi zanke
2	82	20	po drugi izvedbi zanke
3	62	20	
4	42	20	
5	22	20	
6	2	20	

(b) Dokažimo, da se vrednost izraza ohrani po izvedbi zanke.

Definirajmo zaporedji vrednosti spremenljivk q_n in r_n. Prireditvena stavka pred zanko povesta $q_0 = 0$, $r_0 = x$, iz prireditvenih stavkov v zanki pa dobimo $q_{i+1} = q_i + 1$ in $r_{i+1} = r_i - y$, Torej

$$q_{i+1} * y + r_{i+1} = (q_i + 1) * y + r_i - y = q_i * y + 1 * y + r_i - y = q_i * y + r_i$$

kar smo želeli dokazati.

(c) Najprej vidimo, da pri $q = 0$ in $r = x$ velja $q * y + r = x$. Ker zanka ohranja vrednost izraza $q * y + r$, ob koncu izvedbe zanke velja

$$x = q * y + r \ , \qquad 0 \leq r \leq y \ ,$$

in vidimo, da algoritem izvede celoštevilsko deljenje x z y. ($q = x$ div y, $r = x$ mod y).

5.6. Algoritem za računanje $n * m$ s časovno zahtevnostjo $O(\log n)$.

5.7. Zaradi enostavnosti privzemimo, da je $n = c^k$ za neko naravno število k.

$$\begin{aligned} T(n) &= aT\left(\frac{n}{c}\right) + bn \\ &= a\left(aT\left(\frac{n}{c^2}\right) + b\frac{n}{c}\right) + bn \\ &= a^2T\left(\frac{n}{c^2}\right) + b\left(\frac{a}{c} + n\right) \\ &= a^kT\left(\frac{n}{c^k}\right) + b\sum_{i=0}^{k-1}\left(\frac{a}{c}\right)^i \\ &= bn\sum_{i=0}^{\log_c n-1}\left(\frac{a}{c}\right)^i \end{aligned} \tag{5.4}$$

Uporabili bomo formulo za delno vsoto geometrijske vrste

$$\sum_{i=0}^{k} a\,(q)^{i-1} = a\frac{1-q^k}{1-q}\ .$$

če je $|q| < 1$, potem vrsta konvergira in

$$\sum_{i=0}^{\infty} = a\frac{1}{1-q}\ .$$

Ločimo tri primere. Če je $a < c$, potem upoštevamo, da je vsota končno mnogo členov konvergentne geometrijske vrste manjša od njene vsote (saj je $0 < \frac{a}{c} < 1$), in dobimo

$$\sum_{i=0}^{\log_c n-1}\left(\frac{a}{c}\right)^i \le \sum_{i=0}^{\infty}\left(\frac{a}{c}\right)^i = \frac{1}{1-\frac{a}{c}}\ ,$$

torej $T(n) = O(n)$.

V primeru $a = c$ je

$$\sum_{i=0}^{\log_c n-1}\left(\frac{a}{c}\right)^i = \sum_{i=0}^{\infty} 1 = \log_c n = O(\log n)\ ,$$

torej $T(n) = O(n \log n)$.

Če je $a > c$, potem seštejemo končno mnogo členov tokrat nekonvergentne geometrijske vrste in uporabimo znane zveze za računanje z logaritmi:

$$\begin{aligned} b \cdot n\sum_{i=0}^{\log_c n-1}\left(\frac{a}{c}\right)^i &= b \cdot n\left(\frac{1-(\frac{a}{c})^{\log_c n-1}}{1-\frac{a}{c}}\right) \\ &= \frac{b}{\frac{a}{c}-1} \cdot n \cdot \left(\left(\frac{a}{c}\right)^{\log_c n-1} - 1\right) \\ &= O\left(n\frac{a^{\log_c n}}{c^{\log_c n}}\right) \\ &= O(a^{\log_c n}) = O(n^{\log_c a})\ , \end{aligned}$$

torej $T(n) = O(n^{\log_c a})$.

5.8. Pogoj $f(a) * f(b) < 0$ je izpolnjen, če sta vrednosti $f(a)$ in $f(b)$ različno predznačeni in obe različni od nič. Če ta pogoj ni izpolnjen, potem velja $f(a) = 0$ ali $f(b) = 0$ ali pa sta $f(a)$ in $f(b)$ istega predznaka (v tem primeru na intervalu ničle ne bi bilo - da se dokazati, da ta možnost ne more nastopiti, če je f zvezna funkcija[17]). Pogoj $(b - a \geq \text{eps})$ je izpolnjen, če interval še ni dovolj majhen, da bi bili zadovoljni s približno rešitvijo. Če je $(b - a < \text{eps})$, potem za rešitev lahko vzamemo a, b ali $\frac{a+b}{2}$, in vemo, da se ta od prave ničle razlikuje za manj kot $\frac{1}{2}$eps.

Zapisani algoritem z *metodo bisekcije* izračuna ničlo funkcije $f(x) = 2x^3 - 3x^2 + 7x - 10$ na intervalu [1,2]. V zanki najprej izračunamo vrednost funkcije v sredini intervala. Če je vrednost enaka 0, smo ničlo našli. Sicer je vrednost ali pozitivna ali negativna. Potem izberemo tisto polovico intervala, na katerega krajiščih ima f različno predznačeni vrednosti in postopek ponovimo. Postopek ponavljamo, dokler ne najdemo ničle, ali pa dokler intervala, na katerem je ničla, ne zmanjšamo pod predpisano mero natančnosti.

V našem primeru izračunamo najprej $f(1.5) = 2(1.5)^3 - 3(1.5)^2 + 7(1.5) - 10 = 0.5$. Ker je $f(1.5) > 0$, mora biti ničla na intervalu $(1, 1.5)$. Nadaljujemo, $f(1.25) = \ldots = -2.03125 < 0$, torej mora biti vsaj ena ničla med 1.25 in 1.5. In tako dalje.

Izračunane vrednosti f so

$$
\begin{array}{ll}
f(1.000000) = -4.000000 & f(2.000000) = 8.000000 \\
f(1.000000) = -4.000000 & f(1.500000) = 0.500000 \\
f(1.250000) = -2.031250 & f(1.500000) = 0.500000 \\
f(1.375000) = -0.847656 & f(1.500000) = 0.500000 \\
f(1.437500) = -0.195801 & f(1.500000) = 0.500000 \\
f(1.437500) = -0.195801 & f(1.468750) = 0.146423 \\
f(1.453125) = -0.026085 & f(1.468750) = 0.146423 \\
f(1.453125) = -0.026085 & f(1.460938) = 0.059817 \\
f(1.453125) = -0.026085 & f(1.457031) = 0.016779 \\
f(1.455078) = -0.004675 & f(1.457031) = 0.016779 \\
f(1.455078) = -0.004675 & f(1.456055) = 0.006046 \\
f(1.455078) = -0.004675 & f(1.455566) = 0.000684 \\
f(1.455322) = -0.001996 & f(1.455566) = 0.000684 \\
f(1.455444) = -0.000656 & f(1.455566) = 0.000684 \\
f(1.455444) = -0.000656 & f(1.455505) = 0.000014
\end{array}
$$

Ničla je na intervalu $(1.455444, 1.455505)$. Odgovor: ničla je 1.455475 z napako manjšo od 0.000031.

[17] Glej na primer [J.Žerovnik, Matematika I, Fakulteta za strojništvo Maribor 1996.]

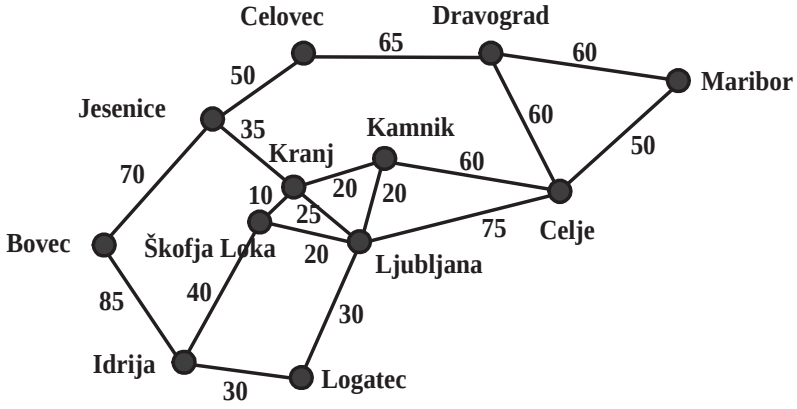
6

RAČUNANJE NAJKRAJŠIH POTI IN RAZDALJ

Poglejmo si naslednji nalogi:

1. Turist se želi peljati iz Maribora v Bovec po najkrajši poti. Na naslednji karti so označene približne razdalje med pari mest. Kakšno razdaljo mora prevoziti? Katero pot naj izbere?

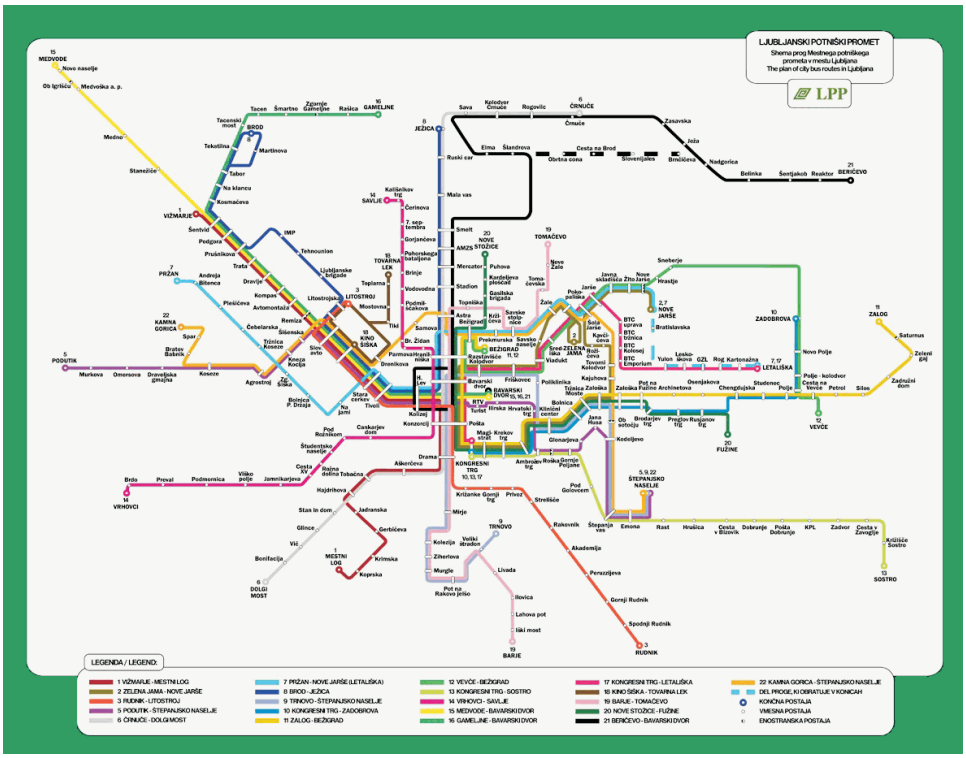
V tem posebnem primeru ni težko najti rešitve z nekaj spretnega ugibanja, toda tak pristop je mnogo manj uspešen v primerih, ko postane mreža cest bolj in bolj zapletena.



2. Na naslednji sliki je shema prog mestnega prometa v Ljubljani. Če nas zanimajo najkrajše poti med deli mesta (med postajami mestnih avtobusov), lahko na osnovi sheme definiramo graf z utežmi na povezavah. Uteži lahko smiselno določimo na več načinov, odvisno od tega, v kakšnem smislu najkrajše poti nas zanimajo. Na primer, lahko se vprašamo, kolikokrat je treba zamenjati avtobus, da pridemo iz Trnovega do tovarne Lek. Drugo najkrajšo pot (verjetno) dobimo, če nas zanima minimalno število vmesnih postajališč na poti.

Spomnimo se, da je razdalja med vozliščema v grafu po definiciji dolžina najkrajšega sprehoda med njima. Ker je sprehod najkrajše dolžine vedno pot, ponavadi rečemo, da je razdalja v grafu dolžina najkrajše poti. Razdalje v grafu lahko izračunamo na več načinov. Tu si bomo najprej ogledali algoritem za računanje razdalj od izbranega vozlišča do vseh ostalih z algoritmom iskanja v širino (DFS, breadth first search). Časovna zahtevnost algoritma je $O(m)$, vse razdalje v grafu pa dobimo v času $O(mn)$, če algoritem izvedemo iz vsakega od vozlišč. Posplošitev algoritma, ki deluje na omrežjih (grafih s pozitivnimi utežmi na povezavah), imenujemo Dijkstrov algoritem. Dijkstrov algoritem ima časovno zahtevnost $O(n^2)$, vse razdalje na omrežju torej lahko izračunamo

v času $O(n^3)$.



Opomba. Včasih nas zanimajo *najdaljše* poti med vozlišči grafa. Na primer takrat, ko z grafom opišemo proizvodni proces in je za dokončanje izdelka potrebno opraviti vse aktivnosti v proizvodni shemi.[1]

6.1. Pregledovanje v širino

V tem razdelku si bomo najprej ogledali algoritem pregledovanje v širino (angl. Breadth first search algorithm, BFS), potem pa ga bomo uporabili za računanje razdalj.

Pregledovanje v širino smo srečali že v 3. poglavju, zdaj pa zapišimo algoritem v bolj formalni obliki:

$vrsta := (v)$;
while vrsta ni prazna **do begin**
  $x := prvi(vrsta)$;
  if x ima novega soseda y

[1] Glej strani 192-195 v [R.Wilson, M.Watkins, Uvod v teorijo grafov, DMFA, Ljubljana 1997].

```
        then dodaj y na konec vrste
        else odstrani x iz vrste
      endif
    endwhile
```

Algoritem lahko uporabimo za računanje razdalj od vozlišča v do vseh ostalih vozlišč s preprostim dodatkom. Na začetku postavimo $d(v) = 0$ in vsakič, ko srečamo novo vozlišče y (ki je sosed x-a) izračunamo $d(y) = d(x) + 1$, ko dodamo y na konec vrste.

V algoritmu smo uporabili podatkovno strukturo **vrsta** (angl. First In First Out queue, FIFO). Običajno vrsto implementiramo tako, da sprogramiramo operacije: *prvi*, *odstrani*, *dodaj* in *prazna*. Operacija *prvi* vrne prvi element vrste, *odstrani* prvi element vrste odstrani iz vrste, operacija *dodaj* doda nov element v vrsto. Operacija *prazna* nam pove, ali je vrsta prazna ali ne. Definicijo podatkovne strukture lahko zapišemo strogo formalno (z aksiomi), tu pa namesto podrobnosti opišemo v pascalskem programu (del programa SZEGED za računanje Szeged indeksa).[2]

Časovna zahtevnost algoritma je $O(m)$, vse razdalje v grafu pa dobimo v času $O(mn)$, če algoritem izvedemo iz vsakega od vozlišč.

Algoritem za računanje razdalj v grafu na osnovi iskanja v širino zapišimo še malo bolj formalno, kot delujoč program v pascalu. Računanje razdalj je del programa, ki izračuna Szegedovo število grafa (glej definicijo na strani 28).

```
      program SZEGED(input,output);
(*
  This program calculates the Szeged index  of a graph.
  Graph is given by adjacency lists of its vertices.
*)
  const size= 100; max_degree = 10;
  type vector = array[1..max_degree] of integer;
       vector_of_vectors = array[1..size] of vector;

  var Degree: vector; (*vertex degrees*)
      Adj_list: vector_of_vectors; (*adjacency lists of the graph*)
      Distance: array[1..size] of vector; (*distances*)
      n: integer; (*number of vertices*)
      Sz: longint;
      u, v, i, j, diff : integer;
      n1, n2 : integer;
      Visited: array[1..size] of boolean;

 (* Data structure FIFO queue *)
     queue: vector;
     First,Last: integer;

  procedure reset_queue;
     begin First:=1; Last:=0; end;
```

[2]Splošneje lahko vrste (angl. queue) imenujemo vse podatkovne strukture, v katere na eni strani dodajamo elemente, na drugi pa jih v skladu z nekim kriterijem odvzemamo. Pogost primer je tudi **sklad**, (angl. LIFO, Last In First Out queue) in vrste s prednostjo. Glej tudi [J.Kozak, Podatkovne strukture in algoritmi, str. 32].

```
  procedure append_queue(x: integer);
     begin queue[Last +1] := x; Last := Last + 1;  end;
  procedure get_queue(var x: integer);
     begin x:= queue[First]; First := First + 1;  end;
  function empty_queue:boolean;
     begin if First>Last then empty_queue:= true else empty_queue:= false end;

 (* Breadth first search *)
  procedure BFS(Start_vertex:integer);
  var i:integer; neighbor: integer;
  begin
     for i:=1 to n do Visited[i]:= false;
     reset_queue;
     Distance[Start_vertex][Start_vertex]:= 0; Visited[Start_vertex] := true;
     for i:=1 to Degree[Start_vertex] do begin
        neighbor:= Adj_list[Start_vertex][i];
        append_queue(neighbor); Visited[neighbor] := true;
        Distance[Start_vertex][neighbor] := 1;
     end;
     while (not(empty_queue)) do begin
        get_queue(u); Visited[u]:= true;
        for i:=1 to Degree[u] do begin
           neighbor:= Adj_list[u][i];
           if not(Visited[neighbor]) then begin
              append_queue(neighbor); Visited[neighbor]:= true;
              Distance[Start_vertex][neighbor] := Distance[Start_vertex][u] + 1;
           end;
        end;
     end;
  end(*BFS*);

 (* Main *)
  begin
     get_graph; (*procedure which reads or generates a graph*)
     for v:=1 to n do BFS(v);
     Sz := 0;
     for v:=1 to n do for j:=1 to Degree[v] do begin
        u:= Adj_list[v][j];
        if u>v then begin
           n1:=0; n2:=0;
           for i:=1 to n do begin
              diff := Distance[v][i] - Distance[u][i];
              if diff<0 then n1:= n1+1;
              if diff>0 then n2:= n2+1
           end;
        Sz := Sz + n1*n2;
        end;
     end;
     writeln( 'Szeged index =', Sz:0);
  end.
```

6.2. Dijkstrov algoritem za iskanje najkrajših poti

Z Dijkstrovim algoritmom izračunamo razdalje (in najkrajše poti) od izbranega vozlišča do vseh drugih vozlišč omrežja, grafa s poljubnimi nenegativnimi utežmi $\lambda(e) \geq 0$ ($\lambda(e) = \lambda(ij)$) na povezavah. Če algoritem poženemo iz vsakega vozlišča, izračunamo vse razdalje (točneje, na grafu s simetričnimi utežmi ($\lambda(ij) = \lambda(ji)$) izračunamo vsako razdaljo po dvakrat).

Opišimo najprej idejo algoritma. Podobno kot pri iskanju v širino začnemo iz vozlišča v, in gradimo drevo W. (Bistvena razlika je v izbiri naslednjega vozlišča iz vrste čakajočih.)

Med izvajanjem postopka je vozlišče lahko v enem od treh „stanj“.

- *novo* vozlišče (ki ga z algoritmom še nismo obravnavali),
- *znano* vozlišče (ki ni več novo) in
- *obdelano* vozlišče (z dokončno izračunano razdaljo).

Z drugimi besedami, množico vozlišč lahko v vsakem trenutku razdelimo v tri skupine: *že obdelana*, *znana* in *nova*. V zapisu algoritma bodo vozlišča nova, preden bodo prišla v vrsto, znana medtem ko bodo v vrsti in obdelana, potem ko jih bomo odstranili iz vrste.

Ko novo vozlišče v algoritmu prvič srečamo kot soseda obravnavanega vozlišča, ga proglasimo za znanega in mu damo začasno oznako, dotlej najkrajšo znano razdaljo.

Iz množice znanih vedno izberemo vozlišče u z najmanjšo začasno oznako in to oznako proglasimo za razdaljo. Vsem sosedam vozlišča pravkar izbranega vozlišča u, ki še nimajo dokončno izračunane razdalje (imenovane tudi potencial), določimo nove oznake tako, da razdaljo, ki je bila izračunana z doslej pregledanimi potmi popravimo, če smo našli krajšo pot. Formalno zapisano, oznaka := MIN(oznaka, potencial(u) + lambda(u,*)). Nazadnje vse nove sosede pravkar obdelanega vozlišča proglasimo za znana in jim določimo začasne oznake.

V zapisu algoritma bomo uporabili naslednje oznake:

$D(x)$ .... razdalja v do x

$L(x)$ .... začasna oznaka (razdalja) v do x

MIN(vrsta) ... izbere element vrste z minimalnim L

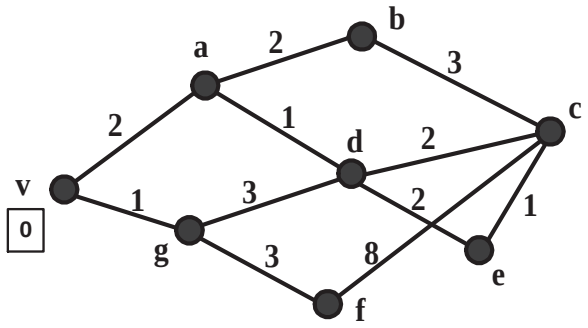
$Min(a, b)$ .... izračuna minimum vrednosti a in b

for vse $x \in V(G)$ **do** $L(x) := \infty$;
queue := (v); daj v med znana vozlišča; $L(v) := 0$;
while vrsta *queue* ni prazna **do begin**
    $x := MIN(queue)$; $D(x) := L(x)$
    for vse (neobdelane) sosede y vozlišča x **do begin**
        if y je nova **then**

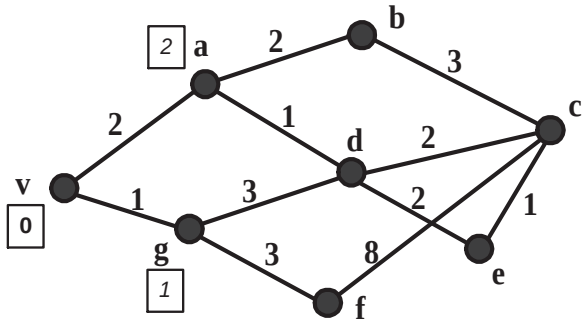
```
            dodaj y v vrsto queue; daj y med znana vozlišča;
        end
        L(y) := Min(L(y), D(x) + λ(x, y))
    end
    odstrani x iz vrste queue
end
```

Dokaz pravilnosti algoritma je v nalogi 6.3. Dijkstrov algoritem ima časovno zahtevnost $O(n^2)$, vse razdalje na omrežju torej lahko izračunamo v času $O(n^3)$.

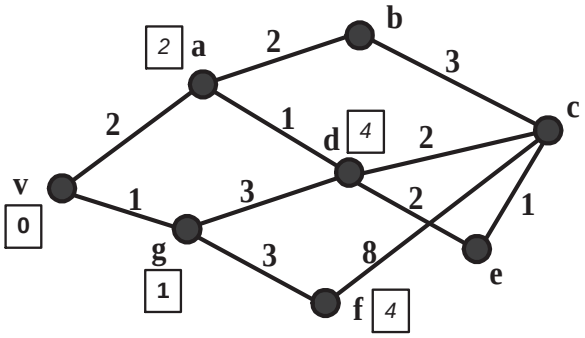
Poglejmo, kako deluje Dijkstrov algoritem na grafu



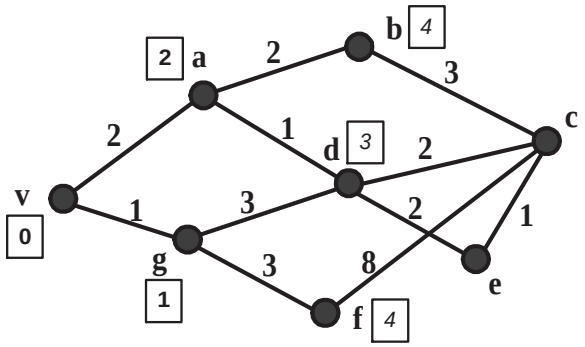
Vozlišče v je seveda na razdalji 0 do samega sebe, njegova soseda a in g pa sta dobila začasni oznaki oziroma razdalji. Izračunano razdaljo smo zapisali v okvir krepko, začasne razdalje pa kurzivno



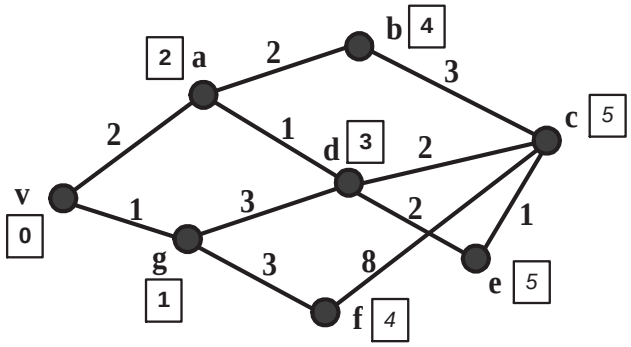
Vozlišče g je najbližje vozlišču v, zato je začasna oznaka kar prava razdalja. Vsem njegovim sosedom izračunamo ali popravimo začasne oznake.



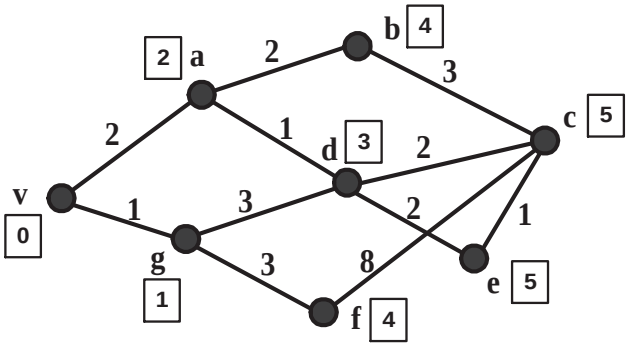
Vozlišče a je najbližje vozlišču v zato je njegova začasna oznaka kar prava razdalja. Vsem njegovim sosedom izračunamo ali popravimo začasne oznake. (Začasno oznako vozlišča d smo s 4 popravili na 3, vozlišče b dobi začasno oznako 4.



Vozlišče d je najbližje vozlišču v, zato je njegova začasna oznaka kar prava razdalja. Vozlišči c in e dobita začasni oznaki.



Vozlišči b in f imata oznaki 4. Izberemo enega od njih, na primer b, ugotovimo $d(v, b) = 4$ in pogledamo, če je treba popraviti začasno oznako pri c. Postopek ponovimo z vozliščem f.



Naslednji sta na vrsti vozlišči c in e in vse točke imajo izračunane razdalje.

Tabelarna metoda

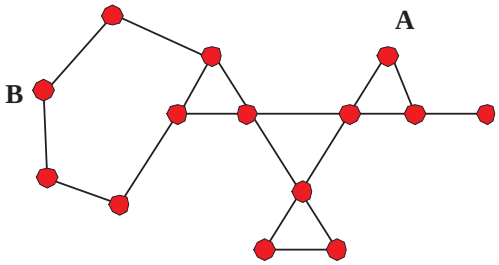
Pri uporabi zgornjega algoritma lahko hitro izgubimo pregled nad oznakami različnih vozlišč. Naslednja metoda z uporabo tabele je primeren način zapisovanja začasnih rezultatov, kadar algoritem izvajamo „peš“.

Vsaki točki omrežja priredimo stolpec tabele. Vsakič, ko kakšnemu vozlišču priredimo potencial, označimo z njim novo vrstico tabele. V novo vrstico zapišemo nove najkrajše (začasne) razdalje do vozlišč, ki jih lahko direktno dosežemo iz vozlišča z znanim potencialom. Naslednja tabela ustreza zgornjemu primeru.

vozlišče	v	a	b	c	d	e	f	g
v	**0**	2						1
g		2			4		4	**1**
a		**2**	4		3		4	
d			4	5	**3**	5	4	
b, f			**4**	5		5	**4**	
c, e				**5**			**5**	

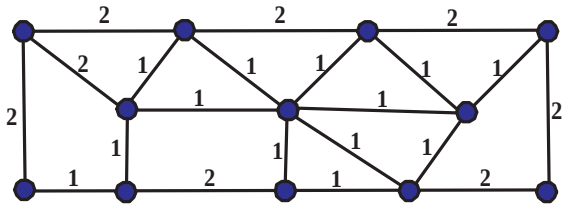
6.3. Naloge z rešitvami

6.1. Graf na sliki 6.1 predstavlja komunikacijsko omrežje. V vsakem vozlišču je strežnik, ki sporočila sprejema in jih po potrebi usmerja naprej. Recimo, da je strošek pošiljanja sporočila odvisen od števila strežnikov, ki morajo sporočilo odposlati naprej. Izračunaj matriko razdalj in določi premer (definiran v nalogi 1.16. na strani 29.). Kakšen je strošek pošiljanja iz vozlišča A v vozlišče B?



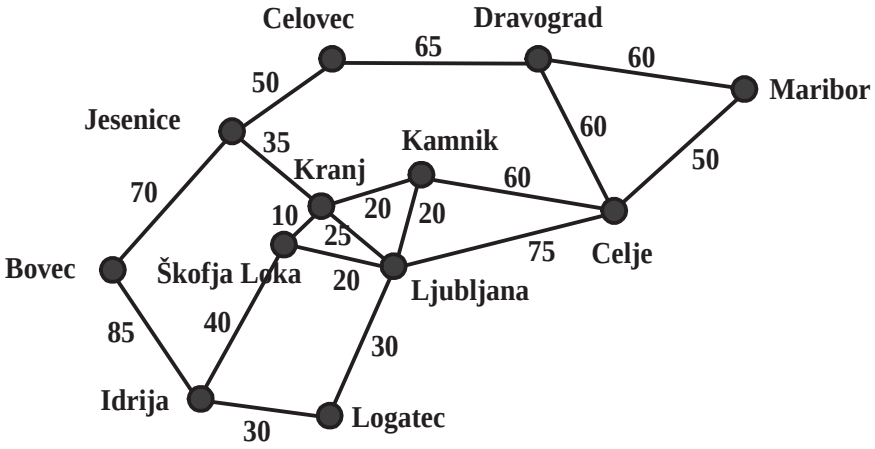
Slika 6.1:

6.2. Poišči vse razdalje med vozlišči grafa

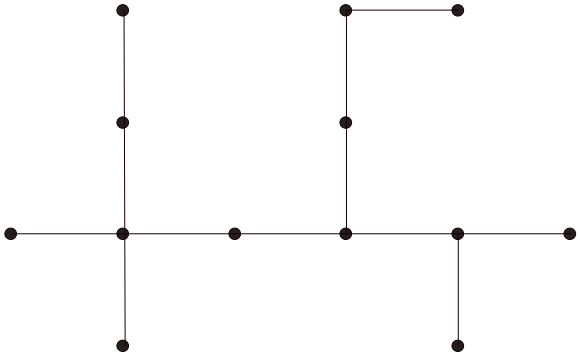


6.3. Dokaži pravilnost Dijkstrovega algoritma.

6.4. Poišči vse razdalje med mesti na omrežju

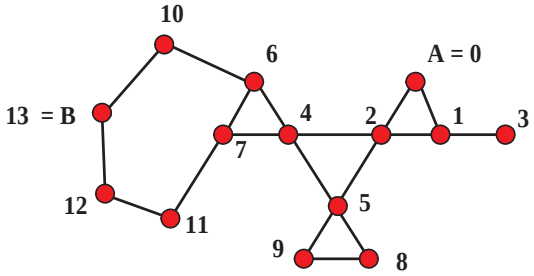


6.5. Izračunaj Wienerjev indeks drevesa



Rešitve

6.1. Označimo vozlišča kot kaže slika.



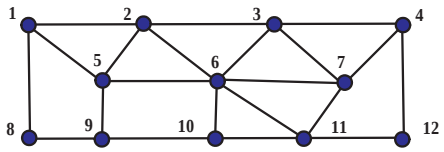
Potem je matrika razdalj (zaradi simetrije so zapisane samo vrednosti v zgornjem trikotniku)

$$\begin{bmatrix}
0 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 4 & 4 & 5 & 5 \\
 & 0 & 1 & 1 & 2 & 2 & 3 & 3 & 3 & 3 & 4 & 4 & 5 & 5 \\
 & & 0 & 2 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 4 & 4 \\
 & & & 0 & 3 & 3 & 4 & 4 & 4 & 4 & 5 & 5 & 6 & 6 \\
 & & & & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 \\
 & & & & & 0 & 2 & 2 & 1 & 1 & 3 & 3 & 4 & 4 \\
 & & & & & & 0 & 1 & 3 & 3 & 1 & 2 & 3 & 2 \\
 & & & & & & & 0 & 3 & 3 & 2 & 1 & 2 & 3 \\
 & & & & & & & & 0 & 1 & 4 & 4 & 5 & 5 \\
 & & & & & & & & & 0 & 4 & 4 & 5 & 5 \\
 & & & & & & & & & & 0 & 3 & 2 & 1 \\
 & & & & & & & & & & & 0 & 1 & 2 \\
 & & & & & & & & & & & & 0 & 1 \\
 & & & & & & & & & & & & & 0
\end{bmatrix}$$

Premer je enak največji razdalji, torej je enak 6, razdalja med A in B pa je enaka 5.

6.2.

Označimo vozlišča kot kaže slika.



Potem je matrika razdalj (zaradi simetrije so zapisane samo vrednosti v zgornjem trikotniku)

$$\begin{bmatrix}
0 & 2 & 4 & 6 & 2 & 3 & 4 & 2 & 3 & 4 & 4 & 6 \\
 & 0 & 2 & 4 & 1 & 1 & 2 & 3 & 2 & 2 & 2 & 4 \\
 & & 0 & 2 & 2 & 1 & 1 & 4 & 3 & 2 & 2 & 4 \\
 & & & 0 & 3 & 2 & 1 & 5 & 4 & 3 & 2 & 2 \\
 & & & & 0 & 1 & 2 & 2 & 1 & 2 & 2 & 4 \\
 & & & & & 0 & 1 & 3 & 2 & 1 & 1 & 3 \\
 & & & & & & 0 & 4 & 3 & 2 & 1 & 3 \\
 & & & & & & & 0 & 1 & 3 & 4 & 6 \\
 & & & & & & & & 0 & 2 & 3 & 5 \\
 & & & & & & & & & 0 & 1 & 3 \\
 & & & & & & & & & & 0 & 2 \\
 & & & & & & & & & & & 0
\end{bmatrix}$$

6.3. Zapis Dijkstrovega algoritma najprej opremimo z oznako vrstice, na katero se bomo sklicevali v dokazu.

za vse $x \in V(G)$, $L(x) := \infty$;
$queue := (v)$; daj v med znana vozlišča; $L(v) := 0$;
while queue is not empty **do begin**
(*) $x := MIN(queue)$; $D(x) := L(x)$
for vse (neobdelane) sosede y vozlišča x **do begin**
if y je nova **then**
dodaj y v queue; daj y med znana vozlišča
end
$L(y) := Min(L(y), D(x) + \lambda(x, y))$
end
odstrani x iz vrste;
end

Označimo z R množico vozlišč z izračunanim D, z $V \setminus R$ pa množico z še nedoločeno vrednostjo D.

Dokažimo, da vedno ko izvajamo (*) velja

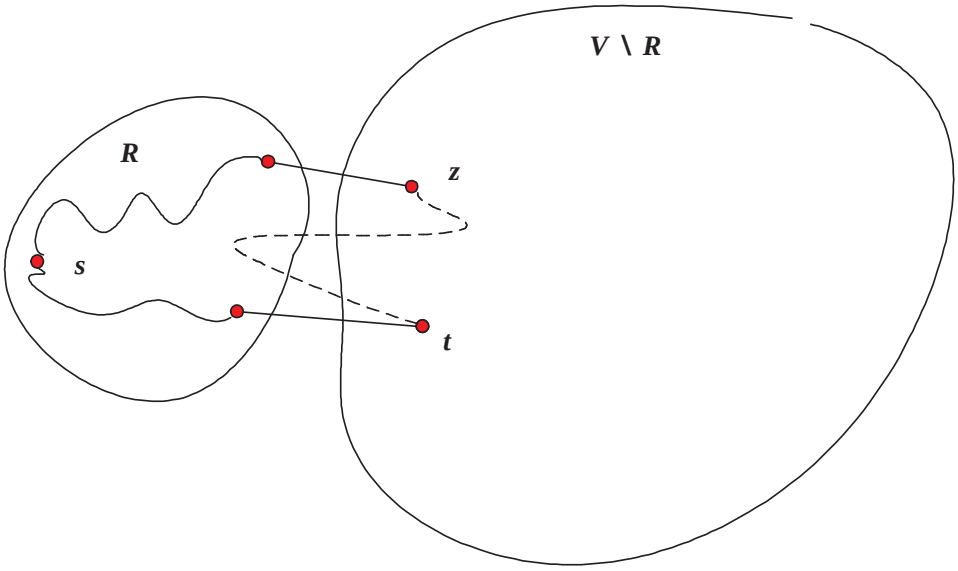
(a) če je $x \in R$ in $y \in V \setminus R$, potem je $L(x) \leq L(y)$.
(b) za vsa vozlišča $x \in R$ je $D(x) = L(x)$ dolžina najkrajše poti med v in x v grafu G.

(c) za vsa vozlišča $y \in V \setminus R$ je $L(y)$ dolžina najkrajše poti na grafu, induciranem na vozliščih $R \cup \{y\}$. Če je $L(y) < \infty$, potem je $L(y) = L(x) + \lambda(x, y)$, za neko vozlišče $x \in R$.

Pri prvem izvajanju vrstice (*) trditve očitno veljajo. (Vrsta ima samo en element, v in $D(V) = L(v) = 0$.)

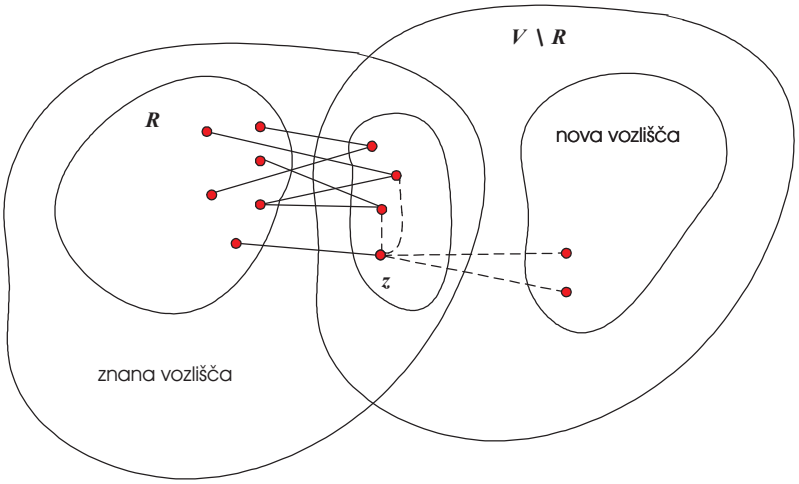
Privzemimo, da na mestu (*) velja (a) in dokažimo, da preostale vrstice ohranjajo veljavnost trditve (a). Recimo, da je bilo izbrano vozlišče z pri izbiri elementa vrste *queue* z minimalnim $L(.)$. Za poljubni vozlišči $x \in R$ in $y \in V \setminus R$ je $L(x) \leq L(z) \leq L(y)$, zaradi (a) in načina izbire z. Nekaterim vozliščem $y \in V \setminus R$ se lahko v zanki spremeni vrednost L, vendar se zaradi $L(y) := Min(L(y), D(z) + \lambda(z, y)) \geq D(z) = L(z)$ veljavnost (a) ohranja.

Zdaj pa privzemimo, da na mestu (*) veljajo (a), (b) in (c) ter dokažimo, da preostale vrstice ohranjajo veljavnost trditve (b). Dovolj je obravnavati samo vozlišče z. Recimo, da obstaja krajša pot med v in z v grafu G. Naj bo t prvo vozlišče na tej najkrajši poti (v smeri od v proti z), ki ni v R. Potem pa je razdalja med v in t krajša kot razdalja med v in z, kar je v protislovju z minimalnostjo z. (Glej sliko 6.2.)



Slika 6.2: Dokaz pravilnosti Dijkstrovega algoritma, (b).

Nazadnje dokažimo še, da se ohranja veljavnost (c). (Glej sliko 6.3.) Ko izberemo vozlišče z, popravimo L za sosede z med znanimi vozlišči v $V \setminus R$ in prvič določimo vrednosti L za prvič srečana nova vozlišča. Za nova vozlišča velja, da gre edina (torej tudi najkrajša) pot do v preko z, torej je L res dolžina najkrajše poti pri novem R. Za vozlišča iz $V \setminus R$ smo z izbiro z morda vpeljali nove poti do v. Če so te poti krajše kot prej, se je zaradi $L(y) := Min(L(y), D(x) + \lambda(x, y))$ v algoritmu pravilno zmanjšala tudi vrednost L.



Slika 6.3: Dokaz pravilnosti Dijkstrovega algoritma, (c).

6.4.

	Mb	Dr	Kl	Je	Bo	ŠL	Kr	Lj	Ka	Ce	Lo	Id
Maribor	0	60	125	165	235	140	130	125	110	50	155	180
Dravograd		0	65	115	185	150	140	135	120	60	165	190
Celovec (Kl)			0	50	120	95	85	110	105	125	140	135
Jesenice				0	70	45	35	60	55	115	90	85
Bovec					0	115	105	130	125	185	115	85
Škofja Loka						0	10	20	30	90	50	40
Kranj							0	25	20	80	55	50
Ljubljana								0	20	75	30	60
Kamnik									0	60	50	70
Celje										0	105	130
Logatec											0	30
Idrija												0

6.5. Vsota vseh razdalj je enaka 258.

Opomba. Vse razdalje v grafu lahko izračunamo v času $O(mn)$, na primer tako, da iz vsakega vozlišča izračunamo razdalje do vseh ostalih z algoritmom Iskanje v širino (s strani 113). Na drevesih pa je mogoče Wienerjevo število izračunati še hitreje kot v splošnem, v času reda $O(n)$, z elegantnim algoritmom Moharja in Pisanskega[3], ki je izboljšana in poenostavljena verzija algoritma Canfielda, Robinsona in Rouvraya.

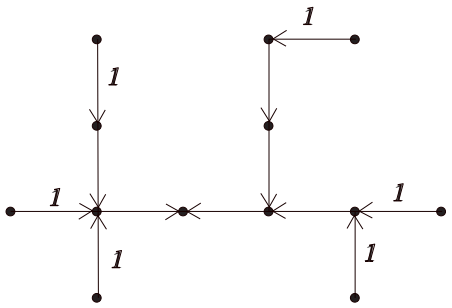
1. Najprej drevo zapišemo kot usmerjeno drevo s korenom.[4] Predpostavimo, da je drevo dano v obliki vektorja T_i, $i = 1, 2, \ldots, n-1$, kjer $T_i = j$ pomeni, da je j oče i-ja. Take predstavitve drevesa ni

[3]B. Mohar, T. Pisanski, How to Compute the Wiener Index of a Graph, *Journal of Mathematical Chemistry* 2 (1988) 267-277.

[4]Drevo s korenom je formalno gledano urejen par (drevo, vozlišče). Na drevesu s korenom lahko (po analogiji z

težko dobiti. Izberemo poljubno točko za koren (ta nima nobenega očeta). Imenujmo jo v. Vsi sosedi točke v so sinovi, torej postavimo $T_x = v$, kazalec iz x nazaj k v. Podobno sinovi sinov pokažejo na svoje očete, in tako dalje.

2. Za vsako točko izračunamo število sinov, torej število puščic, ki kažejo nanjo. (To lahko storimo kar sproti, ko zapisujemo vektor T.)
3. Postavimo vse liste drevesa v vrsto.
4. Vzamemo točko iz čakalne vrste. Zbrišemo ta list drevesa (oz. povezavo do njega) in izračunamo $w(e)$ (kot je to opisano spodaj). Novi list, če se je pojavil, vstavimo v vrsto. Ponavljamo, dokler vrsta ni prazna.
5. Seštejemo vse uteži povezav $w(e)$.

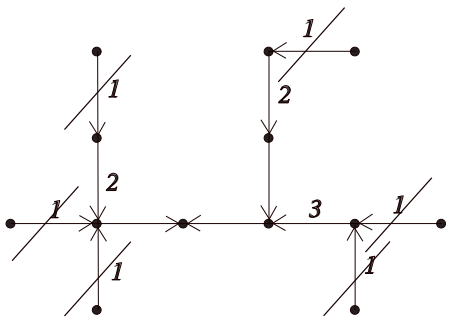


Slika 6.4: Računanje Wienerjevega indeksa drevesa. Prvi korak.

Izračunajmo Wienerjevo število za graf na sliki 6.4. Na povezave bomo zapisovali število točk v tisti „polovici“ drevesa, ki smo ga že zbrisali. Označimo to število s $t(e)$. Iz tega zapisa preberemo utež povezave, $w(e) = t(e)(n - t(e))$. Nadalje t za usmerjeno povezavo $e = uv$, ki je prišla na vrsto za brisanje, dobimo tako, da seštejemo t-je povezav xu in prištejemo 1.

Drevo na sliki 6.4 smo že usmerili in z utežmi $t = 1$ (v katerem koli vrstnem redu) označili povezave, ki vodijo iz listov.

Liste zbrišemo in nadaljujemo postopek z listi novega drevesa (slika 6.5):

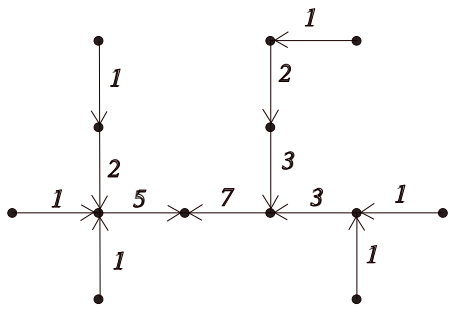


Slika 6.5: Računanje Wienerjevega indeksa drevesa. Nadaljevanje.

Na koncu dobimo uteži (slika 6.6):

---

rodovnimi drevesi) naravno definiramo relacijo oče–sin (ali predhodnik–naslednik). Koren drevesa je oče vseh svojih sosed, vozlišča brez sinov pa so listi drevesa.



Slika 6.6: Računanje Wienerjevega indeksa drevesa. Dobljene uteži.

Wienerjevo število je torej enako $W(T) = \sum_e t(e)(n - t(e)) = 6 \times 1(13 - 1) + 2 \times 2(13 - 2) + 2 \times 3(13 - 3) + 1 \times 5(13 - 5) + 1 \times 7(13 - 7) = 258$.

7

LOKALNO ISKANJE

Kot smo že večkrat omenili, so mnogi problemi kombinatorične optimizacije NP-težki. Praktični problemi so poleg tega tudi pogosto nelinearni, slabo strukturirani in imajo večkriterijske namenske funkcije, zaradi česar so precej kompliciranі za modeliranje in reševanje. Zato je za mnoge primere problemov bilo razvitih veliko specializiranih algoritmov, s katerimi se lahko sorazmerno uspešno lotimo kar velikih nalog. Primer dobro raziskanega problema, ki smo ga že srečali, je problem trgovskega potnika.[1] Po drugi strani pa je v zadnjih desetletjih zelo živahno področje raziskav študij splošnih metod za reševanje problemov kombinatorične optimizacije. Splošne metode včasih imenujemo tudi **metahevristike**.[2] Tu si bomo podrobneje ogledali metodo lokalnega iskanja, na kateri temeljijo še nekatere druge zelo pogosto uporabljane metahevristike: algoritem ohlajanje (angl. simulated annealing),[3] tabu iskanje (angl. tabu search)[4] in genetski algoritmi.[5] Lokalno iskanje je splošna metoda za približno reševanje optimizacijskih problemov.[6] Pri opisu metode privzamemo, da je na množici dopustnih rešitev definirana neka struktura okolic,[7] zato bomo v nadaljevanju najprej definirali nekaj splošnih pojmov v zvezi optimizacijskimi problemi, potem pa se bomo lotili obravnave algoritma.

7.1. Splošni problem kombinatorične optimizacije

Mnoge diskretne optimizacijske probleme lahko obravnavamo kot posebne primere *splošnega problema kombinatorične optimizacije*, ki ga bomo definirali v tem razdelku.

Spomnimo se, da je optimizacijski problem (definiran na strani 102) določen s podatki in nalogo izračunati optimalno vrednost namenske funkcije. Zdaj bomo nalogi optimizacijskega problema priredili množico *dopustnih rešitev*, to je objektov, ki jim lahko izračunamo vrednost namenske funkcije in iz njih preberemo (približno) rešitev optimizacijske naloge. Pogosto lahko istemu optimizacijskemu problemu na različne načine definiramo množice dopustnih rešitev (kot bomo videli iz primerov).

---

[1] Glej poglavje 2 in [E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan in D.B.Shmoys, The Traveling Salesman Problem, John Wiley & Sons, New York, 1985].

[2] Hevristika ali hevristični algoritem je algoritem, ki se pogosto dobro obnese v praksi, vendar nimamo niti dokaza, da so dobljene rešitve dobre aproksimacije. Meta-hevristika je splošna metoda, ki jo lahko s primerno prilagoditvijo uporabimo na veliki množici problemov.

[3] Glej tudi stran **??** in reference tam.

[4] [F.Glover, M.Laguna, Tabu search, Kluwer, Boston 1998.] Slovensko tudi *metoda prepovedanih smeri.*

[5] [D.E.Goldberg, Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, Reading (Maš.) 1998.] Genetski algoritmi so poseben primer evolucijskih algoritmov. Glej tudi [M.Brezočnik, Uporaba genetskega programiranja v inteligentnih proizvodnih sistemih, Fakulteta za strojništvo, Maribor 2000].

[6] Dodatna literatura o lokalnem iskanju in posplošitvah: [E.Aarts in E.Lenstra (ur.), Local Search in Combinatorial Optimization, John Wiley & sons, New York 1997], [M.Del Amico, F.Maffioli in S.Martello (ur.), Annotated Bibliography in Combinatorial Optimization, John Wiley & sons, New York 1997].

[7] Lahko rečemo, da je s tem definirana *topologija* na množici dopustnih rešitev.

Definirajmo: **Splošni problem kombinatorične optimizacije** je družina nalog. Naloga je par $(\mathbf{S}, f)$, kjer je $\mathbf{S}$ množica **dopustnih rešitev**, f pa je preslikava $f : \mathbf{S} \to \mathbb{R}$, ki jo imenujemo **namenska funkcija**. Treba je poiskati tako dopustno rešitev $x_{min} \in \mathbf{S}$, za katero velja

$$f_{min} = f(x_{min}) = \min_{x \in \mathbf{S}} f(x) \tag{7.1}$$

Problem iskanja maksimuma definiramo analogno, je pa trivialno ekvivalenten problemu minimuma na isti množici $\mathbf{S}$ z namensko funkcijo $\tilde{f} = -f$.

Čeprav končna, pa je množica $\mathbf{S}$ pogosto zelo obsežna (njena velikost je pogosto eksponentna funkcija velikosti problema). Na primer: pri problemu barvanja točk grafa imamo pri n točkah k^n različnih barvanj s k barvami. V tej množici barvanj iščemo dobra barvanja, pri katerih sta poljubni sosednji točki pobarvani z različnima barvama. Pri formulaciji problema trgovskega potnika, v kateri so dopustne rešitve permutacije vozlišč, je število dopustnih rešitev kar $n!$, kolikor je vseh permutacij. Za drugi primer omenimo problem trdnjav na šahovnici, kjer moramo na šahovnici v obliki n-razsežne kocke z robom 3 s čim manj trdnjavami napasti vsa polja. Podatek je samo število n (dimenzija šahovnice), za množico dopustnih rešitev pa običajno vzamemo vse možne postavitve trdnjav, ki jih je kar 2^{3^n} (glej tudi nalogo 7.6.).

Zaradi obsežnosti pregledovanje vse množice $\mathbf{S}$ običajno ne pride v poštev. Pri NP–težkih problemih je pregled množice $\mathbf{S}$ algoritem eksponentnega reda. In če privzamemo, da velja hipoteza P≠NP, potem takega algoritma tudi ni mogoče odkriti.

Zaradi velikosti množice dopustnih rešitev včasih definiramo sorazmerno preprosto, predvsem pa sorazmerno hitro operacijo, ki nam iz dane dopustne rešitve generira dopustno rešitev, ki se od nje „malo“ razlikuje. Takima dopustnima rešitvama potem rečemo, da sta sosednji pri izbrani definiciji operacije, ki generira sosede. Tako definiramo **relacijo sosednosti** na množici dopustnih rešitev. V splošnem si lahko predstavljamo, da je k vsaki dopustni rešitvi prirejena množica sosed, ali pa da obstaja algoritem, ki k dani dopustni rešitvi generira elemente iz množice sosed.

S tem ko smo definirali **soseščino** dopustne rešitve $x \in \mathbf{S}$ smo na množico $\mathbf{S}$ na naraven način vpeljali razdaljo. Z izbiro soseščin $\mathbf{N}(x)$ je definiran graf s povezavami $x \sim y$ ($x \sim y \iff y \in \mathbf{N}(x)$) na množici točk $\mathbf{S}$, ki ga opremimo z običajno metriko najkrajših poti, le–ta pa določa metrično topologijo. V opisani metriki so $\mathbf{N}(x)$ ravno enotske krogle. Kot bomo videli na primerih, je za dani problem mogoče na več načinov smiselno definirati pojem soseščine. V takih primerih nam različne izbire v splošnem dajo različne topologije na množici $\mathbf{S}$, s tem pa tudi različno obnašanje splošnih algoritmov, ki jih bomo definirali v nadaljevanju. Nas bo tu poleg topologije seveda zanimala tudi učinkovitost (hitrost) pregledovanja soseščine in generiranja dopustnih rešitev. Običajno lahko pričakujemo, da bomo imeli ali hiter algoritem in „grdo“ topologijo (z mnogimi lokalnimi ekstremi) ali pa obratno, „lepo“ topologijo, vendar počasen algoritem.

Tako splošna definicija relacije sosednosti seveda pove bolj malo, zato si poglejmo nekaj tipičnih primerov.

Primer 1.

PROBLEM: k-barvanje

PODATKI: enostaven graf G

VPRAŠANJE: Ali obstaja dobro k-barvanje grafa G?

DOPUSTNE REŠITVE: vsa k-barvanja

DEFINICIJA SOSEDOV: barvanji sta sosednji, če se razlikujeta v barvi največ enega vozlišča

NAMENSKA FUNKCIJA: število povezav z enako pobarvanima krajiščema

ali pa

DOPUSTNE REŠITVE: delna dobra k-barvanja[a]

DEFINICIJA SOSEDOV: barvanji sta sosednji, če se razlikujeta v barvi največ enega (pobarvanega ali nepobarvanega) vozlišča

NAMENSKA FUNKCIJA: število pobarvanih vozlišč

[a]Delno barvanje je preslikava, ki je definirana na podmnožici množice vozlišč, na preostanku pa (še) ni definirana. Delno barvanje je dobro, če med pobarvanimi pari vozlišč ni enako pobarvanih sosedov.

ali pa

DOPUSTNE REŠITVE: dobra k-barvanja

DEFINICIJA SOSEDOV: barvanju c priredimo sosednje barvanje tako, da izberemo poljubno vozlišče, ki ima enako pobarvanega soseda, in mu spremenimo barvo.[a]

NAMENSKA FUNKCIJA: število pobarvanih vozlišč

[a]Po tej definiciji dobra barvanja nimajo sosedov!

PROBLEM: barvnost grafa

PODATKI: enostaven graf G

VPRAŠANJE: Kolikšno je kromatično število grafa G?

DOPUSTNE REŠITVE: vsa dobra barvanja

DEFINICIJA SOSEDOV: barvanji sta sosednji, če se razlikujeta v barvi največ enega vozlišča

NAMENSKA FUNKCIJA: število uporabljenih barv

ali

DOPUSTNE REŠITVE: vsa k-barvanja

DEFINICIJA SOSEDOV: barvanju c priredimo sosednje barvanje tako, da izberemo poljubno vozlišče, ki ima enako pobarvanega soseda, in mu spremenimo barvo.

NAMENSKA FUNKCIJA: število uporabljenih barv + število povezav z enako pobarvanima krajiščema

Primer 2.

PROBLEM: minimalno vpeto drevo

PODATKI: enostaven graf G z utežmi na povezavah

CILJ: Poišči vpeto drevo z minimalno utežjo!

DOPUSTNE REŠITVE: vsa vpeta drevesa

DEFINICIJA SOSEDOV: drevesi T_1 in T_2 sta sosednji, če obstajata povezavi $e_1 \in E(T_1)$ in $e_2 \in E(T_2)$, tako da je $E(T_1) \setminus e_1 = E(T_2) \setminus e_2$

NAMENSKA FUNKCIJA: vsota uteži povezav vpetega drevesa

Primer 3. Problem trgovskega potnika smo v poglavju 2 predstavili na dva načina. Tu izberimo predstavitev s permutacijami in zapišimo morda najbolj pogosto uporabljano definicijo sosednosti na množici dopustnih rešitev.

PROBLEM: trgovski potnik

PODATKI: enostaven graf G z utežmi na povezavah

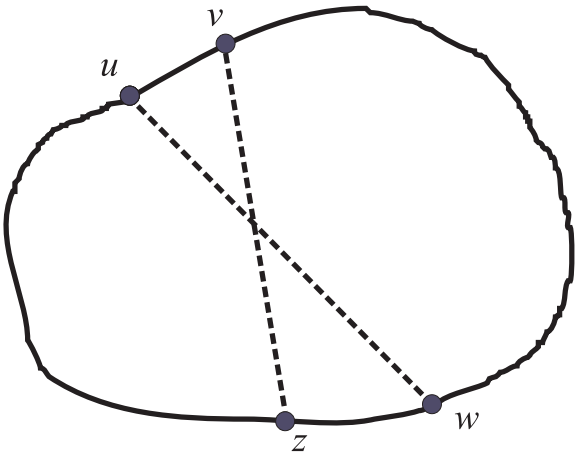
CILJ: Poišči Hamiltonov cikel z minimalno utežjo!

DOPUSTNE REŠITVE: vse permutacije vozlišč

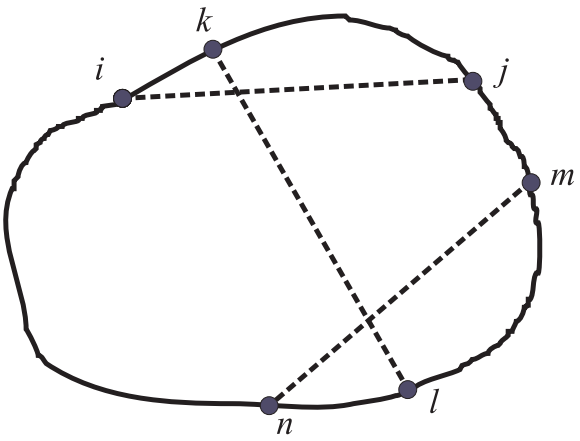
DEFINICIJA SOSEDOV: 2-opt

NAMENSKA FUNKCIJA: vsota uteži cikla, ki ga določa permutacija

Dopustni rešitvi priredimo (2-opt-)sosednjo rešitev takole: na ciklu izberemo dve nesosednji povezavi, recimo $e = uv$ in $f = wz$ in povezavi e in f zamenjamo s povezavama uw in zv. Glej tudi nalogo 7.1.



Podobno lahko definiramo 3-opt ali v splošnem r-opt sosede, če dovolimo zamenjave treh (ali r) povezav. Na sliki je primer 3-opt soseda, kjer smo povezave ij, kl in mn zamenjali z ik, jm in ln .



Pri zamenjavah moramo paziti, da namesto enega ne dobimo dveh ali več ciklov.

Ker so okolice 3-opt lahko že zelo obsežne, je Or 1976 predlagal modificirano definicijo, pri kateri pride v poštev samo majhen del 3-opt sosedov. Možne in tudi uporabne so še mnoge druge definicije okolic za problem trgovskega potnika.[8]

Potem ko privzamemo, da smo definirali sosede dopustnih rešitev, lahko definiramo *lokalni minimum* na množici **S**: Točka y je **lokalni minimum**, če velja $f(y) \leq f(x)$ za vsak $x \in \mathbf{N}(y)$.

Analogno lahko definiramo **lokalni maksimum**.

Zdaj lahko zapišemo enostaven algoritem, ki ga imenujemo algoritem **lokalna optimizacija** ali

[8]Glej strani 218-222 v [E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan in D.B.Shmoys, The Traveling Salesman Problem, John Wiley & Sons, New York, 1985]. Primer nestandardne okolice je uporabljen v [J.Brest in J.Žerovnik, An Approximation algorithm for the Asymmetric Traveling Salesman Problem, Ricerca Operativa 28 (1999) 59-67].

lokalno iskanje (angl. local search, neighborhood search, iterative improvement, včasih tudi multistart algorithm), ki ga lahko uporabimo za reševanje kateregakoli problema kombinatorične optimizacije.

Algoritem LS: (lokalna optimizacija)
repeat
  $x := x_0$; (*izberi začetno stanje x_0*)
  repeat
    poišči naslednjega soseda, $y \in N(x)$
    if $f(y) < f(x)$ (*novi sosed je boljši*) **then** $x := y$; (*premakni se*)
  until množica sosedov je pregledana
until **ni** preveč korakov

Enostaven premislek nas pripelje do ugotovitve, da se notranja zanka konča v lokalnih minimumih. V nobenem primeru v notranji zanki algoritem ne more iz lokalnega minimuma. To, da se „ustavi“ v lokalnih ekstremih, je tudi glavna zamera, ki jo temu algoritmu namenjajo kritiki.

O časovni zahtevnosti tako splošnega algoritma se ne da povedati veliko. Prav mogoče je, da je celo za eno samo iskanje lokalnega optimuma potreben nepolinomski računski čas. Zadostuje, da so okolice dovolj „bogate“. O tem nas prepriča trivialni primer: vedno je mogoče definirati soseščino tako, da so vse dopustne rešitve dosegljive ena iz druge. V tem primeru potrebujemo za pregled ene same okolice $O(|\mathbf{S}|)$ časa! (Za tolažbo pa bomo optimalno rešitev zanesljivo dobili. Glej tudi nalogo 7.2..)

V primerih, ko so soseščine zelo velike, včasih uporabimo različico algoritma, ki ne pregleduje vse množice $\mathbf{N}(x)$. To naredimo tako, da namesto vrstice „poišči naslednjega soseda“ uporabimo algoritem $\mathcal{N}$, ki nam vrne naključno izbranega soseda. Potem je potrebno spremeniti tudi pogoj za izstop iz notranje zanke, na primer v „**repeat ... until** *dovolj dolgo so bili generirani sami slabši sosedi*“. „*dovolj dolgo*“ lahko definiramo različno glede na to, kako hiter algoritem želimo in glede na to, kako zanesljivo in koliko dobre rešitve potrebujemo. Opisana različica je osnova za posplošitev na takoimenovane „plezalne“ algoritme (angl. hill climbing algorithms), med katerimi je verjetno najbolj znan algoritem Ohlajanje, ki ga bomo podrobneje obravnavali kasneje (na strani **??**).

7.2. Naloge z rešitvami

7.1. Za nalogo trgovskega potnika, dano z razdaljami (London – Birmingham = 195, London – Leeds = 317, London – Manchester = 336, Birmingham – Manchester = 153, Birmingham – Leeds = 196, Leeds – Manchester = 71, Glasgow – Edinburgh = 76, Glasgow – London = 652, Glasgow – Leeds = 400, Glasgow – Manchester = 345, Glasgow – Birmingham = 467, Edinburgh – London = 651, Edinburgh – Leeds = 403, Edinburgh – Manchester = 350, Edinburgh – Birmingham = 471)

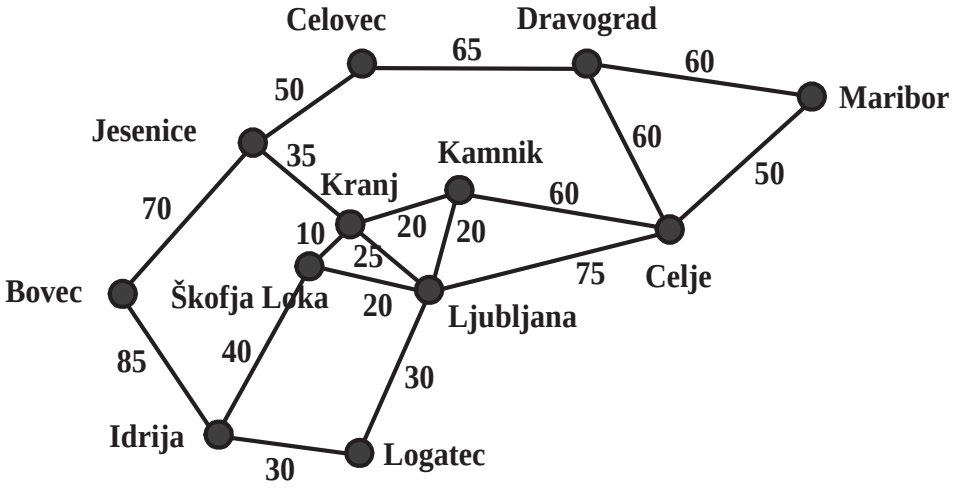
(a) Nariši ustrezni uteženi graf (omrežje)!

(b) Definiraj množico dopustnih rešitev!

(c) Poišči optimalno rešitev!

(d) Katere dopustne rešitve so v (2-opt) okolici dopustne rešitve (London - Leeds - Glasgow - Edinburgh - Manchester - Birmingham)?

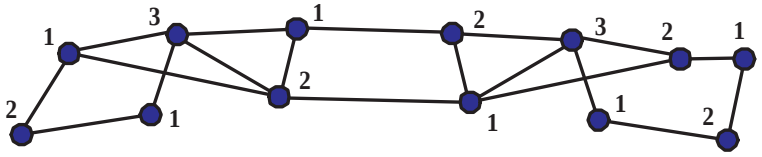
7.2. Koliko je Hamiltonovih obhodov na omrežju na sliki spodaj? Koliko je dopustnih rešitev, če vzamemo za dopustne rešitve vse permutacije mest.



7.3. Stroj lahko pripravimo za izdelovanje k vrst izdelkov: I_1,I_2,I_3,...,I_k. Označimo z S_i stanje stroja, ko je pripravljen za izdelovanje izdelka I_i. Ko stroj ne obratuje, mora biti v mirujočem stanju S_0. Sprememba nastavitve stroja iz stanja S_i v stanje S_j traja $t(i,j)$, $i = 0,1,\ldots,k$, $j = 0,1,\ldots,k$ časa. Tipično dnevno naročilo je seznam K_1, K_2, K_3, ..., K_k, kjer K_i pomeni število naročenih izdelkov I_i. Želimo čim bolj skrajšati čas obratovanja.

1. Formuliraj nalogo kot optimizacijski problem (podatki, namenska funkcija, naloga). Ali je problem soroden kateremu od znanih optimizacijskih problemov?
2. Definiraj primerno množico dopustnih rešitev.

7.4. Na sliki je dobro 3-barvanje grafa



(a) Privzemimo, da smo izbrali za dopustne rešitve vsa 3-barvanja in da sta dve dopustni rešitvi sosednji, če se razlikujeta v barvi enega vozlišča. Koliko je sosedov dopustne in optimalne rešitve na sliki?

(b) Privzemimo, da smo izbrali za dopustne rešitve vsa 2-barvanja in da sta dve dopustni rešitvi sosednji, če se razlikujeta v barvi enega vozlišča. Stroškovna funkcija naj bo, kot običajno, število enobarvnih povezav. Poišči kakšno lokalno optimalno dopustno rešitev.

7.5. V čem se spodaj zapisani algoritem razlikuje od Algoritma LS?

Algoritem LS2: (lokalna optimizacija)
repeat
  izberi začetno stanje x_0
  repeat
    poišči najboljšega soseda in se premakni, če je ta boljši
  until najboljši sosed ni boljši
until ni preveč korakov

7.6. V n dimenzijah imejmo n–dimenzionalno "šahovnico" s $3 \times 3 \times \cdots \times 3 = 3^n$ polji. Vsako polje šahovnice lahko opremimo s koordinatami. Ker smo se omejili na dolžino šahovnice 3, je koordinata eno od števil 0, 1, 2. Poljubno polje 3–dimenzionalne šahovnice tedaj enolično opišemo s trojico (i, j, k), na primer $(0, 0, 0)$, $(0, 2, 1)$, $(2, 2, 0)$ itd. Položaj trdnjave opišemo s poljem, na katero je postavljena. Trdnjava na dvodimenzionalni šahovnici napada vsa polja, ki se od njenega položaja razlikujejo v natanko eni koordinati. In posplošimo: trdnjava na n–dimenzionalni šahovnici napada vsa polja, ki se od njenega položaja razlikujejo v natanko eni koordinati.

(a) Zapiši nalogo formalno kot optimizacijski problem.

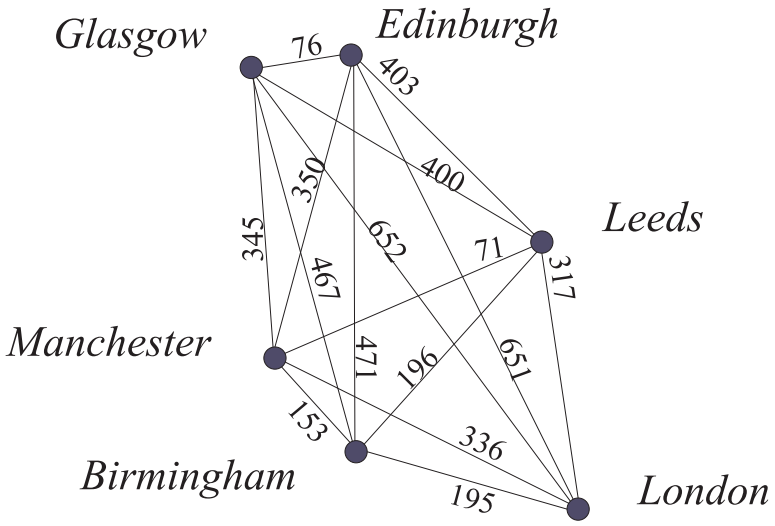
(b) Izberi primerno definicijo za množico dopustnih rešitev in za sosednost med dopustnimi rešitvami.

(c) Reši nalogo za $n = 2, 3$ in 4.

Rešitve

7.1.

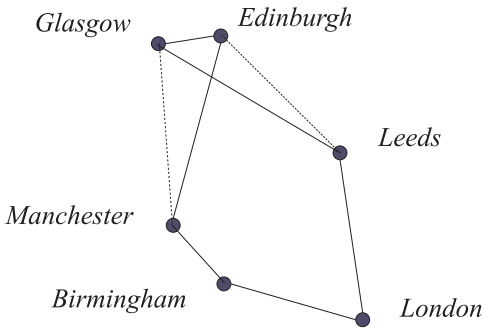
(a)



(b) Dopustne rešitve so vse permutacije šestih mest. Množica ima 6! = 720 elementov.

(c) Optimalna rešitev je London - Leeds - Manchester - Glasgow - Edinburgh -Birmingham z dolžino 1475.

(d) V (2-opt) okolici dopustne rešitve (London - Leeds - Glasgow - Edinburgh - Manchester - Birmingham) je 9 dopustnih rešitev (pri izbiri dveh nesosednjih povezav vsaka povezava lahko nastopa v paru s tremi drugimi. Tako pregledamo vsako možnost po dvakrat, $6 \times 3 : 2 = 9$). Ena od izbir, v kateri smo Leeds-Glasgow in Edinburgh-Manchester zamenjali z Leeds-Edinburgh in Glasgow-Manchester, je na sliki



7.2. Vseh permutacij je 12!.

Na grafu imamo samo en Hamiltonov obhod. Če bi dopustne rešitve definirali kot množico Hamiltonovih obhodov, bi imela ta samo en element.

7.3.

PROBLEM: nastavitve stroja

PODATKI: matrika razsežnosti $k \times k$ z elementi $t(i, j)$

CILJ: Poišči zaporedje (permutacijo) oznak izdelkov $\sigma(0) = 0, \sigma(1), \sigma(2), \ldots, \sigma(k)$, tako da bo vsota $\sum_{i=1}^{k} t(\sigma(i-1), \sigma(i)) + t(\sigma(k), 0)$ minimalna.

DOPUSTNE REŠITVE: vse permutacije izdelkov

NAMENSKA FUNKCIJA: vsota uteži $t(i, j)$, ki jih določa permutacija

Če vzamemo za vozlišča omrežja izdelke in začetno stanje, za uteži pa vrednosti $t(i, j)$, vidimo, da je problem ekvivalenten problemu trgovskega potnika.

7.4. (a) Sosedov (vsake dopustne rešitve) je $(k-1)^n = 2^{13}$. (b) Optimalna rešitev ima tri enobarvne povezave. (Po ena enobarvna povezava bo zagotovo na vsakem lihem ciklu. Nekateri lihi cikli imajo skupne povezave.)

7.5. Algoritem vedno pregleda vso okolico dopustne rešitve in izbere dopustno rešitev z najboljšo vrednostjo namenske funkcije. (Prejšnji algoritem je izbral prvo sosednjo dopustno rešitev, ki je izboljšala vrednost namenske funkcije.) Tej varianti lokalnega iskanja bi lahko rekli lokalno iskanje v smeri najhitrejšega padanja (angl. steepest descent).

7.6.

(a)

PROBLEM: problem trdnjav

PODATKI: n

CILJ: Poišči minimalno število trdnjav, ki jih lahko postavimo na n–dimenzionalno šahovnico (s stranicami dolžine 3) tako, da bodo vsa polja šahovnice napadena.

Opomba. V Angleščini problem imenujejo „the Football–Pool problem“, kar bi lahko za silo prevedli kot „problem športne napovedi“. Velja namreč naslednje: vsako rešitev problema (množica n-teric ničel, enic in dvojk) lahko uporabimo za športno napoved z n-pari. Rešitev ima lepo lastnost: vnaprej vemo, da bomo gotovo vsaj enkrat zadeli drugo nagrado. Drugače povedano, v naši rešitvi bo vsaj en stolpec, s katerim bomo zgrešili največ en izid. Ker plačamo pri športni napovedi vsak stolpec (torej vsako n-terico), je optimalna rešitev najcenejša rešitev z lastnostjo, da vedno zadane vsaj drugo nagrado.

Opomba. V jeziku teorije grafov lahko nalogo povemo tudi takole: poišči dominantno število kartezičnega produkta n faktorjev K_3. Dominantno število grafa je minimalna moč dominantne množice. Množica vozlišč $D \subseteq V(G)$ je dominantna, če za vsako vozlišče $v \in V(G)$ velja $v \in D$ ali pa ima v soseda v D ($uv \in E(G)$ in $u \in D$).

(b) Ena od smiselnih izbir je naslednja:

DOPUSTNE REŠITVE: Za dopustne rešitve lahko vzamemo vse postavitve trdnjav na šahovnico.

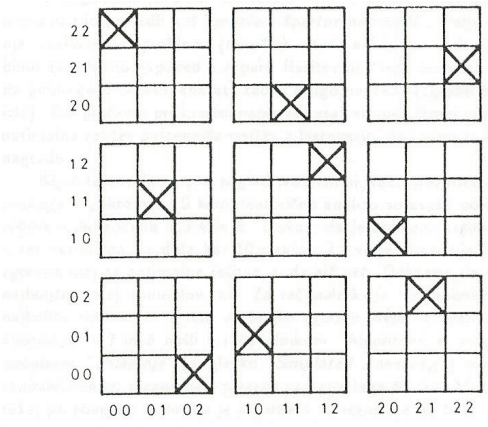
DEFINICIJA SOSEDOV: Premik ene trdnjave (odstranitev, dodajanje ali oboje).

NAMENSKA FUNKCIJA: Stroškovna funkcija rešitve naj bo seštevek števila uporabljenih trdnjav in števila nenapadenih polj.

Tu smo se odločili, da so dopustne rešitve vse možne postavitve trdnjav. Vsako postavitev trdnjav na šahovnico, pri kateri so vsa polja napadena, lahko imenujemo *približna rešitev* problema. Rešitev, pri kateri je uporabljeno minimalno število trdnjav, pa je seveda optimalna rešitev.

Opomba. Na temo problema trdnjav je bilo objavljeno precej člankov. Pogosto so najboljše meje dosegli z uporabo hevristik, na primer z ohlajanjem in s tabu iskanjem.[9]

(c) Rešitev za $n = 4$.



Število trdnjav v optimalni rešitvi za vrednosti $n = 2, 3, 4, 5$ je v spodnji tabeli. Kot zanimivost omenimo, da je dokaz, da je dobljena rešitev za dimenzijo 5 res optimalna, dolg kar 10 strani.[10] Za $n > 5$ optimalnih rešitev ne poznamo, znane so samo zgornje in spodnje meje.

n	2	3	4	5
število trdnjav	3	5	9	27

[9] [Laarhoven, P.J.M., Aarts, E.H.L., Lint, J.H. and Wille, L.T. : New Upper Bounds for the Footbal Pool Problem for 6,7 and 8 Matches, Journal of Combinatorial Theory A 52 (1989) 304-312], [R. Davies, G. F. Royle, Graph domination, tabu search and the football pool problem, Discrete Appl. Math. 74 (1997) 217-228].

[10] [H.J.L.Kamps, J.H.Lint, The Footbal Pool Problem for 5 Matches Journal of Combinatorial Theory 3 (1967) 315–325.]