Univerza v Mariboru

Fakulteta za logistiko

Visokošolski učbenik

PRINCIPI MODELIRANJA V LOGISTIKI

Dejan Dragan

Celje, 2010

Naslov:
Principi modeliranja v logistiki

Izdajatelj:
Fakulteta za logistiko Univerza v Mariboru

Avtor:
dr. Dejan Dragan

Pomoč pri urejanju:

Tea Vizinger

Recenzenta:

Doc.dr. Damir Vrančić
Doc. dr. Đani Juričić

KAZALO

Spremna beseda

To delo je zasnovano kot učbenik v pripravi, ki obravnava snov predmeta »Principi modeliranja v logistiki«, ki se predava na podiplomskem študiju Fakultete za logistiko.

Delo se osredotoča na področje reševanja logističnih in transportnih problemov in njihove optimizacije s pomočjo teorije grafov, ter pokriva kratke teoretične osnove z dodanimi rešenimi primeri. Mišljeno je kot začasno gradivo, ki naj bi ga študentje uporabljali poleg ostalih gradiv (prosojnic, itn) pri študiju tega predmeta, dokler ne bo v celoti izdelan vsestransko obsegajoč in še nekoliko bolj razširjen učbenik iz tega predmeta.

Pri delu s študenti je bilo ugotovljeno, da potrebujejo veliko računskih vaj pri osvajanju obsežnega repertoarja snovi v okviru kvantitativnih predmetov. To pomeni, da morajo preizkusiti metode za reševanje problemov na kar največjem naboru nalog, da so res učinkovito kos izpitnim vprašanjem in sposobni kvalitetnega razumevanja snovi.

V okviru tega se pričakuje vsaj osnovno predznanje študentov s področja visokošolske matematike, ponekod pa tudi vsaj osnovno razumevanje iz osnov programiranja.

V gradivu je takorekoč vsaka naloga opremljena ne le s končno rešitvijo, pač pa tudi s celotnim postopkom reševanja. Pri tem je nazorno prikazan prav vsak korak računanja, z jasnim ciljem, da lahko sleherni študent ujame ritem razlage in slednjemu brez težav sledi do konca izračunov.

To delo seveda ni v celoti originalno, pač pa se opira na številne učbenike in druga gradiva. Večina slednjih je navedena v seznamu literature, zato na tem mestu naštejmo le tista gradiva, na katera smo se najbolj opirali pri tvorbi tega dela: Žerovnik: Osnove teorije grafov in diskretne optimizacije, Wilson in Watkins: Uvod v teorijo grafov (slovenski prevod), Aldous in Wilson: Graphs and Applications, An introductory Approach, ter Teodorović: Transportne mreže, algoritamski pristup.

Potrebno je tudi poudariti, da je to delo šele prva verzija učbenika v pripravi, zato ni izključena možnost določenih tiskarskih in podobnih napak. Za morebitne napake se bralcu že vnaprej opravičujem in bom hvaležen za vsak kritičen komentar. Prav tako se bralcu opravičujem za morebitno ne najboljšo vidljivost določenik slik. Tudi te bodo izdelane v boljši resoluciji v naslednjih verzijah učbenika.

Nazadnje bi se želel zahvaliti vsem, ki so kakorkoli sodelovali pri izdelavi tega dela, še posebej študentom, ki so mi v okviru nekaterih seminarskih nalog pomagali izdelati določene primere tega dela.

Avtor

1 UVOD

Glavno področje obravnave tega dela je takoimenovana teorija grafov. Gre za eno najhitreje razvijajočih se področij sodobne matematike, verjetno predvsem zaradi uporabnosti v mnogih znanostih in sodobnih tehnologijah. Teorija grafov je področje diskretne matematike, ki se ukvarja z grafi, strukturami, sestavljenimi iz točk in medsebojnih povezav [16].

Za uradni začetek teorije grafov se pogosto vzameta objavi člankov Leonharda Eulerja o sedmih mostovih v Konigsbergu leta 1736 in Vandermondeov članek o šahovskem problemu s konjem. Grafi so se izkazali kot odlično orodje za modeliranje mnogih problemov, ki jih srečamo v vsakdanjem in poklicnem življenju. Tako s teorijo grafov rešujemo probleme, kot so iskanje učinkovitih algoritmov v računalništvu, dodeljevanje frekvenc oddajnikom v brezžičnih omrežjih, reševanje različnih diskretnih problemov, podobnih Eulerjevim mostovom, itn.

Poglejmo si še nekaj tipičnih primerov problemov, ki jih lahko rešimo s pomočjo teorije grafov [16]:

- Denimo smo se znašli v sredini labirinta. Ali obstaja zanesljiva metoda, ki nam pomaga poiskati pot ven?

- Kako bi poiskali najkrajšo pot, če bi se želeli peljati iz Kopra v Mursko Soboto?

- Če bi poskusili pobarvati zemljevid Združenih Držav Amerike tako, da bi sosednji državi vedno pobarvali z različnima barvama, bi ugotovili, da potrebujemo samo štiri barve. Ali to velja za vse zemljevide, ali pa obstajajo zemljevidi, za katere je potrebno več barv?

Čeprav so ti problemi na prvi pogled precej različni, lahko vse opišemo kot takšne, pri katerih imamo opravka z nekimi objekti in povezavami med njimi. Zato jih lahko

rešujemo s teorijo grafov, ki se izkaže kot izredno učinkovita pri reševanju tovrstnih nalog.

Mnogo takšnih in podobnih nalog ima korenine v pomembnih praktičnih problemih iz znanosti in tehnologije. Pri tem velja, da je dobila teorija grafov v zadnjih letih precejšnjo vzpodbudo prav iz potrebe po reševanju konkretnih nalog v industriji [16]. Poleg tega je mogoče pri reševanju problemov s področja analize omrežij in operacijskih raziskav z uporabo tehnik s področja teorije grafov doseči znatne prihranke v času in denarju [16].

Pomembno pa je, da znamo takšne naloge predstaviti v jeziku teorije grafov, kar je možno storiti s pomočjo matematičnega modeliranja. V procesu modeliranja potrebujemo formulacijo naloge v takšni obliki, da jo lahko obravnavamo s tehnikami teorije grafov. Seveda to ni vedno enostavno opravilo. Način, s pomočjo katerega je izvedeno modeliranje, ter stopnja natančnosti opisa originalnega problema s pomočjo matematičnega modela, se lahko pomembno razlikujeta od problema do problema.

Potemtakem so glavne teme tega dela predstavitev teorije grafov, modeliranje problemov, ki jih rešujemo s pomočjo te teorije, ter ilustracija postopkov reševanja problemov s poudarkom na logističnih primerih.

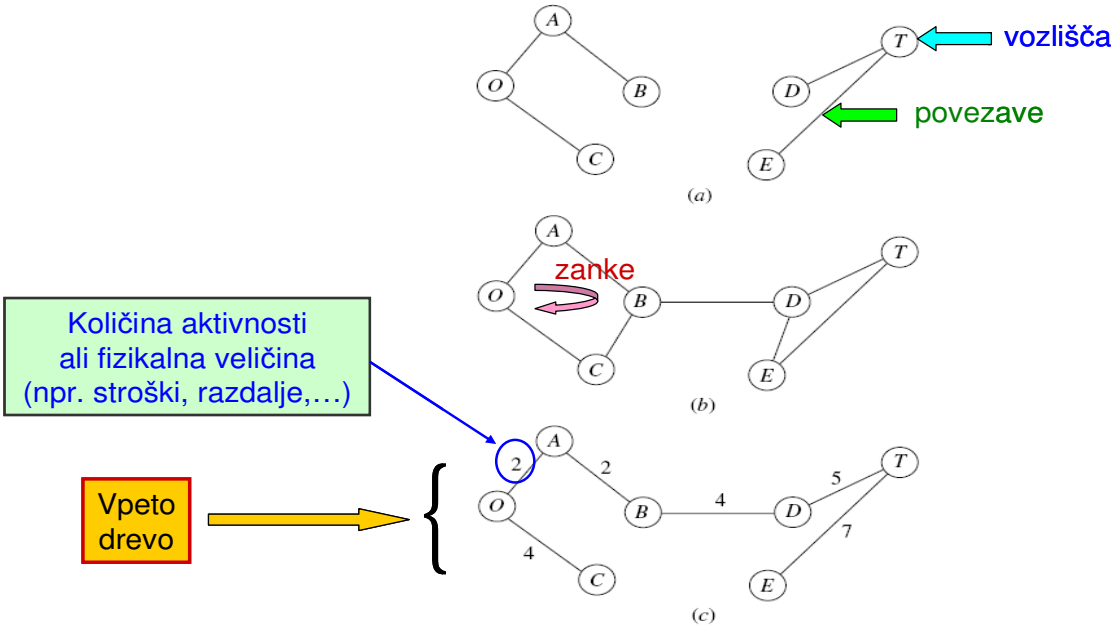
Pri tovrstnih postopkih gre predvsem za takoimenovano optimizacijo mrežnih struktur, grafe namreč pogosto poimenujemo tudi mreže. Mreže igrajo pomembno vlogo tudi na področju transporta, komunikacij, produkcije, distribucije, projektnega planiranja, itn. Z njimi lahko na eleganten način vizuelno in konceptualno ilustriramo in reprezentiramo medsebojne odnose in povezave, ki vladajo med posameznimi komponentami nekega sistema.

Poglejmo si nekaj osnovnih definicij, ki izhajajo iz topologije mrež in grafov [6]:

- Vozlišča,
- Povezave,
- Mreža ali graf (skupek vozlišč in povezav),
- Drevo (povezan podgraf brez ciklov (zank), ki vsebuje nekatera izmed vozlišč grafa).
- Vpeto drevo (drevo, ki vsebuje vsa vozlišča grafa).

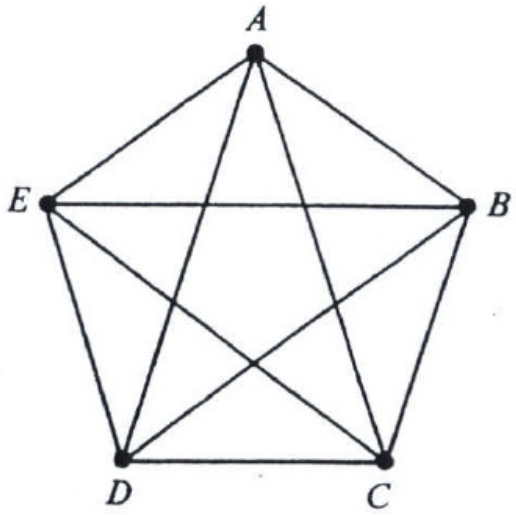
- Minimalno vpeto drevo (vpeto drevo, pri katerem je skupna dolžina povezav med posameznimi pari vozlišč najmanjša).

Slika 1 ilustrira nekaj primerov mrež oz. grafov, kjer so npr. razvidna vozlišča, povezave, zanke in vpeto drevo.



Slika 1.: Ilustriracija primerov mrež oz. grafov, kjer so razvidna vozlišča, povezave, zanke in vpeto drevo.

Pri grafih imamo torej opravka z nekim diagramom točk, ki so med seboj povezane s črtami (slika 2). Povezave med točkami lahko pomenijo tudi vezi med atomi v kemijski molekuli, žice med komponentami v električnem vezju, ceste med mesti na zemljevidu in podobno. Poznamo tudi takoimenovane usmerjene grafe ali digrafe (directed graphs), ki jih definiramo podobno kot grafe, le da ima vsaka povezava usmeritev, ki jo prikažemo s puščico. Zančilen primer takšnih grafov je npr. predstavitev cestnega omrežja z enosmernimi cestami.



Slika 2.: Primer osnovnega grafa

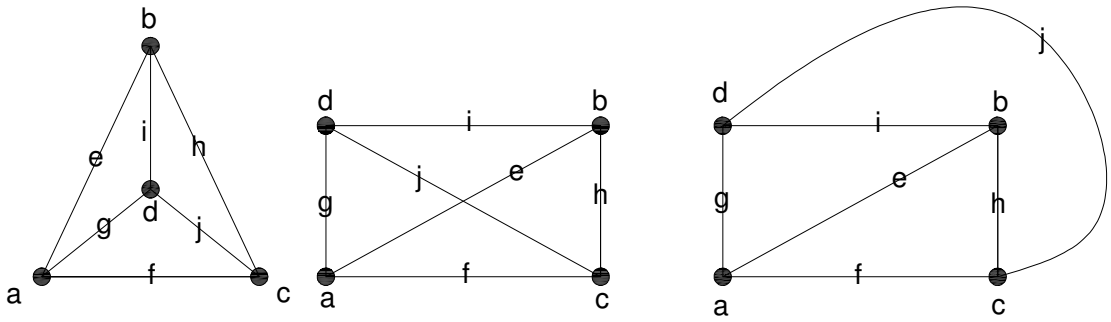
Učbenik daje največji poudarek naslednjim področjem: Definicije in primeri s področja teorije grafov, opis značilnih grafov in dreves, postopki optimizacije na drevesih, algoritmi s področja teorije grafov, reševanje značilnih problemov, kot npr. problemov najkraše poti, maksimalnega pretoka in minimalno vpetega drevesa, itn.

V naslednjem poglavju si bomo najprej pogledali nekaj osnovnih definicij in primerov s področja teorije grafov, ter podali osnoven opis značilnosti grafov.

2 DEFINICIJE IN OSNOVNI POJMI IZ TEORIJE GRAFOV

Graf G = (V(G), E(G)) je določen z množico vozlišč in seznamom povezav [16, 19]. 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).

Povezavi grafa priredimo vozlišče začetek (prvo krajišče) in vozlišče konec (drugo krajišče) povezave. Torej 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. Na sliki 3 so narisane 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}.

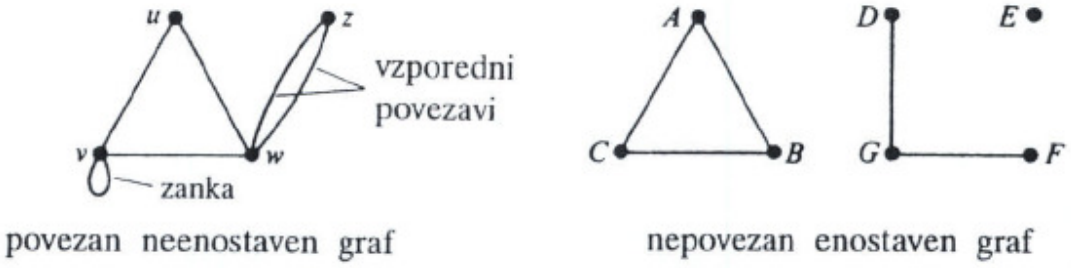


Slika 3: Tri risbe istega grafa

Posebno za majhne grafe je risba grafa zelo uporabna. Kadar opazujemo strukturo grafa, pogosto oznake vozlišč niso pomembne, zato v takšnem primeru graf lahko narišemo brez oznak.

V nekaterih aplikacijah je pri povezavi pomembno, kako je usmerjena. Takrat si povezavo predstavljamo kot usmerjeno povezavo ali puščico. Grafe z usmerjenimi povezavami imenujemo **usmerjeni grafi ali digrafi (directed graphs)**. Če nas usmerjenost povezav ne zanima, pa govorimo **o neusmerjenih grafih**.

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.** Primer zanke in vzporedne povezave je podan na sliki 4 [16].



Slika 4: Primer povezanega neenostavnega in nepovezanega enostavnega grafa

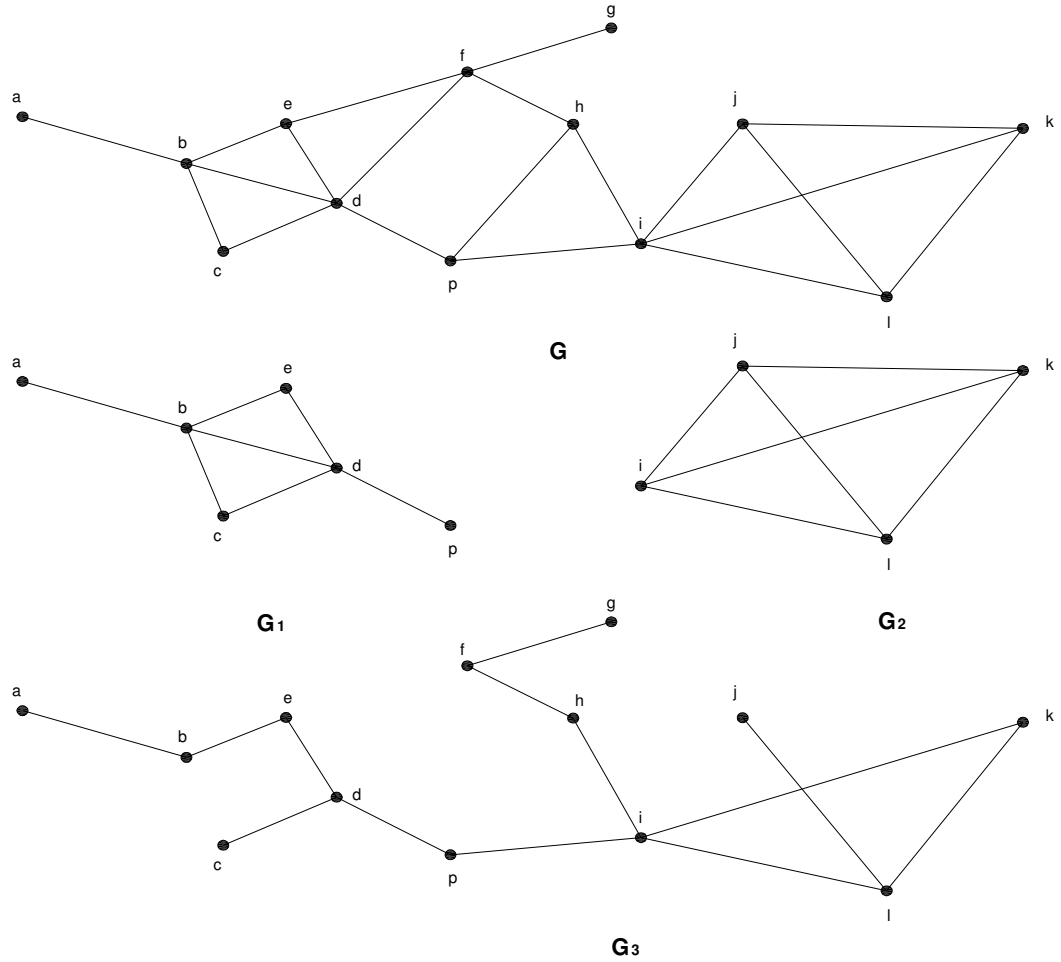
Če sta vozlišči grafa v in w medsebojno povezani s povezavo, potem rečemo, da sta **sosednji vozlišči**. Če gre povezava e = uv skozi vozlišči u in v, potem je **incidentna** (sosednja) z vozliščema u in v. Po drugi strani sta tedaj krajišči povezave e, vozlišči u in v, **incidentn**i s povezavo e (ležita na povezavi e) [16, 19].

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

Utežen graf je tisti graf G = (V(G),E(G)), ki vsaki povezavi grafa priredi **utež**. Utežem na povezavah pogosto lahko rečemo tudi dolžine ali cene povezav. V nekaterih aplikacijah lahko utežimo tudi vozlišča grafa, ko vsakemu vozlišču priredimo utež. Seveda lahko hkrati definiramo uteži na povezavah in na vozliščih [16 ,19].

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. Tudi pri teoriji grafov je tako, kjer lahko grafu tvorimo njegove **podgrafe**, primer je prikazan na sliki 5.

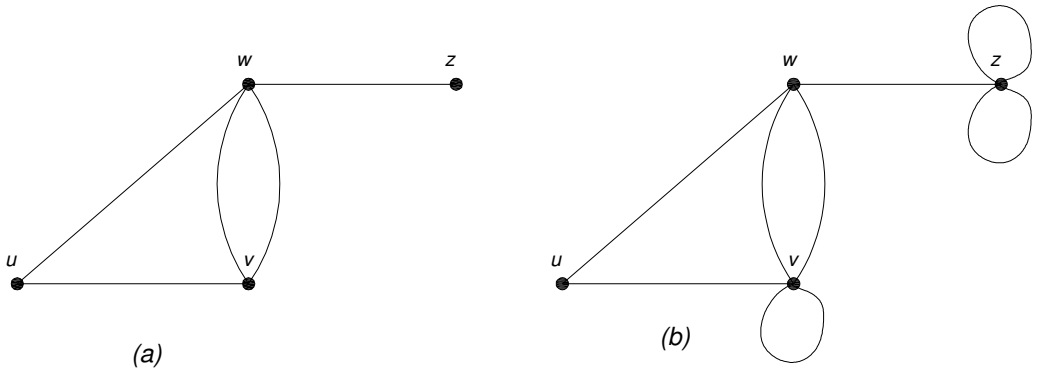
Na sliki 5 je G1 podgraf grafa G, induciran na vozliščih {a,b,c,d,e,p}, graf G2 je podgraf, induciran na povezavah {ij,ik,il,jk,jl,kl}, G3 pa je **vpet podgraf** grafa G [16, 19].



Slika 5: Graf G s podgrafi G1, G2 in G3

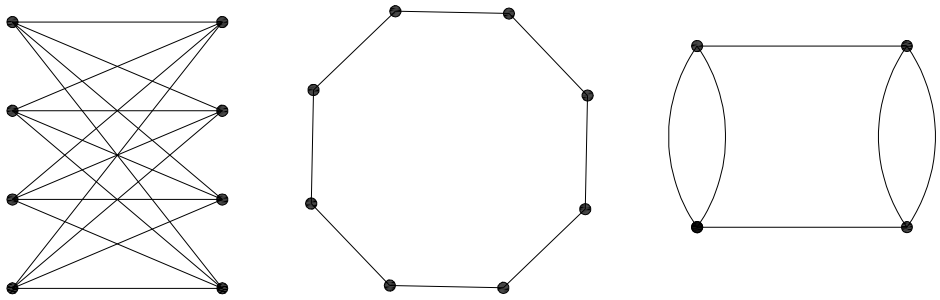
Naj bo G graf brez zank, v pa vozlišče grafa G. Tedaj velja, da je **stopnja vozlišča** v število povezav, ki vsebujejo v [16, 19]. Stopnjo vozlišča označimo s **st(v). Največjo stopnjo vozlišča** grafa G označimo z **Δ(G),** najmanjšo pa z **δ(G).** Za vsako vozlišče v grafa G torej velja: **δ(G) ≤ st(v) ≤ Δ(G). Soseščina vozlišča v,** $N_G(v)$, pa je množica vseh vozlišč, ki so sosednja z vozliščem v. Za graf a) na sliki 6 velja, da ima stopnje vozlišč: st(u) = 2, st(v) = 3, st(w) = 4, st(z) = 1 (zaporedje stopenj je (1,2,3,4)).

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 vrednost 2 k stopnji vozlišča [16, 19]. Tako ima graf b) na sliki 6 stopnje vozlišč: st(u) = 2, st(v) = 5. st(w) = 4, st(z) = 5 (zaporedje stopenj je (2,4,5,5)).



Slika 6: Ilustracija stopenj vozlišč v grafu

Graf je **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 sliki 7 je nekaj primerov regularnih grafov različnih stopenj [16].



Slika 7: Regularni grafi

<u>Lema o rokovanju:</u>

Prvi graf na sliki 7 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 je vsota stopenj vozlišč enaka natanko dvakratnemu številu povezav. Izkaže se, da to velja za vse grafe in rezultat imenujemo lema o rokovanju*** [16, 19].

Lema o rokovanju velja tudi za grafe z zankami, saj tudi vsaka zanka prispeva k stopnji pripadajočega vozlišča natanko vrednost 2. Poimenovanje lema o rokovanju izvira iz dejstva, da lahko z grafom predstavimo skupino ljudi, ki se rokuje na zabavi. V takšnem grafu so ljudje predstavljeni z vozlišči, povezavo med posameznima 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 sodelujeta natanko dve roki [19].

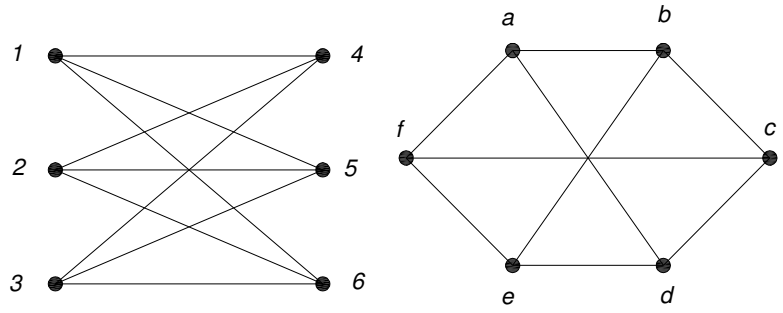
Iz leme o rokovanju lahko sklepamo naslednja dejstva [19]:

- V vsakem grafu je vsota vseh stopenj vozlišč grafa sodo število.
- V vsakem grafu, ki ima liho število vozlišč, je vsota stopenj vozlišč sodo število.
- Če ima graf G n vozlišč in je regularen stopnje r, ima natanko $\frac{n \cdot r}{2}$ povezav.

Izomorfizem grafov

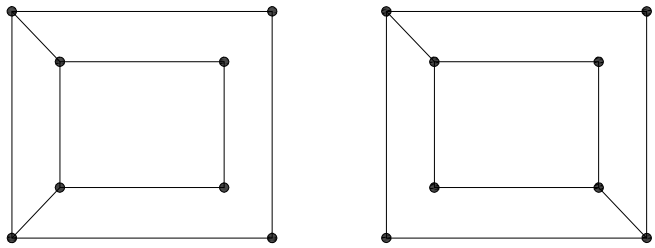
Dva grafa G in H sta izomorfna, če lahko graf H dobimo iz grafa G tako, da spremenimo oznake vozlišč [16, 19]. To je možno v primeru, če obstaja takšna povratno enolična preslikava med vozlišči G in vozlišči H, da je število povezav, ki povezujejo katerikoli par vozlišč v G, enako številu povezav, ki povezujejo pripadajoči par vozlišč v H [16, 19].

Grafa na sliki 8 sta izomorfna, ker imamo opravka s preslikavo, definirano z: 1→ a, 2 → c, 3 → e, 4 →b, 5 → d, 6 → f. Ker je v tem primeru vozlišč malo, lahko to enostavno preverimo s pregledom vseh parov vozlišč in njihovih slik.



Slika 8: Izomorfna grafa

Grafa na sliki 9 pa nista izomorfna. V prvem grafu imamo namreč dve sosednji vozlišči stopnje 2, v drugem grafu pa takega para vozlišč ni. Izomorfizem torej ohranja stopnjo vozlišča, pri čemer se sosednja vozlišča preslikajo v sosednja vozlišča [16, 19].

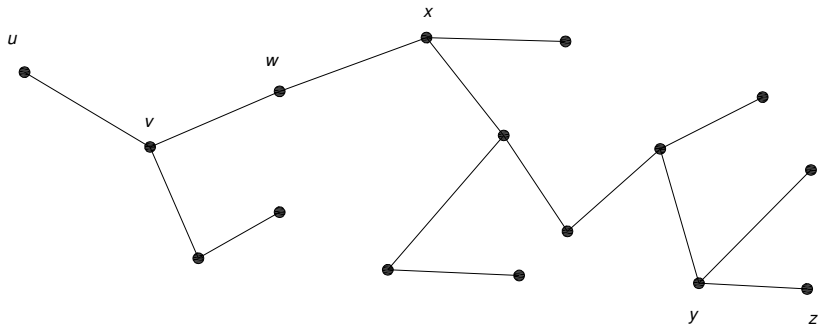


Slika 9: Neizomorfna grafa

Invarianta grafa ali stalnica je lastnost grafa, ki se ohranja pri izomorfizmu [19]. Invariante grafa so: število vozlišč, število povezav, zaporedje stopenj.

Sprehodi po grafih

V definiciji sprehoda je ,,drugo vozlišče" povezave vedno isto kot ,,prvo vozlišče" naslednje povezave. Intuitivno si sprehod na sliki 10 lahko predstavljamo kot sprehod od vozlišča u do vozlišča v, potem do w, potem do x, in tako naprej, dokler nazadnje ne končamo v vozlišču z.

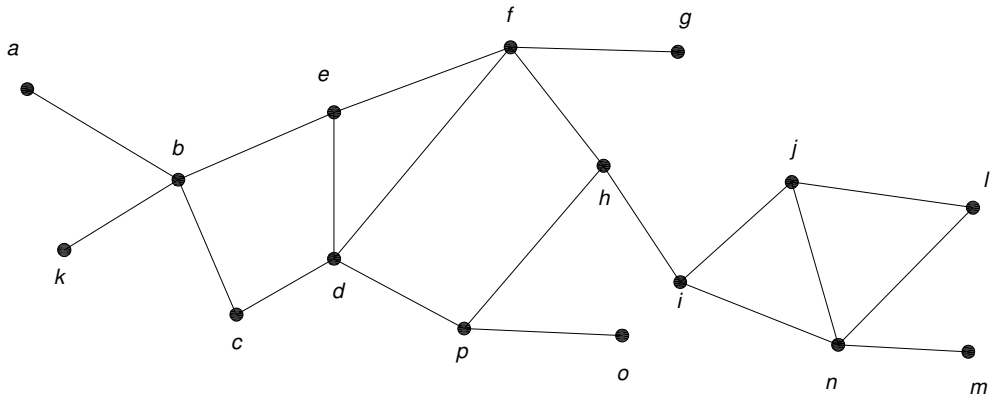


Slika 10: Sprehodi na grafu

Ker povezave niso usmerjene, lahko takšen sprehod razumemo tudi kot potovanje od z nazaj proti y, in tako dalje do vozlišč x, w, v, ter nazadnje do u. V definiciji sprehoda ni zahteve, 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** [16, 19].

Posebej poimenujemo tiste sprehode in poti, ki se začnejo in končajo v istem (izhodiščnem) vozlišču. Tedaj imamo opravka s **sklenjenim sprehodom ali obhodom** [19]. Če so vse povezave obhoda različne, potem ga poimenujemo **enostaven obhod ali sklenjena sled**. Če pa so v obhodu vse povezave in vsa vozlišča (razen izhodiščnega) različna, potem ga imenujemo **cikel** [16, 19].

V grafu na sliki 11 je obhod b-c-d-p-h-f-d-e-b sklenjena sled, ki ni cikel, saj se vozlišče d pojavi dvakrat. Po drugi strani pa so sklenjene sledi b-c-d-p-h-f-e-b, i-j-l-n-i in b-c-d-e-b tudi cikli. Cikle dolžine tri, kot sta na primer d-e-f-d ali n-i-j-n, imenujemo **trikotnik**i [16, 19]. Pri opisovanju sklenjenih sprehodov lahko uporabimo za začetek katerokoli vozlišče sprehoda. Prav tako običajno velja dogovor da se izhodiščno vozlišče ne zapiše dvakrat. Tako na primer trikotnik d-e-f-d lahko enakovredno zapišemo z d-e-f, e-f-d ali f-d-e.



Slika 11: Sprehodi na grafu in pojem ciklov

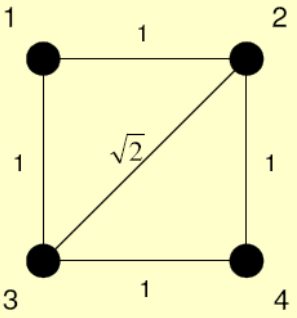
Povezanost grafov in razdalje

Graf G je povezan, če obstaja pot med poljubnim parom vozlišč, sicer pa je nepovezan (glej sliko 4) [19]. Nepovezan graf razpade na nekaj povezanih podgrafov, ki jih imenujemo (povezane) komponente grafa. **Most** je povezava v povezanem grafu, brez katere bi bil graf nepovezan [19].

Razdalja $d_G(u,v)$ med vozliščema u in v grafu G je dolžina najkrajšega sprehoda 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, v grafu G, velja: $d_G(u,v)<\infty$ [19].

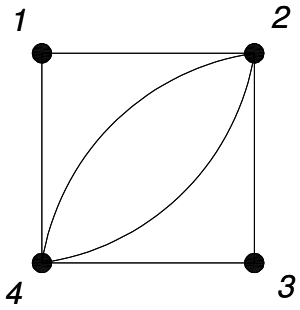
Matriko D(G) velikosti n x n, v kateri element v i-ti vrstici in j-tem stolpcu pove razdaljo med vozliščema i in j v grafu G, imenujemo matrika razdalj grafa G (glej sliko 12 a)). Opazimo lahko, da so diagonalni elementi v matriki razdalj enaki 0. V neusmerjenem grafu je matrika razdalj simetrična [19].

a)



$$D(G)=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & \sqrt{2} & 1 \\ 1 & \sqrt{2} & 0 & 1 \\ 2 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

b)

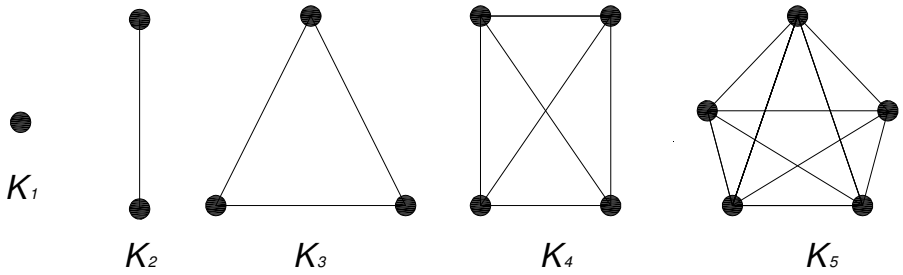


$$D(G)=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix} \end{array}$$

Slika 12: Graf G. a) matrika razdalj D(G), b) matrika sosednosti D(G)

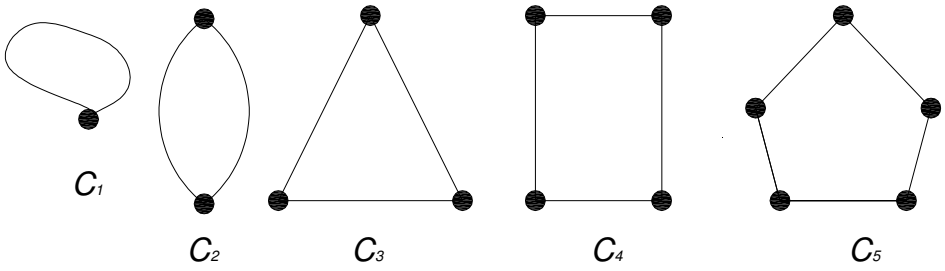
Primeri grafov

Poln graf je graf, v katerem je vsak par različnih vozlišč povezan z natanko eno povezavo [16, 19]. Poln graf na n vozliščih označimo s *Kn* (Slika 13). Graf *Kn* je regularen stopnje n-1, zato ima po lemi o rokovanju $\frac{n\cdot(n-1)}{2}$ povezav [19].



Slika 13: Primeri polnih grafov

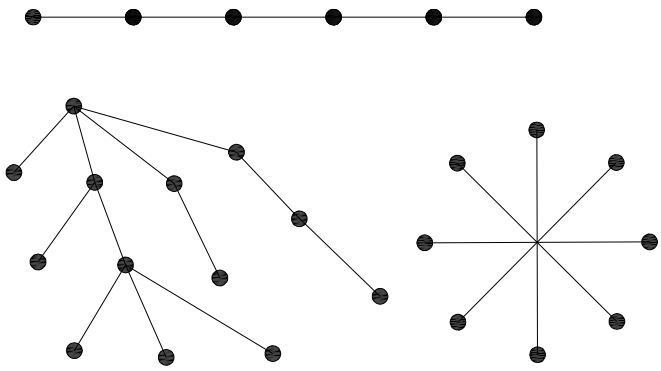
Cikel je graf, ki ga sestavlja en sam cikel. Graf cikla označimo s *Cn*. Graf *Cn* je regularen stopnje 2 in ima n povezav [16, 19]. Slika 14 prikazuje nekatere primere ciklov.



Slika 14: Primeri ciklov

Pot je graf, ki ga sestavlja ena sama pot. Pot na n vozliščih označimo s *Pn*. Graf *Pn* ima n-1 povezav in ga lahko dobimo iz cikla *Cn* z odstranitvijo katerekoli povezave [16, 19].

Drevo je povezan graf brez ciklov [16, 19]. Nekaj primerov dreves je narisanih na sliki 15. Naj bo G poljuben povezan graf. Podgraf grafa G, ki vsebuje vsa vozlišča grafa G in je drevo, imenujemo vpeto drevo grafa G [19].



Slika 15: Primeri dreves

Predstavitve grafov

Grafična predstavitev grafov 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.

Obstajajo pa tudi načini, kjer se za upodobitev grafov uporablja matrike. 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. V nadaljevanju so prikazane najpomembnejše vrste: matrika sosednosti in incidenčna matrika. Pri prvem tipu matrike zapišemo tabelo, v kateri označimo, kateri pari vozlišč so povezani. Pri drugem tipu matrike pa zapišemo tabelo, v kateri označimo, katera vozlišča so incidentna s katerimi povezavami [16, 19].

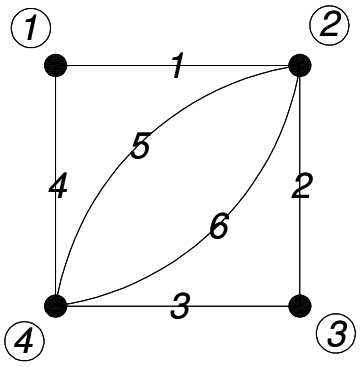
Matrika sosednosti

Matriko sosednosti lahko razložimo na primeru, ki je ilustriran na sliki 12 b).

Na levi strani slike 12 b) imamo graf s štirimi vozlišči, na desni strani pa imamo matriko reda 4x4. 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 število 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 število 2. Vozlišči 1 in 3 nista povezani, zato je v tretjem stolpcu prve vrstice in v prvem stolpcu tretje vrstice število 0. Podobno lahko sklepamo za preostale elemente matrike sosednosti. Vidimo lahko tudi, da je matrika simetrična glede na glavno diagonalo [16, 19].

Incidenčna matrika

V matriko sosednosti zapišemo podatke o sosednosti vozlišč. V incidenčno matriko pa zapišemo informacije, katere vozlišča in povezave so incidentne (sosednje) [16, 19]. Poglejmo naslednji primer, ki je prikazan na sliki 16.



$$I(G) = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \\ \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$$

Slika 16: Graf in incidenčna matrika I(G)

Na levi strani slike 16 imamo graf s štirimi vozlišči in šestimi povezavami, na desni strani pa matriko reda 4 x 6. Elementi matrike so 1 ali 0, odvisno od tega, ali sta ustrezno vozlišče in povezava incidentni (sosednji) ali ne (vozlišče in povezava sta incidentni, če je vozlišče krajišče povezave) [16, 19].

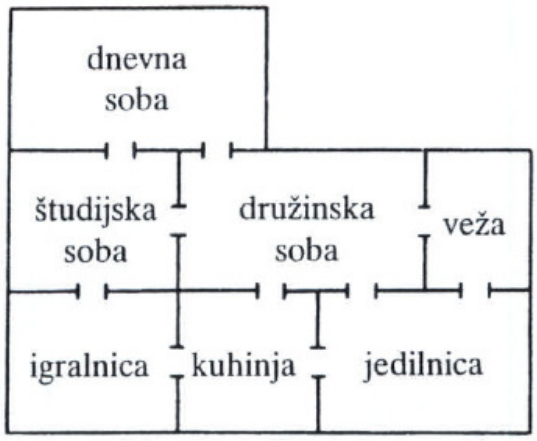
3 REALNI PRIMERI APLIKACIJ IZ TEORIJE GRAFOV

Na kratko bomo predstavili naslednje realne primere aplikacij, kjer lahko učinkovito apliciramo teorijo grafov:

- Tlorisi v gradbeništvu,
- Električna vezja,
- Napeljave instalacij v hiše,
- Problem najhitrejše poti med dvema mestoma, ter
- Problem enkratnega prehoda sedmih mostov.

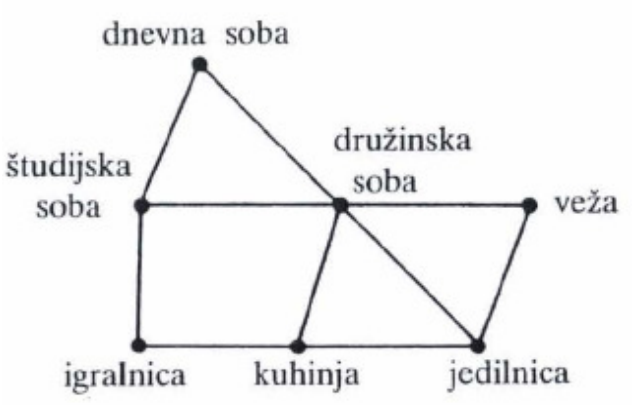
Tlorisi v gradbeništvu

Na sliki 17 je načrt spodnjega nadstropja neke hiše [16].



Slika 17: Načrt spodnjega nadstropja neke hiše

Za tako majhne načrte z diagramom na sliki 17 zelo nazorno prikažemo, iz katere sobe lahko pridemo v katero drugo sobo. Če je sob več, pa je primernejša predstavitev z grafom, kjer sobe zamenjamo z majhnimi črnimi pikami (vozlišči), kar je prikazano na sliki 18 [16].



Slika 18: Graf spodnjega nadstropja neke hiše

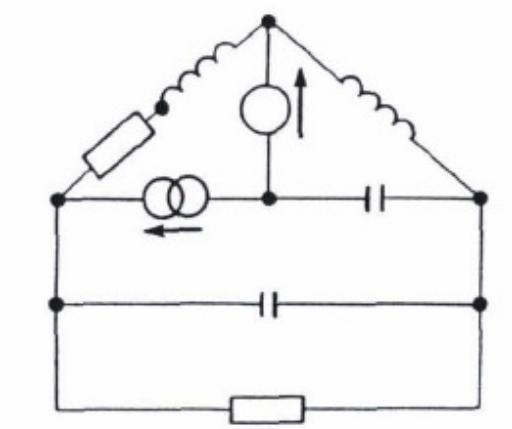
Iz slike 18 je razvidno tudi, da smo prehode skozi vrata na sliki 17 nadomestili s povezavami.

Arhitekti takšne grafe imenujejo tudi diagrami pretoka, ker so uporabni pri analizi gibanja ljudi v večjih stavbah. Posebej uporabni so pri načtovanju letališč in veleblagovnic. Seveda so takšni diagrami uporabni za predstavitev povezav med sobami, ne dajejo pa nobene informacije o velikosti in obliki sob [16].

Električna vezja

Diagram (graf) na sliki 19 ponazarja električno vezje z dvema uporoma, dvema kondenzatorjema, dvema tuljavama, ter generatorjema toka in napetosti.

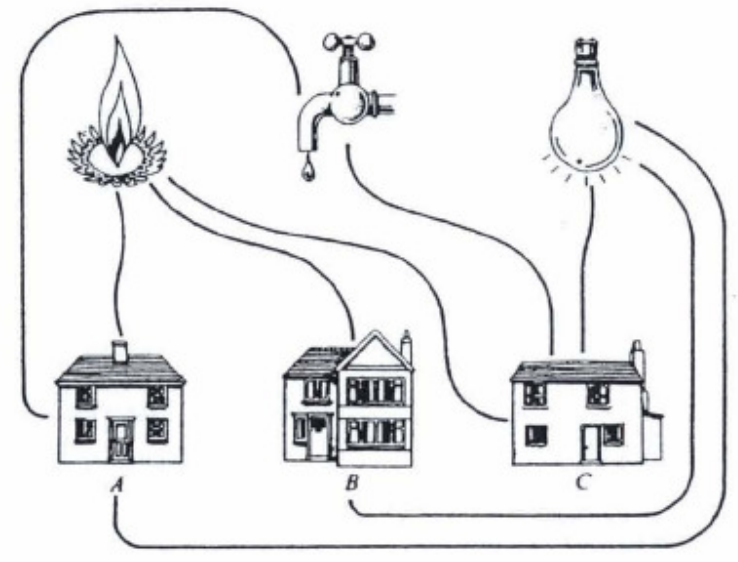
Diagrami te vrste so zelo uporabni za prikaz povezav med posameznimi deli vezja. Vendar pa nam ne dajo nikakršne informacije o geometriji vezja. Tako npr. nič ne vemo o dolžini in debelini žic ter njihovi legi [16].



Slika 19: Graf za primer električnega vezja

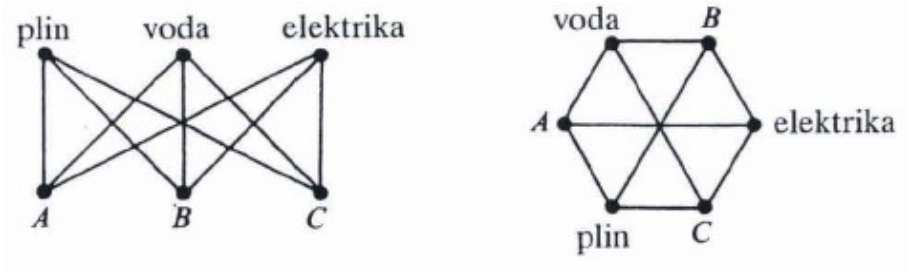
Napeljave instalacij v hiše

Na sliki 20 je ilustriran problem napeljave instalacij v hiše. Pri tej nalogi želirno povezati tri hiše, A, B in C, s tremi napeljavami: plinovodom, vodovodom in električno napeljavo. Zaradi varnosti zahtevamo, da se napeljave ne smejo križati. Zanima nas torej, če lahko naredimo vse povezave, ne da bi se medseboj križale. Kot je razvidno iz slike 20, lahko brez problemov potegnemo osem od devetih povezav. Dilema pa nastopi pri deveti povezavi [16, 19].



Slika 20: Instalacije za tri hiše

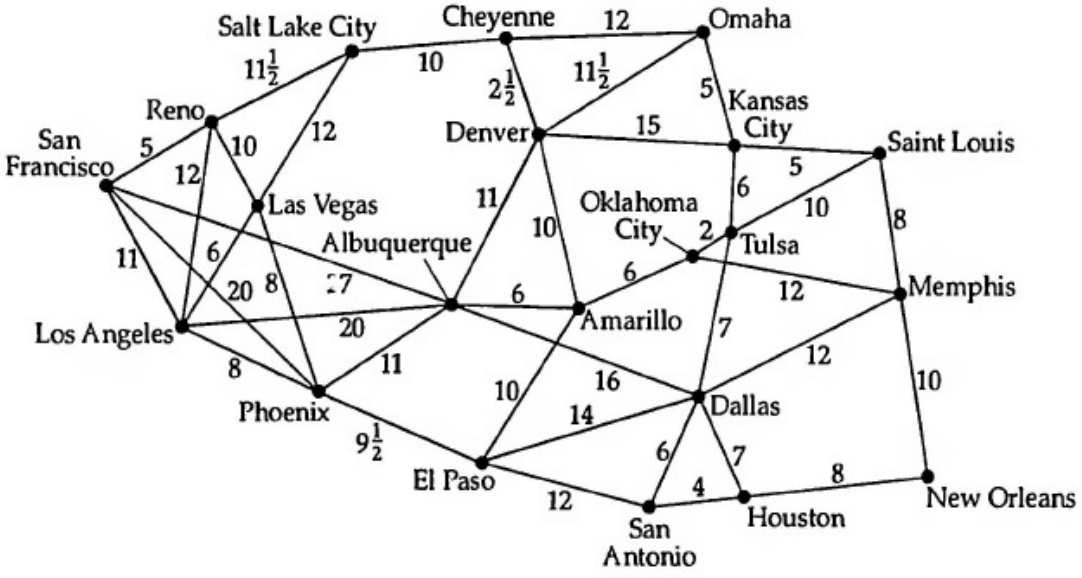
Nalogo lahko predstavimo z levim grafom na sliki 21 [16, 19]. Za točke vzamemo tri hiše in tri izvore napeljav. Graf ima torej šest vozlišč in devet povezav. Isto nalogo lahko predstavimo tudi z desnim grafom na sliki 21. Oba grafa namreč sporočata isto, da so tri hiše povezane z vsako od treh napeljav. Naloga napeljav pa je ugotoviti, ali se da narisati takšen graf, ki bo enak grafoma na sliki 21, vendar ne bo prišlo do sekanja povezav.



Slika 21: Grafa za instalacije treh hiš

Problem najhitrejše poti med dvema mestoma

Ko uporabljamo cestni zemljevid, niso pomembne le povezave med posameznimi mesti, pač pa tudi pripadajoče razdalje oz. časi potovanja med mesti. Slika 22 prikazuje primer cestnega zemljevida za ZDA, kjer so označene glavne poti oz. povezave med mesti [16]. Kot je razvidno iz slike 22, so povezave tudi utežene s časi trajanja posameznih voženj med pari sosednjih mest.

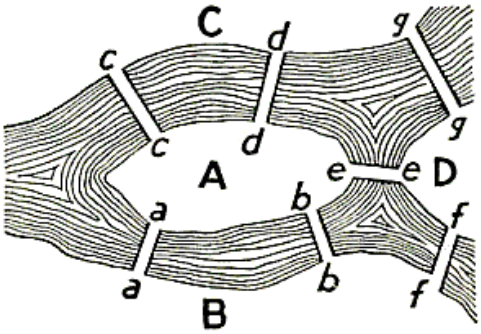


Slika 22: Cestni zemljevid ZDA

Seveda želimo med dvema mestoma potovati po čim krajši poti čim manj časa. Problem je torej, kako npr. poiskati najkrajšo (najhitrejšo) pot med mestoma Los Angeles in Amarillo, ali npr. med mestoma San Francisco in Denver. Kot se izkaže, je uporaba teorije grafov lahko silno uporaben pripomoček pri reševanju tovrstnih problemov.

Problem enkratnega prehoda sedmih mostov

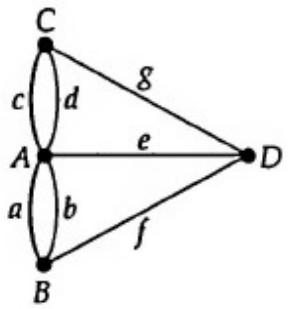
V zgodnjem 18. stoletju je srednjeveško mesto Konigsberg vsebovalo centralen otok Kneiphof, okoli katerega je tekla reka Pregel. Tovrstna kofiguracija je mesto razdelila na 4 dele A, B, C in D, ki so bili povezani s sedmimi mostovi (a, b, c, d, e, f, g), kar je ilustrirano na sliki 23 [16, 19].



Slika 23: Srednjeveško mesto Konigsberg

Kot je znano, so si tedaj meščani dali duška z vprašanjem, če obstaja takšna pot, ki bi prečkala vsak most natanko enkrat in se vrnila v izhodiščno pozicijo.

Tovrsten problem lahko predstavimo s pomočjo grafa, kjer posamezne dele mesta (A, B, C in D) nadomestimo z vozlišči, prehode čez mostove (a, b, c, d, e, f, g) pa s povezavami med vozlišči (slika 24) [16].



Slika 24: Graf za srednjeveško mesto Konigsberg

Prvotni problem torej lahko transformiramo v problem s področja teorije grafov, kjer si izberemo na sliki 24 izhodiščno vozlišče, ter nato poskušamo najti takšno pot, ki ne bo nobenega mostu prečkala dvakrat (ter se vrnila v izhodišče).

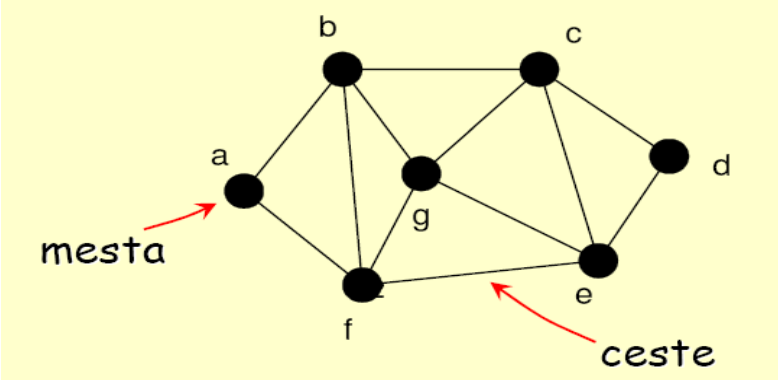
4 EULERJEVI IN HAMILTONOVI GRAFI

V tem poglavju bomo spoznali dva zelo pomembna tipa grafov: Eulerjevega in Hamiltonovega, poimenovana po matematikih Leonhardu Eulerju in Williamu Rowanu Hamiltonu [16, 19]. Dotična grafa sta namreč imela ključen pomen v razvoju teorije grafov, zato si ju velja pobliže pogledati.

Preden pa gremo na definicije obeh tipov grafov, si oglejmo še naslednji primer, kjer bo jasno razvidna razlika med enim in drugim tipom grafa.

Primer:

Denimo imamo opravka z naslednjim grafom, ki med seboj povezuje mesta z določenim naborom cest (slika 25) [2].



Slika 25: Graf, ki povezuje mesta s cestami

V povezavi z grafom na sliki 25 lahko definiramo naslednja dva problema [2]:

- **Problem trgovskega potnika**, kjer potnik želi najti takšno pot, ki obišče vsako mesto natanko enkrat in se vrne na izhodiščno lokacijo (npr. vozlišče a). Primera takšnih poti sta: a-b-c-d-e-g-f-a in a-f-e-d-c-g-b-a.

- **Problem kitajskega poštarja**, kjer želi poštar najti takšno pot, ki gre preko vseh cest natanko enkrat in se vrne na izhodiščno lokacijo (npr. vozlišče a). Primera takšnih poti sta: a-b-c-d-e-f-b-g-c-e-g-f-a in a-f-g-c-d-e-g-b-c-e-f-b-a.

Čeprav lahko poštar potuje po vsaki cesti le enkrat, pa načeloma lahko obišče določena mesta večkrat. Po drugi strani lahko potnik obišče vsako mesto le enkrat, vendar lahko načeloma potuje po določenih cestah večkrat [2, 16, 19].

Ključni problem pri trgovskem potniku je najti takšen cikel, ki vključuje vsa vozlišča danega grafa in vsebuje minimalno skupno razdaljo prepotovane poti [2]. Ključni problem pri kitajskem poštarju pa je najti takšen enostaven obhod, na katerem so vse povezave grafa in vsebuje minimalno skupno razdaljo prepotovane poti [2].

Z mislijo na pravkar postavljena problema tega primera bomo v nadaljevanju uvedli definiciji za Eulerjev in Hamiltonov graf [2, 16, 19].

Definicija za Eulerjev graf:

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

Definicija za Hamiltonov graf:

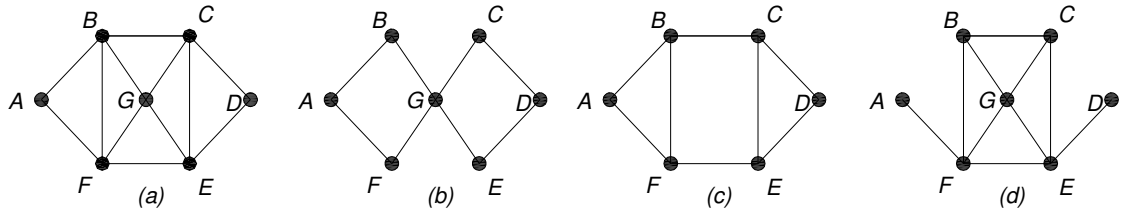
*Povezan graf je **Hamiltonov**, če obstaja cikel, na katerem so vsa vozlišča grafa. Tak cikel imenujemo **Hamiltonov cikel**.*

Če povežemo ti definiciji s prej podanim problemom, je očitno problem trgovskega potnika tesno povezan s problematiko Hamiltonovih grafov, problem kitajskega poštarja pa s problematiko Eulerjevih grafov [2].

V nadaljevanju si poglejmo primer, ki se tiče Eulerjevih in Hamiltonovih grafov.

Primer:

Na sliki 26 so podani štirje grafi. Ugotovite, katerega tipu grafov pripadajo.



Slika 26: Primer štirih grafov

Ugotovitve iz slike 26 so naslednje [16, 19]:

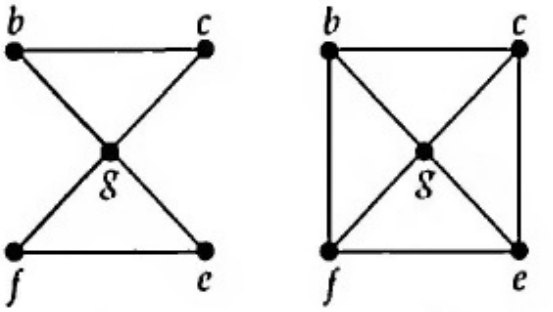
- Graf (a) je Eulerjev in Hamiltonov;
- Graf (b) je Eulerjev in ni Hamiltonov; Eulerjev obhod je *ABGCDEGF;*

- Graf (c) je Hamiltonov in ni Eulerjev; Hamiltonov cikel je *ABCDEF;*
- Graf (d) ni niti Eulerjev niti Hamiltonov.

Poglejmo si še en primer, kjer je en graf Eulerjev in ni Hamiltonov, drug graf pa je Hamiltonov in ni Eulerjev [2, 16, 19].

Primer:

Na sliki 27 sta podana dva grafa. Ugotovite, katerega tipu grafov pripadata.



Slika 27: Primer dveh grafov

Iz slike 27 je razvidno, da je levi graf Eulerjev (z obhodom b-c-g-f-e-g-b) in ni Hamiltonov, desni graf pa je Hamiltonov (s ciklom b-c-g-e-f-b) in ni Eulerjev.

4.1 Eulerjevi grafi

Če se vrnemo na problem enkratnega prehoda sedmih mostov iz Konigsberga (na sliki 23), smo videli, da ga lahko prevedemo v problem, kjer najprej tvorimo graf na sliki 24, nato pa poskušamo poiskati enostaven obhod, na katerem so vključene vse povezave grafa natanko enkrat. Vendar se izkaže, da takšen graf ne vsebuje nobenega Eulerjevega obhoda, zatorej problema Konigsberških mostov ni možno rešiti s tako zastavljeno strategijo [2, 16, 19].

Euler je obravnaval problem tudi na bolj splošnih razporeditvah delov mesta in mostov. Tako je našel splošno pravilo, kdaj je obhod iskanega tipa mogoče najti, torej kdaj je graf Eulerjev [2, 16, 19]. 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 vrednost 2 k stopnji tega vozlišča. (To velja tudi za prvo in zadnjo povezavo na obhodu, ki skupaj tudi prispevata vrednost 2 k stopnji začetnega vozlišča). **Odtod pa sledi, da mora imeti v Eulerjevem grafu vsako vozlišče sodo stopnjo** [16, 19].

Euler je opazil, da je ta pogoj tudi zadosten in (v drugačni terminologiji) dokazal naslednji izrek [16, 19]:

Eulerjev izrek, kdaj imamo Eulerjev graf:

Naj bo G povezan graf. Potem je G Eulerjev graf natanko takrat, ko ima vsako vozlišče G sodo stopnjo – ***EULERJEV IZREK.***

Dokaz za ta izrek lahko bralec zasledi v literaturi [16, 19]. Zato si v nadaljevanju raje poglejmo algoritem, s katerim je mogoče izvesti konstrukcijo Eulerjevega obhoda, takoimenovani Fleuryjev algoritem [16, 19].

Fleuryjev algoritem za konstrukcijo Eulerjevega obhoda:

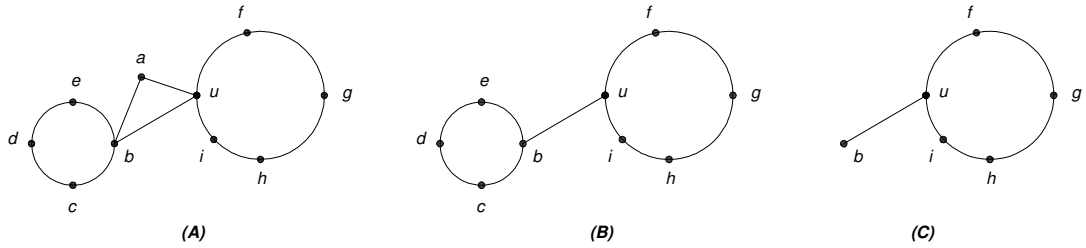
Če je *G* Eulerjev graf, potem lahko vedno izvedemo naslednje korake in dobimo Eulerjev obhod grafa *G*. Algoritem poteka takole [16, 19]:

1. *KORAK: Izberemo začetno vozlišče.*
2. *KORAK: Prečkamo poljubno povezavo, most izberemo le v primeru, kadar ni na voljo nobene druge povezave.*
3. *KORAK: Prehojeno povezavo odstranimo. Prav tako odstranimo vsa vozlišča, ki so postala izolirana.*
4. *KORAK: Končamo, ko ni nobene povezave več.*

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

Primer:

Fleuryjev algoritem lahko ponazorimo na primeru na sliki 28 na grafu (A), kjer poskušamo poiskati Eulerjev obhod grafa [16, 19]. Začenši v *u*, lahko izberemo povezavo *ua,* potem pa *ab.* Ko odstranimo ti dve povezavi (in izolirano vozlišče a), dobimo graf (B). Povezave *bu* seveda še ne smemo uporabiti, ker je most, zato izberemo povezavo *bc,* potem pa še *cd, de* in *eb.* Odstranimo vse uporabljene povezave (in vozlišča *c*, *d* in e), pri čemer dobimo graf (C). Zdaj izberemo povezavo *bu,* ki je prej nismo smeli izbrati, saj je bila most. Prehodimo še cikel *ufghiu* in zaključimo. Eulerjev obhod se torej glasi: *u-a-b-c-d-e-b-u-f-g-h-i-u.*



Slika 28: Primer uporabe Fleuryjevega algoritma

4.1.1 Problemi Eulerjevega tipa

4.1.1.1 Poleulerjevi grafi

Vrnimo se k problemu prečkanja sedmih mostov. Denimo, da bi meščane še vedno zanimal sprehod, ki bi prečkal vsak most natanko enkrat, vendar ne bi več vztrajali, da se mora sprehod vrniti v začetno vozlišče.

Tedaj bi imeli opravka s takoimenovanim **odprtim sprehodom**, ki je sprehod, pri katerem sta začetno in končno vozlišče različni. Odprt sprehod, na katerem so

vse povezave grafa, pa imenujemo Eulerjev sprehod. Tako pridemo do naslednje definicije [16, 19].

Definicija za poeulerjev graf:

Povezan graf je poleulerjev, če obstaja odprt sprehod, na katerem so vse povezave grafa G (poleulerjev sprehod).

Podajmo še naslednji izrek [16, 19]:

Izrek:

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

4.1.1.2 Problem kitajskega poštarja

Pomemben problem je problem kitajskega poštarja, katerega smo na kratko spoznali že v začetku poglavja 4. Problem se je pojavil v različnih preoblekah in ga je leta 1962 formuliral Meigu Guan iz Kitajske.

Problem kitajskega poštarja se glasi takole (1. formulacija) [16, 19]:

Poštar želi razdeliti pošto vzdolž 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 poštar običajno nekatere dele ulic prehoditi večkrat in želi čim bolj skrajšati skupno dolžino teh delov. Podobni problemi so se pojavili tudi v drugih kontekstih. Tak primer je obsežna študija o poteh plugov za čiščenje snega pred leti v Zürichu [16]. 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 že začela s podobnimi raziskavami za optimiranje pometanja in čiščenja ulic [16].

S pojmom uteženih grafov smo se srečali že v poglavju 2. Utežen graf je graf, v katerem je vsaki povezavi prirejeno pozitivno število, ki ga imenujemo utež. Kitajski problem poštarja lahko z besednjakom uteženih grafov definiramo takole [16, 19]:

Problem kitajskega poštarja se glasi (2. formulacija):

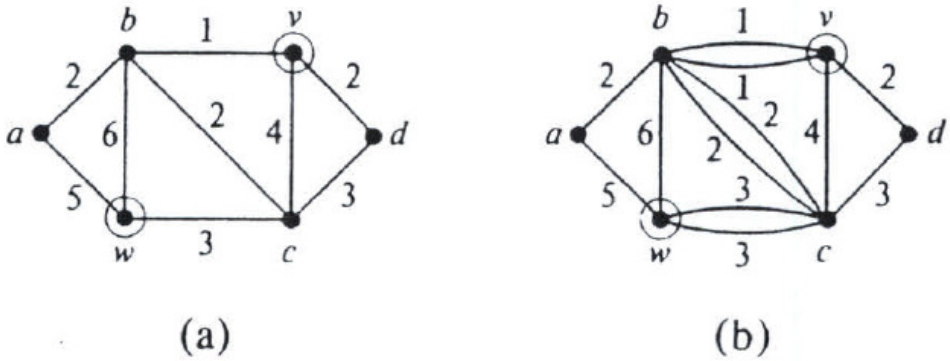
Poišči zaprt sprehod z najmanjšo skupno težo, na katerem je vsaka povezava grafa vsebovana vsaj enkrat.

Splošna rešitev problema kitajskega poštarja je algoritem, v katerem kombiniramo ideje Fleuryjevega algoritma in algoritma za iskanje najkrajših poti [16, 19]. 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 problema 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 [16, 19]. Če želimo poiskati optimalno rešitev za problem 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 na katerih je vsota uteži minimalna. Takemu naboru povezav rečemo **minimalno popolno prirejanje** [16, 19].

V nadaljevanju si poglejmo primer, na katerem bomo osvetlili problematiko reševanja problema kitajskega poštarja [16, 19].

Primer:

Za dani graf na sliki 29 poišči rešitev problema kitajskega poštarja.



Slika 29: 1. primer reševanja problema kitajskega poštarja

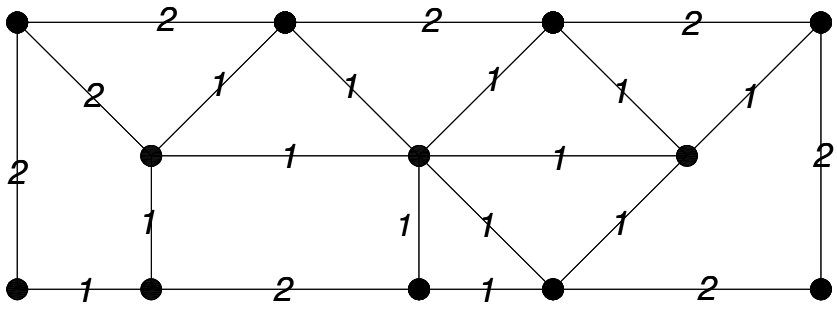
Rešitev se glasi takole (graf (a) vsebuje samo 2 vozlišči lihe stopnje!):

1. Poišči v grafu (a) vozlišči lihe stopnje, to sta vozlišči v in w (3 stopnje).
2. Med vozliščema v in w poišči pot z najmanjšo težo, to je pot v-b-c-w.
3. Skupna teža te poti je: 1+2+3 = 6.
4. Podvoji vsako od povezav na tej poti. Tako pridemo do Eulerjevega grafa (b).
5. Poišči Eulerjev obhod na tem grafu. Če uporabimo Fleuryjev algoritem, dobimo npr. naslednji rezultat: a-b-v-d-c-v-b-c-b-w-c-w-a.
6. Edine povezave, ki smo jih uporabili dvakrat, so povezave na poti v-b-c-w.

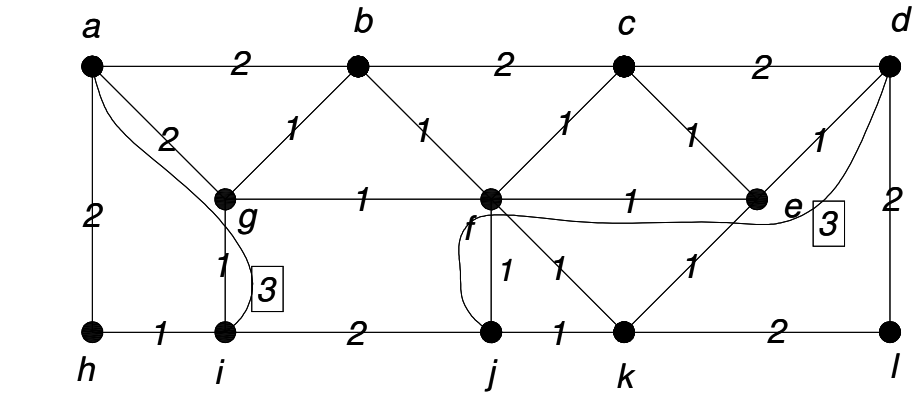
V nadaljevanju si poglejmo še en primer, na katerem bomo osvetlili problematiko reševanja problema kitajskega poštarja [16, 19].

Primer:

Za dani graf na sliki 30 poišči rešitev problema kitajskega poštarja.



a.)



b.)

Slika 30: 2. primer reševanja problema kitajskega poštarja

Rešitev se glasi takole (graf (a) vsebuje 4 vozlišča lihe stopnje, oznake vozlišč so podane na grafu b)):

1. Poišči v grafu (a) vozlišča lihe stopnje, to so vozlišča a, i, j in d (3 stopnje).

2. Med temi vozlišči poišči poti z najmanjšo težo, to so poti s skupnimi težami: *d(a,i) = 3, d(a,j) = 4, d(a,d) = 5, d(i,j) = 2, d(i,d) = 4, d(j,d) = 3*.

3. Izmed 4 vozlišč a, i, j in d moramo izbrati pol manj, to je dve dodatni poti z najmanjšo težo. Izberemo poti med a in i ter med j in d (skupna dolžina teh dveh poti je 6), s čimer pokrijemo vsa vozlišča (a, i, j, d), pri katerih je vsota uteži povezav minimalna. Optimalen obhod bo imel strošek enak vsoti vseh uteži vseh povezav originalnega grafa (29), ter dodatni uteži dodatnih dveh poti (6). Skupni strošek obhoda bo torej 29+6=35.

4. Podvoji vsako od povezav na obeh dodatnih poteh. Tako pridemo do grafa (b) na sliki 30.

5. Poišči Eulerjev obhod na tem grafu. Če uporabimo Fleuryjev algoritem, dobimo npr. naslednji rezultat: a-b-c-d-e-c-f-b-g-a-h-i-g-f-j-k-f-e-k-l-d-e-f-j-i-g-a.

4.2 Hamiltonovi grafi

Hamiltonovi grafi so tisti grafi, pri katerih obstaja cikel, ki gre skozi vsa vozlišča grafa [16, 19]. Značilna problema, ki sta lahko predstavljena v smislu Hamiltonovih ciklov, sta

problem šahovskega konja in problem trgovskega potnika, ki smo ga omenili že na začetku poglavja 4. V tem problemu je potrebno poiskati takšno pot, ki obišče vsako vozlišče natanko enkrat in se vrne na začetno vozlišče. Pri tem mora pot pokrivati najkrajšo možno skupno razdaljo, kar z drugimi besedami pomeni, da je potrebno najti minimalno - uteženi Hamiltonov cikel v grafu [16, 19].

4.2.1 Potrebni in zadostni pogoji za hamiltonskost grafov

Ugotoviti, ali je nek graf Hamiltonov ali ne, ni tako preprosto kot pri Eulerjevih grafih. To pomeni, da ne poznamo nobenega takšnega pogoja, kot npr. velja Eulerjev izrek za Eulerjeve grafe. Iskanje potrebnih in zadostnih pogojev za hamiltonskost grafa je pomembno področje proučevanja v današnji teoriji grafov [16, 19].

Zato je smiselno preučevati zgolj tiste različne vrste razredov grafov, ki so gotovo Hamiltonovi. Na primer: očitno je, da so cikli C_n (glej sliko 14) Hamiltonovi grafi za vse vrednosti n. Tudi polni grafi K_n (glej sliko 13) so Hamiltonovi, če je n $\geq$ 3. Če npr. vozlišča označimo z 1, 2, ..., n, potem je Hamiltonov cikel gotovo 1-2-3-... -n [16, 19].

Če vzamemo Hamiltonov graf in mu dodamo nekaj povezav, potem je dobljeni graf še vedno Hamiltonov, saj lahko uporabimo isti Hamiltonov cikel kot pri prvotnem grafu. Torej je za grafe z veliko povezavami ali velikimi stopnjami vozlišč bolj verjetno, da bodo Hamiltonovi, kot za grafe z manj povezavami ali majhnimi stopnjami vozlišč. To trditev lahko povemo bolj natančno na različne načine. Dva najpomembnejša sta zadostna pogoja G.A. Diraca in O. Oreja, objavljena leta 1952 in 1960 [16, 19].

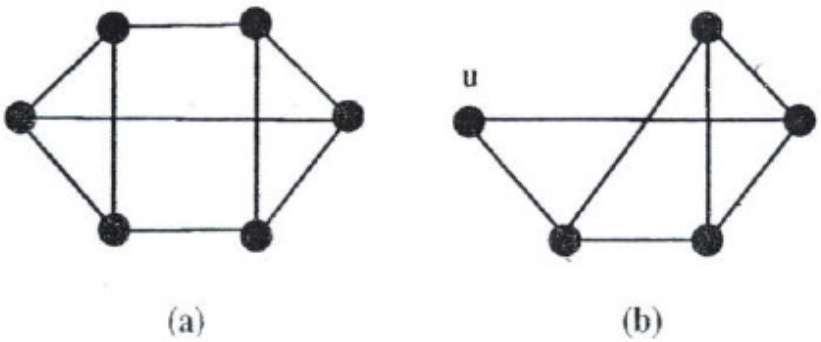
<u>Diracov izrek za ugotavljanje Hamiltonskosti grafa</u>

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

Orejev izrek za ugotavljanje Hamiltonskosti grafa

Naj bo G enostaven graf z n vozlišči in n ≥ 3. Če je st(v) + st(w) ≥ n za vsak par nesosednjih vozlišč v in w, potem je G Hamiltonov graf.

Uporaba dotičnih izrekov je ponazorjena na grafih, ki so prikazani na sliki 31 [16].



Slika 31 Primer dveh grafov (Diracov in Orejev izrek)

Iz slike 31 je razvidno, da je v grafu (a) število vozlišč n = 6 in stopnja st(v) = 3 za vsako vozlišče v, zato je ta graf gotovo Hamiltonov po Diracovem izreku. V grafu (b) je n = 5, toda v vozlišču u je stopnja st(u) = 2, zato Diracovega izreka ne moremo uporabiti. Vendar pa po drugi strani velja, da je je st(v) + st(w) ≥ 5 za vse pare nesosednjih vozlišč v in w (pravzaprav za vse pare vozlišč v in w), zato je graf gotovo Hamiltonov po Orejevem izreku [16, 19].

4.2.2 Problemi hamiltonskega tipa

V nadaljevanju si bomo pogledali, kaj so to Polhamiltonovi grafi, ter se nekoliko pobliže seznanili s problemom trgovskega potnika, ki smo ga omenili že na začetju poglavja 4.

4.2.2.1. Polhamiltonovi grafi

Kot pri Eulerjevih grafih, poskušamo tudi tukaj uporabiti sorodne ideje in rezultate [16, 19].

Definicija Polhamiltonovih grafov

Polhamiltonovi grafi so tisti grafi, v katerih obstaja pot (ne pa cikel!), ki obišče vsako vozlišče natanko enkrat. Tako pot običajno imenujemo Hamiltonova pot.

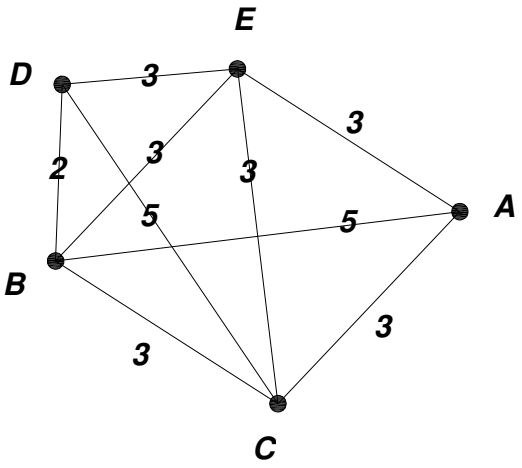
V splošnem velja, da ni nobenega generalnega kriterija, ki bi testiral, če je dani graf polhamiltonovega tipa.

4.2.2.2 Problem trgovskega potnika

Problem trgovskega potnika je verjetno najbolj raziskovan problem kombinatorične optimizacije. 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 tudi v mnogih drugih problemih, včasih pri zelo različnih kontekstih.

Problem trgovskega potnika se glasi [16, 19]:
Trgovski potnik želi obiskati nekaj mest in se vrniti v svoje mesto, tako da bo vsako mesto obiskal natanko enkrat. Pri tem naj bo skupna prevožena razdalja najkrajša možna. Katero pot naj izbere, če pozna vse razdalje med mesti?

Načeloma lahko tovrsten 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 takšne kot na sliki 32, potem mora trgovski potnik iz mesta A obiskati mesta v zaporedju A-C-B-D-E-A (ali v obratnem vrstnem redu A-E-D-B-C-A), s skupno razdaljo 14 [16, 19].



Slika 32: Problem trgovskega potnika

Če uteži v grafu ne pomenijo razdalj, ampak čas ali stroške, potrebne za potovanje, potem dobimo rešitev problema trgovskega potnika tako, da poiščemo cikel z najmanjšim časom ali stroški [16, 19].

Ko pa se število mest poveča, hitro zabredemo v težave, saj ni znan algoritem, ki bi dal enostavno in hitro rešitev problema. Iskanje optimalne rešitve namreč načeloma zahteva pregled vseh možnih poti in izbiro najkrajše, kar uvršča tovrsten problem med takoimenovane NP-teške probleme [16, 19]. Rešitev je na takšen način še mogoče najti v razumnem času za primere z npr. desetimi mesti, saj je takrat število vseh možnih ciklov 9! = 362880. Tedaj bo računalnik, ki lahko pregleda milijon poti na sekundo, našel optimalno rešitev v cca. 0.36 sekunde. Po drugi strani pa za 20 mest obstaja že okoli $1.22 \cdot 10^{17}$ možnih poti in računalnik bi za rešitev pri isti hitrosti pregledovanja potreboval skoraj 4000 let! [19].

Zato se moramo običajno zadovoljiti s približnimi rešitvami. To pa nas pripelje do obravnave algoritmov, s katerimi izračunarno zgolj približne rešitve, ter do metod za ocenjevanje vrednosti optimalne rešitve.

Natančna formulacija problema trgovskega potnika bi se glasila takole: poiskati želimo cikel z najmanjšo utežjo, ki obišče vsako vozlišče (Hamiltonov cikel z najmanjšo utežjo).

Seveda pri tem privzamemo, da je dani uteženi graf Hamiltonov, ali pa imamo opravka z uteženim polnim grafom [16, 19].

Zanimivo je primerjati ta problem s problemom kitajskega poštarja, ki ga lahko razumemo kot Eulerjev analogon tega problema. Pri poštarju ni nobenih težav, saj 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 pa 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 takšnega algoritma. Zato je še posebej pri velikem številu vozlišč najprimerneje, da poiskušamo poiskati le približne rešitve [19].

4.3 Primeri problemov Eulerjevega in Hamiltonovega tipa

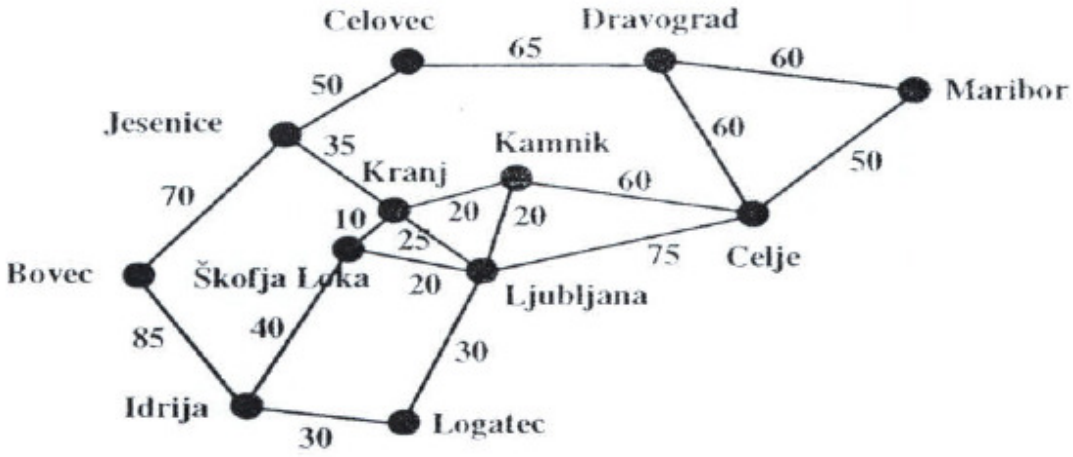
4.3.1 Problem razporejanja opravil

Nekaj neodvisnih opravil je potrebno izvesti na enem računalniku. Opravila so zapleteni računalniški programi, zato je potrebno pred vsakim opravilom nastaviti računalnik posebej za vsako 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, nas zanima, kako je potrebno razporediti izvajanje opravil, da bodo skupni stroški najmanjši? Povezavo med tem problemom in problemom trgovskega potnika lahko vidimo, če si problem predstavimo grafično. Pri problemu trgovskega potnika narišemo utežen graf, v katerem vozlišča ustrezajo mestom, ki jih mora potnik obiskati, povezave ustrezajo cestam med mesti, uteži na povezavah pa razdaljam med mesti. Pri problemu razporejanja opravil pa narišemo utežen poln graf, v katerem vozlišča ustrezajo opravilom, povezave povezujejo

opravila, uteži pa ustrezajo stroškom nastavitve računalnika pri ustreznem paru opravil [16].

4.3.2 Problem trgovskega potnika pri obhodu slovenskih mest

Poiskati je potrebno rešitev za problem trgovskega potnika pri obhodu slovenskih mest, prikazan v obliki grafa na sliki 33 [19].



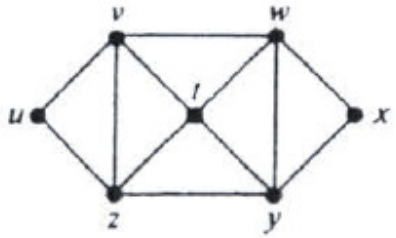
Slika 33: Problem trgovskega potnika pri obhodu slovenskih mest

Rešitev, ki jo dobimo z enim izmed algoritmov za reševanje problema trgovskega potnika (glej poglavje 8.), se glasi [19]:

Maribor -Dravograd - Celovec - Jesenice - Bovec - Idrija - Logatec - Ljubljana - Škofja Loka - Kranj - Kamnik - Celje - Maribor. Pri tem se prepotuje pot v dolžini 550 km.

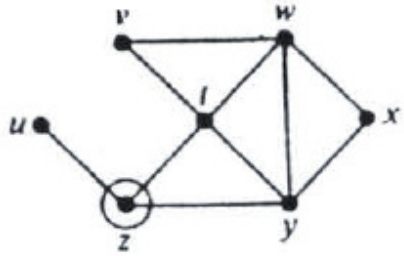
4.3.3 Primer uporabe Fleuryjevega algoritma

Z uporabo Fleuryjevega algoritma poišči Eulerjev obhod za graf na sliki 34, začni z odstrantvijo povezav u-v in v-z [16, 19].



Slika 34: Primer uporabe Fleuryjevega algoritma

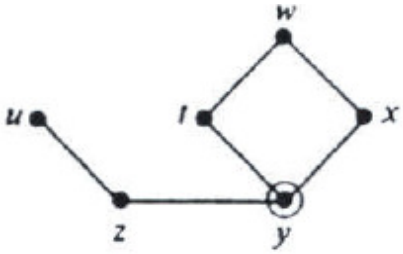
Če na sliki 34 odstranimo povezavi u-v in v-z, dobimo graf na sliki 35.



Slika 35: Odstranitev povezav u-v in v-z

Rešitev:

1. Na sliki 35 najprej odstranimo povezavo z-t. Povezave u-z namreč ne smemo odstraniti, saj je most.
2. Nato odstranimo povezavo v-t, nato v-w in izolirano vozlišče v.
3. Odstranimo še povezavo w-y, pri čemer pridemo do slike 36.

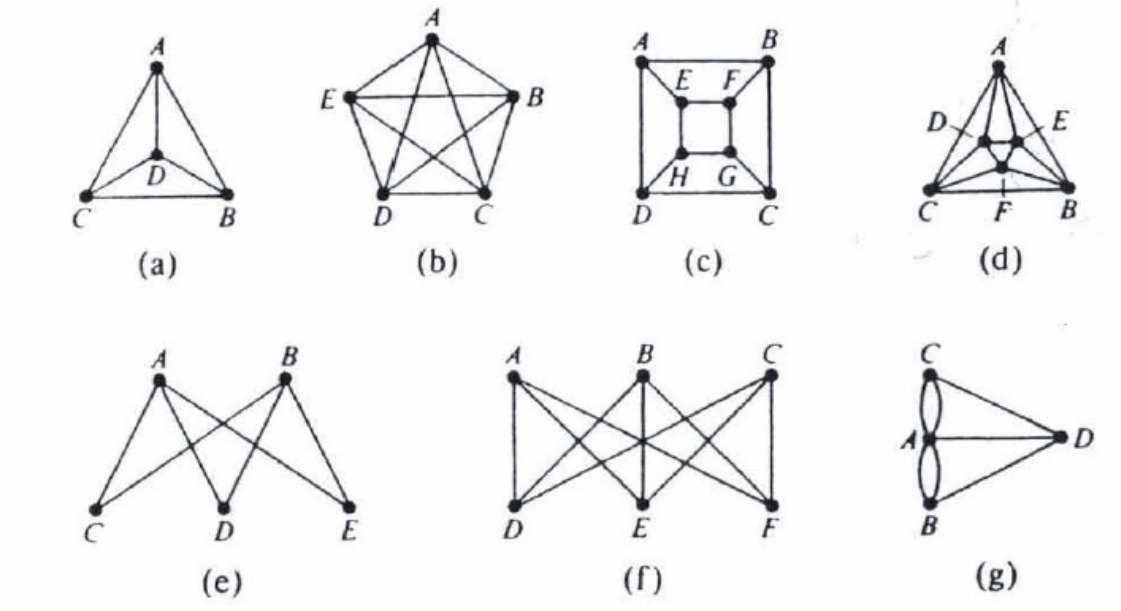


Slika 36: Graf z odstranjenimi povezavami

4. Sedaj ne smemo odstraniti povezave y-z, ker je most. Zato raje prehodimo cikel y-t-w-x-y in se vrnemo po povezavah y-z in z-u (ker ni druge možnosti) v izhodišče. Eulerjev obhod se torej glasi: u-v-z-t-v-w-y-t-w-x-y-z-u.

4.3.4 Primer določanja, če gre za Eulerjeve oz. Hamiltonove grafe

Določi, kateri od naslednjih grafov so Eulerjevi ali Hamiltonovi ali oboje in zapiši Eulerjev obhod ali Hamiltonov cikel, kjer je to mogoče [16].



Slika 37: Primeri grafov

Rešitev je podana na sliki 38.

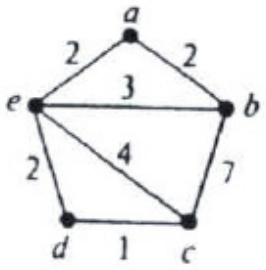
	Eulerjev graf	*Eulerjev obhod*
(a)	ni	...
(b)	je	*ABCDEACEBDA*
(c)	ni	...
(d)	je	*ABCADCFBEFDEA*
(e)	ni	...
(f)	ni	...
(g)	ni	...

	Hamiltonov graf	*Hamiltonov cikel*
(a)	je	*ABCDA*
(b)	je	*ABCDEA*
(c)	je	*ABCDHGFEA*
(d)	je	*ABCDFEA*
(e)	ni	...
(f)	je	*ADBECFA*
(g)	je	*ACDBA*

Slika 38: Rešitev za primere grafov na sliki 37.

4.3.5 Primer problema kitajskega poštarja

Za dani graf na sliki 39 poišči rešitev problema kitajskega poštarja [16, 19].



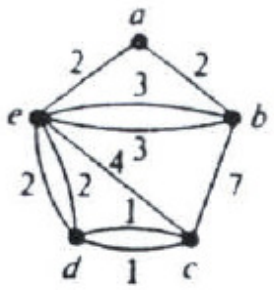
Slika 39: Primer reševanja problema kitajskega poštarja

Rešitev se glasi:

1. Poišči v grafu vozlišča lihe stopnje, to sta vozlišči b in c (3 stopnje).

2. Med tema vozliščama poišči poti z najmanjšo težo, to je pot (b-e-d-c) s skupno težo: *d(b,c) = 3+2+1 = 6.*

3. Podvoji vsako od povezav na tej poti. Tako pridemo do grafa na sliki 40.

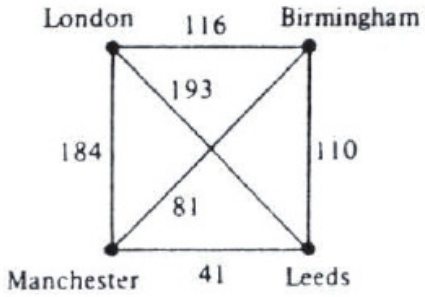


Slika 40: Podvojitev povezav na dodatni poti

4. Poišči Eulerjev obhod na tem grafu. Če uporabimo Fleuryjev algoritem, dobimo npr. naslednji rezultat: a-b-c-d-e-d-c-e-b-e-a, kjer je skupna dolžina poti enaka 27.

4.3.6 Primer problema trgovskega potnika v Londonu

Trgovski potnik je doma iz Londona. Katero pot naj izbere, da bo obiskal vsa 4 mesta na sliki 41 in pri tem naredil najkrajšo pot, pri čemer naj bo izhodišče mesto Leeds.



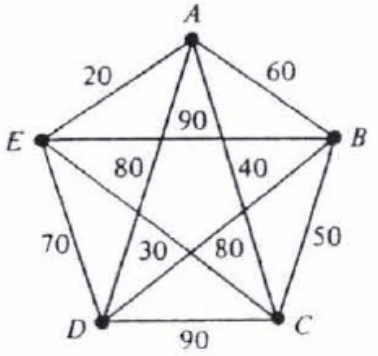
Slika 41: Primer problema trgovskega potnika iz Londona

Rešitev:

Pri rešitvi bomo uporabili metodo najbližjega soseda (glej poglavje 8.). Trgovski potnik najprej pogleda, katera izmed razdalj do mest (Manchester, Birnigham in London) je najkrajša. Očitno je to razdalja do Manchestra, zato se zapelje tja. Ponovno pogleda, katera razdalja do mest (London, Birmingham) je najkrajša. Očitno je to razdalja do Birminghama, zato se zapelje tja. Odtod mu preostane le še pot do Londona, ter povratek nazaj do Leedsa. Optimalna rešitev je torej: Leeds - Manchester - Birmingham - London - Leeds, v skupni dolžini 431 dolžinskih enot.

4.3.7 Primer problema trgovskega potnika v deželi Kombinatorija

Kralj dežele Kombinatorija se je odločil, da bo obiskal podložnike, ki živijo v štirih največjih mestih kraljevine. Njegova palača je v mestu A, ostala mesta so B, C, D in E (glej sliko 42). Kako mora potovati, da bo potovanje najkrajše?



Slika 42: Primer problema trgovskega potnika iz Kombinatorije

Pri rešitvi bomo znova uporabili metodo najbližjega soseda. Kralj najprej pogleda, katera izmed razdalj do mest (B, C, D, E) je najkrajša. Očitno je to razdalja do mesta E (20), zato se zapelje tja. Ponovno pogleda, katera razdalja do mest (B, C ,D) je najkrajša. Očitno je to razdalja do mesta C (30) , zato se zapelje tja. Odtod mu preostane le še pot do mest B ali D, zato se zapelje do bližjega mesta, to je mesta B (50). Odtod pa mu preostane le še pot do mesta D (80) in povratek nazaj do mesta A (80). Optimalna rešitev je torej: A-E-C-B-D-A , v skupni dolžini 20+30+50+80+80 = 260 dolžinskih enot.

5 DREVESA

Drevesa so v mnogih pogledih 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 kar nekaj hipotez, ki niso bile dokazane za splošne grafe, vendar pa je znano, da veljajo za drevesa.

5.1 Karakteristične lastnosti dreves

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 [16, 19].

Izrek za drevo:

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.

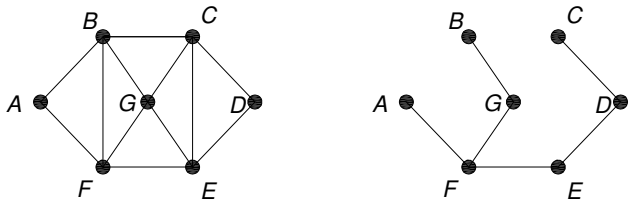
e. Vsak par vozlišč grafa T je povezan z natanko eno potjo.

f. V grafu T ni nobenega cikla. Če dodamo katerokoli povezavo, dobimo natanko en cikel.

Definicija za vpeto drevo:

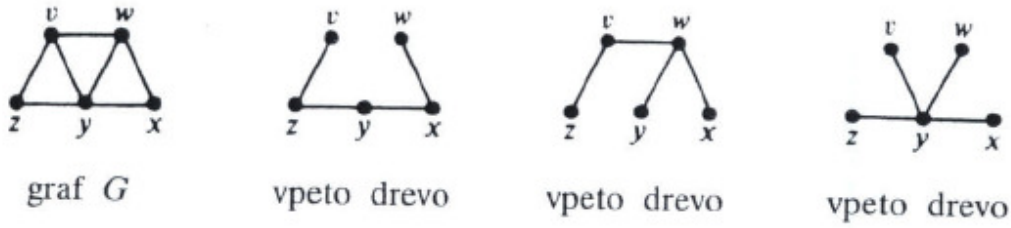
Naj bo *G* povezan graf. **Vpeto drevo** v *G* je podgraf grafa *G,* na katerem so vsa vozlišča grafa in je drevo. Povezave drevesa imenujemo **veje.**

Na sliki 43 je graf, ob njem pa eno izmed njegovih vpetih dreves.



Slika 43: Graf in eno izmed njegovih vpetih dreves

Slika 44 pa prikazuje graf in tri izmed njegovih vpetih dreves [16].



Slika 44: Graf in tri izmed njegovih vpetih dreves

V vsakem povezanem grafu G lahko najdemo vpeto drevo z uporabo ene od naslednjih dveh metod [16, 19].

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), ter opazujemo preostale cikle. Postopek ponavljamo, dokler ni nobenega od preostalih ciklov več. Dobljeni graf je vpeto drevo.

Tako na primer lahko v grafu na sliki 44 (ki ima 5 ciklov) odstranimo povezave:
vy (ki npr. pretrga cikel v-w-y-v) - ostanejo 3 cikli,
yz (ki npr. pretrga cikel v-w-y-z-v) - ostane en cikel, ter
xy (ki pretrga edini preostali cikel w-x-y-w) - ni več ciklov.

Tako dobimo drugo izmed vpetih dreves na sliki 44.

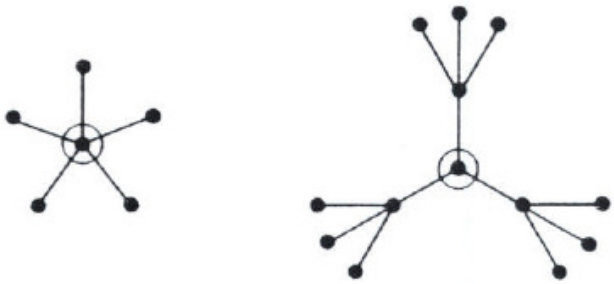
Metoda konstrukcije. Po vrsti izbiramo povezave G tako, da ne dobimo nobenega cikla.

Ponavljamo, dokler nismo vključili vseh vozlišč. Na ta način ne ustvarimo nobenega cikla, in dobimo vpeto drevo.

Tako na primer lahko v grafu na sliki 44 izberemo povezave: v-z, w-x, x-y in y-z, pri čemer dobimo prvo izmed vpetih dreves na sliki 44.

5.2 Centri in centroidi dreves

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 [16]. 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", kar pomeni, da imajo (vsaj približno) enako število vozlišč [16, 19]. Seveda nas zanima, kaj to sploh pomeni sredina drevesa? Za nekatera drevesa je to enostavno definirati, kot npr. na sliki 45.



Slika 45: Trivialne sredine dreves

Po drugi strani pa ni tako trivialno lahko določiti npr. sredini dreves, predstavljenih na sliki 46.

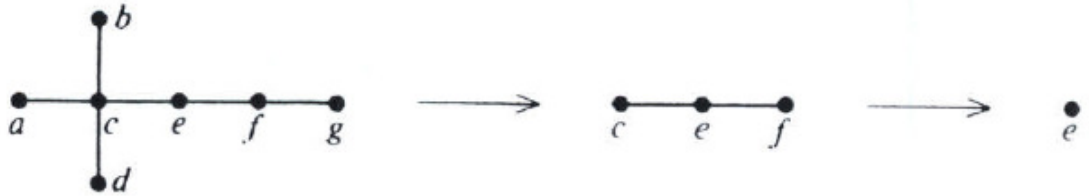
Slika 46: Netrivialne sredine dreves

Sredino drevesa lahko definiramo na več načinov. V nadaljevanju sta predstavljena dva načina za določitev sredine drevesa [16, 19].

Metoda 1 za določitev centra drevesa:

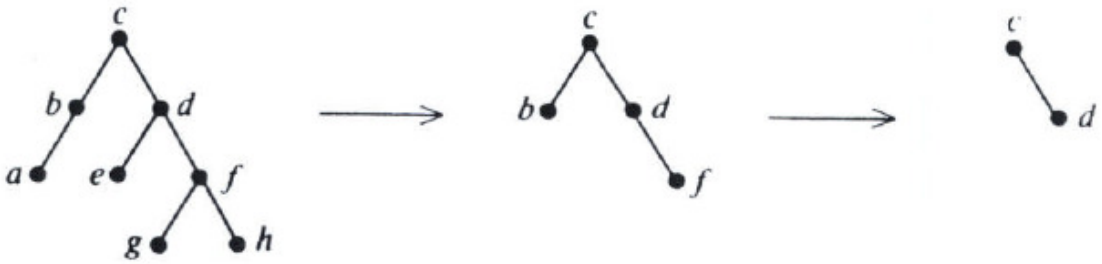
Odstranimo vsa vozlišča stopnje 1, skupaj s povezavami, ki jih imajo za krajišče. To ponavljamo tako dolgo, dokler ne dobimo ali enega vozlišča (imenovanega center) ali dveh vozlišč (imenovanih bicenter) .

Primer uporabe metode 1 je za primer centra drevesa predstavljen na sliki 47.



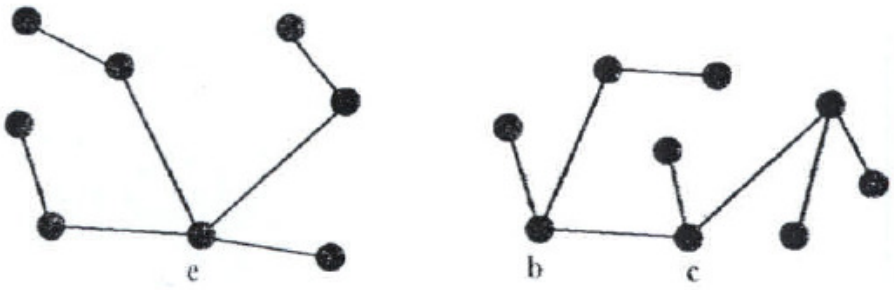
Slika 47: Določanje centra drevesa s 1. metodo

Primer uporabe metode 1 je za primer bicentra drevesa predstavljen na sliki 48.



Slika 48: Določanje bicentra drevesa s 1. metodo

Na drevesu več kot dveh vozlišč v srednini (centru) ne more biti. Na sliki 49 je prikazan še en primer centriranega drevesa s centrom *e* in bicentriranega drevesa s centrom *b*-*c*.

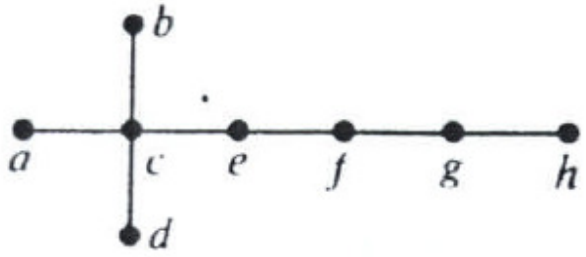


Slika 49: Centrirano in bicentrirano drevo

<u>Metoda 2 za določitev centra drevesa:</u>

Za vsako vozlišče v stopnje 2 ali več preštejemo število vozlišč v vsakem od poddreves, ki izhajajo iz v (izoliranih vej ne upoštevamo!). Naj bo n_v *najmanjše od teh števil. Dokazati se da, da za drevo z n vozlišči velja: ali obstaja samo eno vozlišče v, za katero je* $n_v \le \frac{1}{2}(n-1)$ *(centroid) ali pa obstajata dve sosednji vozlišči v in w, za kateri je* $n_v = n_w = \frac{1}{2}(n)$ *(bicentroid).*

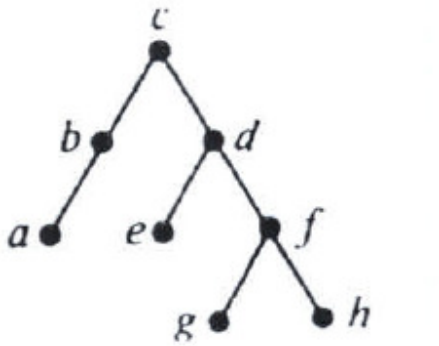
Centroid ali bicentroid si lahko predstavljamo kot "težišče" drevesa [16, 19]. Drevo s centroidom imenujemo centroidno drevo, drevo z bicentroidom pa bicentroidno drevo. Vsako drevo je centroidno ali bicentroidno, oboje hkrati pa ne more biti. Primer bicentroidnega drevesa je prikazan na sliki 50.



Slika 50: Bicentroidno drevo

Iz slike 50 je razvidno, da velja naslednje: n_c = 4, n_e = 4, n_f = 5 in n_g = 6. Ker je $n_c = n_e = \frac{1}{2}(n) = \frac{1}{2}(8) = 4$, je dotično drevo bicentroidno z bicentroidom *c-e.*

Primer centroidnega drevesa pa je prikazan na sliki 51.



Slika 51: Centroidno drevo

Iz slike 51 je razvidno, da velja naslednje: n_b = 6, n_c = 5, n_d = 3 in n_f = 5. Ker je $n_d \leq \frac{1}{2}(n-1) = \frac{1}{2}(8-1) = 3.5$, je dotično drevo centroidno s centroidom *d.*

5.3 Pregledovanje dreves

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

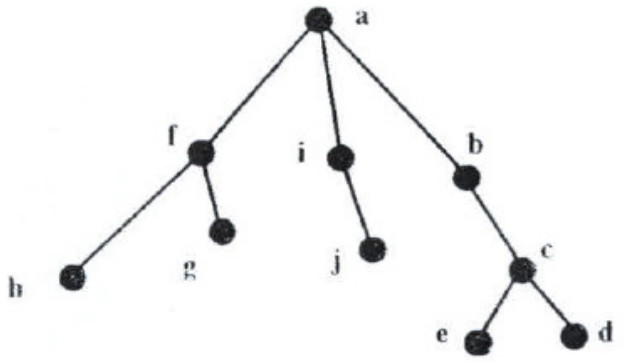
obiskovanja vozlišč [16, 19]. Običajno ju imenujemo pregledovanje v globino (DFS, depth-first search) in pregledovanje v širino (BFS, breadth-first search) [16, 19]. Pri vsaki od teh metod zapisujemo seznam že obiskanih vozlišč drevesa in označimo, v katero smer je bila povezava že uporabljena. Metodi se razlikujeta samo v načinu, kako konstruirata zaporedje obiskanih vozlišč.

5.3.1 Pregledovanje drevesa v globino

Osnovna ideja pregledovanja v globino je prodiranje kar se da v globino, preden se pot pregledovanja razveji [16, 19].

Primer:

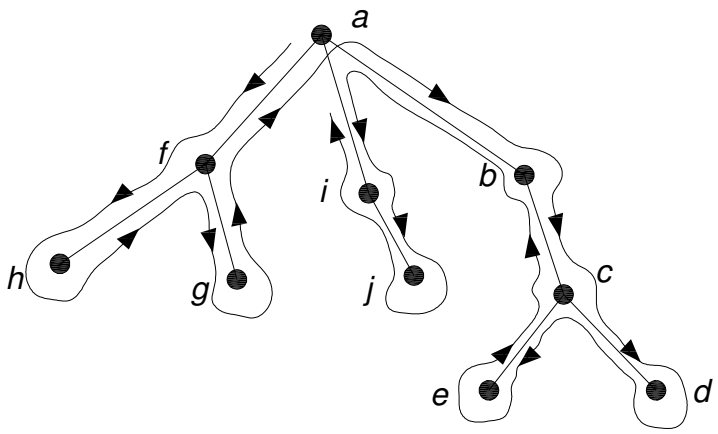
Opišimo postopek na drevesu na sliki 52.



Slika 52: Drevo T

Začnemo 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 vozlišče *g.* Ker iz *g* ne moremo nadaljevati v globino, se vrnemo v *f* in nato v *a.* Vozlišče *a* ima neobiskane sosede, to sta vozlišči *i* in *b.* Izberemo enega od njiju in nadaljujemo na opisan način. Tako dobimo enega od možnih pregledov, ki je s puščicami označen na sliki 53. Ker smo vedno izbirali enega od neobiskanih sosedov vozlišča, v katerem smo, je večkrat na voljo več

enakovrednih izbir in zato zaporedje obiskovanja vozlišč ni enolično določeno.

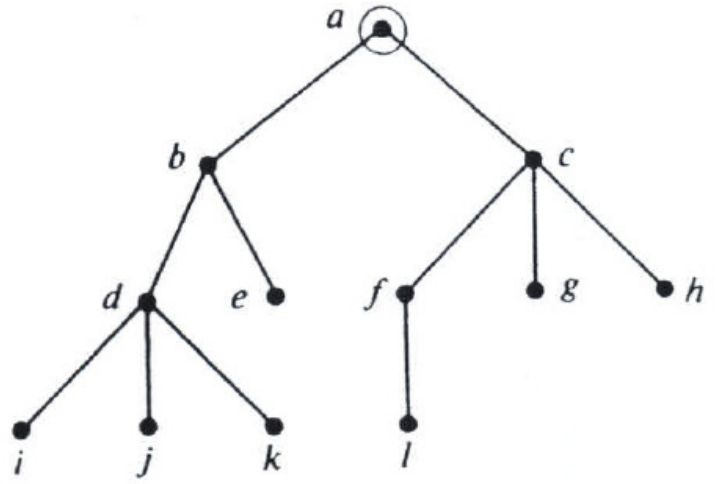


Slika 53: Drevo T z enim od pregledov v globino

Poglejmo si še en primer pregledovanja drevesa v globino.

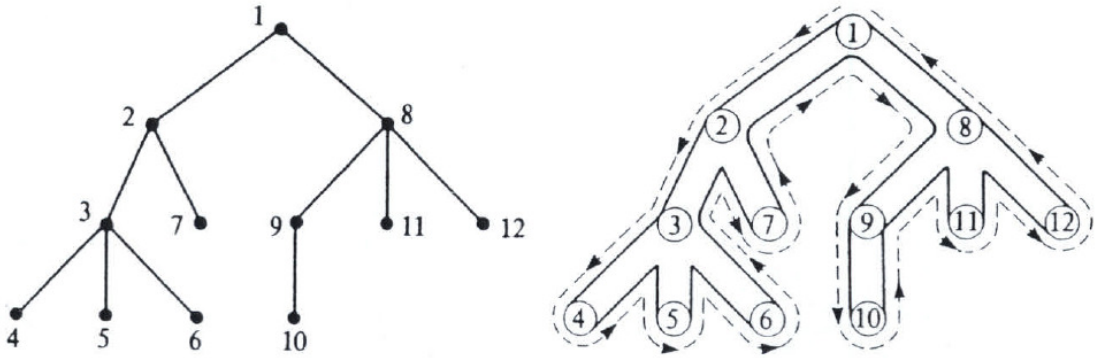
Primer:

Dano imamo drevo na sliki 54. Izvedi postopek pregledovanja v globino.



Slika 54: Drevo M

Rešitev pregledovanja v globino izvedemo podobno kot v prejšnjem primeru. Ko napredujemo, obiskana vozlišča po vrsti oštevilčimo. Pri izbiri, katero sosednjo točko bomo izbrali, se denimo vedno odločimo za najbolj levega prostega soseda, čeprav takšna sistematičnost ni nujno potrebna. Dobimo rezultat, kot ga prikazuje slika 55.



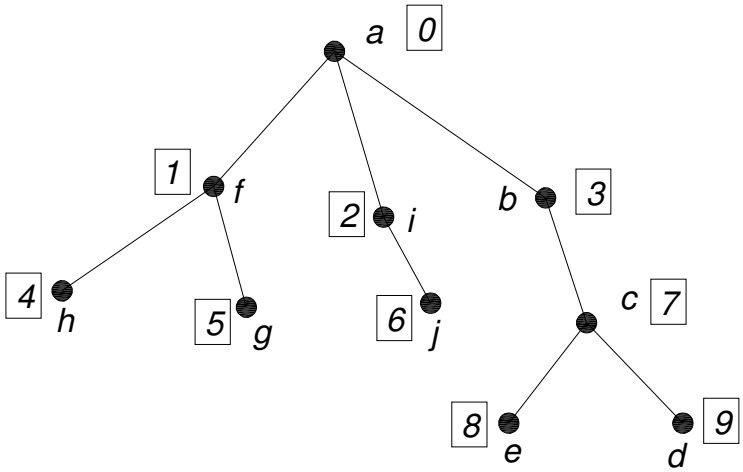
Slika 55: Označeno drevo M z enim od pregledov v globino

5.3.2 Pregledovanje drevesa 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 vozlišču, preden gremo globlje v naslednje vozlišče [16, 19].

Primer:

Opišimo postopek na drevesu na sliki 56.



Slika 56: Drevo T z enim od pregledov v širino

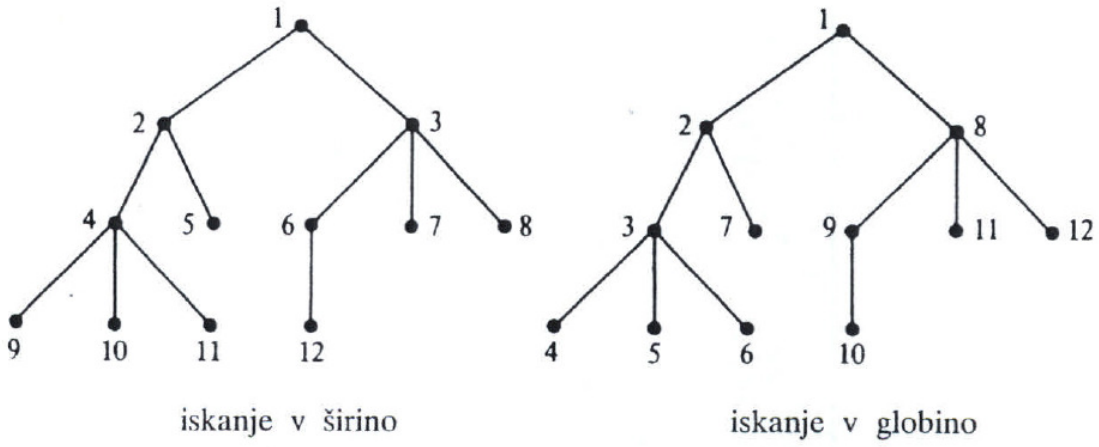
Začnimo v vozlišču *a.* Izberemo poljubno sosednje vozlišče, recimo vozlišče *f.* V naslednjem koraku izberemo enega od preostalih sosedov vozlišča *a,* na primer *i*. V

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 njegovih sosedov, na primer *h.* V naslednjem koraku izberemo edinega preostalega neobiskanega soseda *g* in ker *f* nima več nobenega neobiskanega soseda, se premaknemo nazaj v a in nato v naslednje vozlišče *i* in nadaljujemo s pregledom sosedov. Postopek ponavljamo, dokler ne oštevilčimo vseh vozlišč. Na sliki 56 smo z oznakami v okvirčkih označili enega od možnih zaporedij pregledovanja v širino z začetkom v vozlišču *a.* Seveda je običajno na voljo več enakovrednih izbir označevanja zaporedja vozlišč in zato zaporedje obiskovanja vozlišč ni enolično določeno.

Če so vozlišča drevesa narisana v vodoravnih "nivojih" kot v danem primeru, potem pri pregledovanju v širino vsakič obiščemo vsa vozlišča enega nivoja, preden se premaknemo za en nivo globlje. 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 naprej.

5.3.3 Primer razlike med obema pregledovanjema drevesa

Poglejmo si še en primer (glej sliki 54 in 58), kjer je razvidna razlika v označevanju vozlišč, če jih pregledujemo v širino ali v globino. Razlika v zaporedju oštevilčenih vozlišč je očitna.

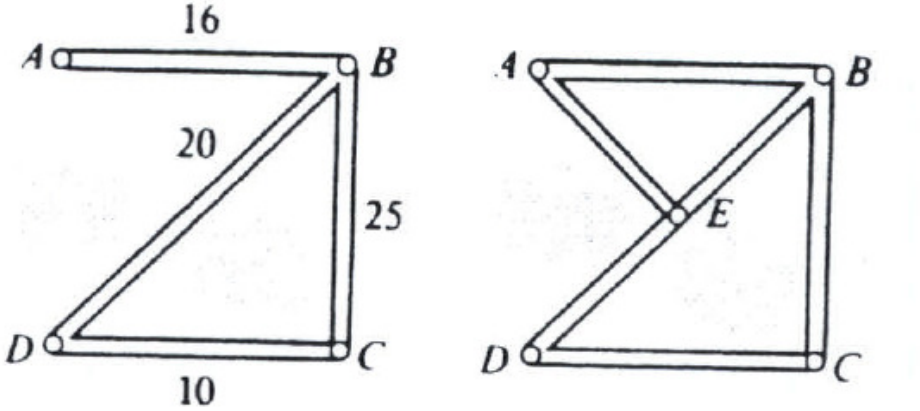


Slika 57: Primerjava iskanja v širino in iskanja v globino drevesa

5.4 Problem minimalnega vpetega drevesa

Probleme minimalno vpetega drevesa uvrščamo med tipične probleme, ki se pojavljajo pri takoimenovani optimizaciji mrežnih struktur [6].

Recimo, da želimo zgraditi sistem kanalov za izsuševanje, ki naj bo povezan z danimi lokacijami [16]. 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 (glej sliko 58)?



Slika 58: Primera sistemov kanalov za izsuševanje

Praktičnih primerov za nalogo minimalnega vpetega drevesa je veliko. Podobno kot o kanalih za namakanje bi lahko premišljali tudi o naftovodih, o komunikacijskih povezavah (telefon, internet) ali o energetskem omrežju [16].

Naštejmo še dva značilna primera problemov minimalno vpetih dreves [6, 16]:

- Načrtovanje sistema transportnih poti, ki bo povezal vse lokacije (vozlišča) s čim manjšimi stroški (stroški izgradnje in vzdrževanja poti in vmesnih vozlišč ter pripadajoče infrastrukture bodo najmanjši).

- Načrtovanje sistema optičnih kablov, ki bo povezal vsa vozlišča z izhodiščem, pri čemer bo skupna dolžina kablov najmanjša (slika 59).

Slika 59: Ilustracija problema načrtovanja sistema optičnih kablov, ki bo povezal vsa vozlišča z izhodiščem, pri čemer bo skupna dolžina kablov najmanjša.

Problem minimalno vpetega drevesa lahko definiramo na naslednji način [6, 16]:

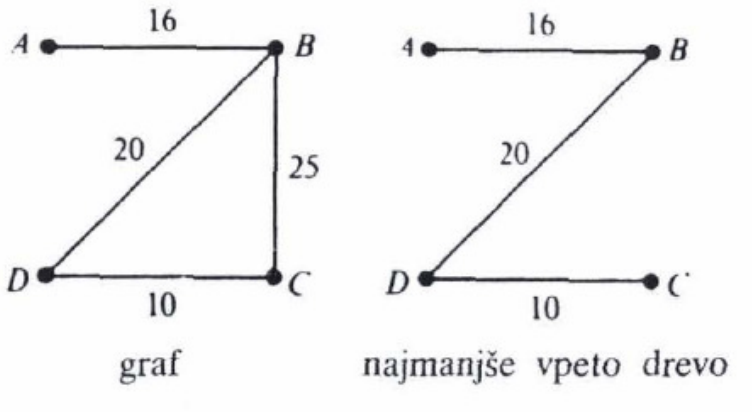
- ***Želimo realizirati takšno omrežje, ki bo povezalo vsa vozlišča grafa brez tvorjenja zank, pri čemer bo vsota povezav najkrajša/najcenejša.***

Poiskati moramo torej takšen 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.

Primer:

Za dani graf poiščite minimalno vpeto drevo (slika 60).



Slika 60: Primer grafa in njegovega minimalno vpetega drevesa

V tem 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 najdražjo povezavo BC. Skupna utež dobljenega minimalno vpetega drevesa pa je 16 + 20 + 10 = 46.

V splošnem lahko za dani uteženi graf poiščemo minimalno vpeto drevo s pomočjo naslednjih dveh algoritmov [6, 16, 19]:

- Kruskalov algoritem, ter
- Primov algoritem.

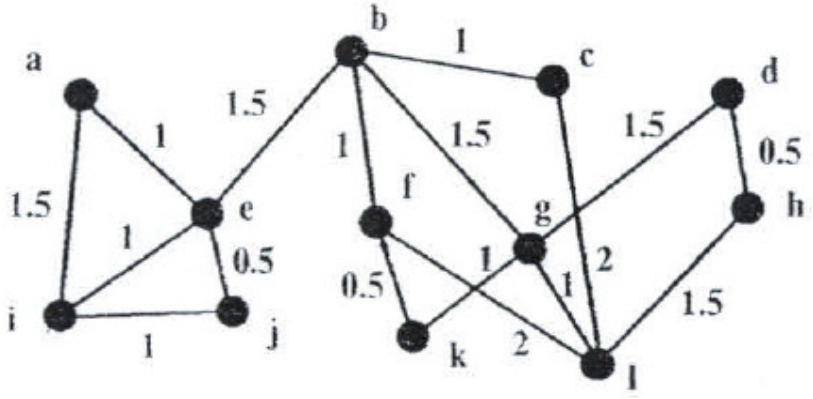
V nadaljevanju sta predstavljena oba algoritma za sestavo minimalno vpetega drevesa.

5.4.1 Kruskalov algoritem

Kruskalov algoritem imenujemo tudi požrešna metoda [16, 19]. Ime požrešna metoda 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.

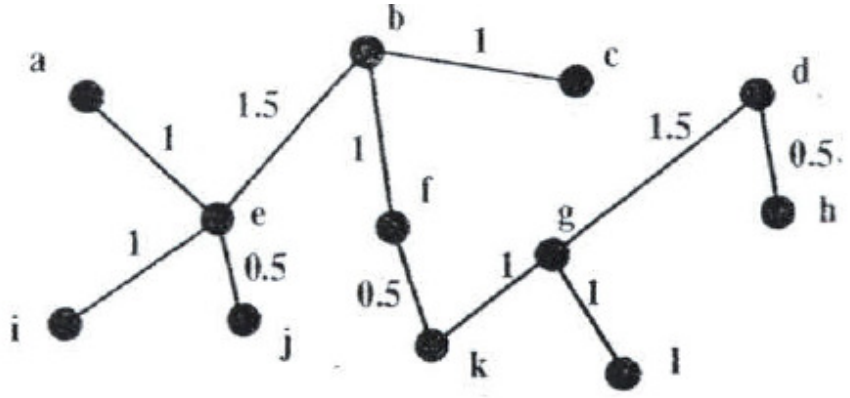
Pri Kruskalovem algoritmu sestavimo minimalno vpeto drevo grafa G tako, da po vrsti izbiramo povezave z naraščajočimi utežmi in jih dodajamo v drevo, razen v primerih, ko bi z njimi tvorili cikle (zanke). Algoritmi te vrste običajno niso uspešni v praksi (zaradi prevelike časovne potratnosti na računalnikih), uspešno pa delujejo, če z njimi računamo "peš" na majhnih grafih [6, 16].

Uporaba algoritma je predstavljena na primeru grafa na sliki 61.



Slika 61: Kruskalov algoritem na grafu

Najprej uredimo povezave po 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). Nato konstruirajmo vpeto drevo na vozliščih *V(G),* pri čemer dodajajmo povezave v prejšnjem vrstnem redu (po naraščajočih utežeh), razen če bi dodana povezava povezala že povezani vozlišči. V dodajanju zaporedja povezav *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 s skupno utežjo 10.5 na sliki 62 (Povezav *hl, cl* in *fl* ne smemo dodati, saj bi dobili cikle). Kot je razvidno, postopek dodajanja povezav vršimo toliko časa, dokler v konstruirano drevo ne vključimo vseh vozlišč. Nadaljnje dodajanje povezav je nato nepotrebno in bi le tvorilo nove cikle.

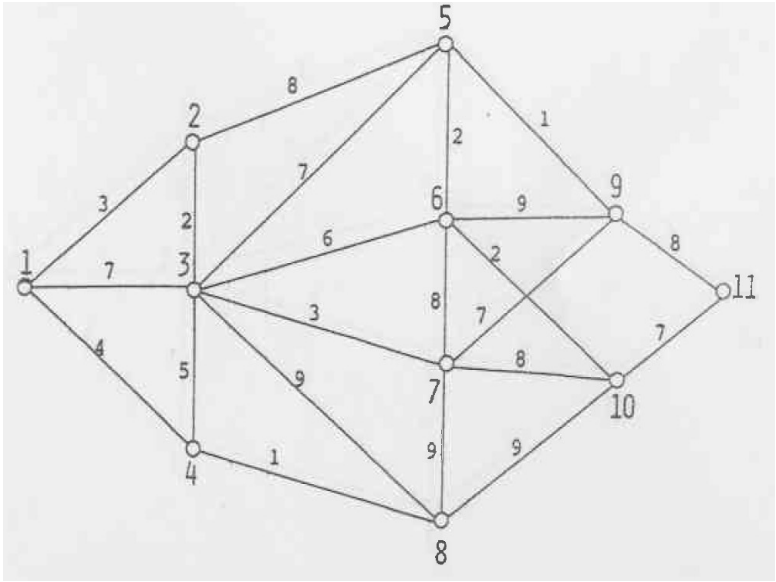


Slika 62: Minimalno vpeto drevo, dobljeno s Kruskalovim algoritmom.

V nadaljevanju si bomo pogledali še nekaj primerov, kako lahko za dane mreže z uporabo Kruskalovega algoritma učinkovito in enostavno poiščemo minimalno vpeto drevo [6].

Primer 1:

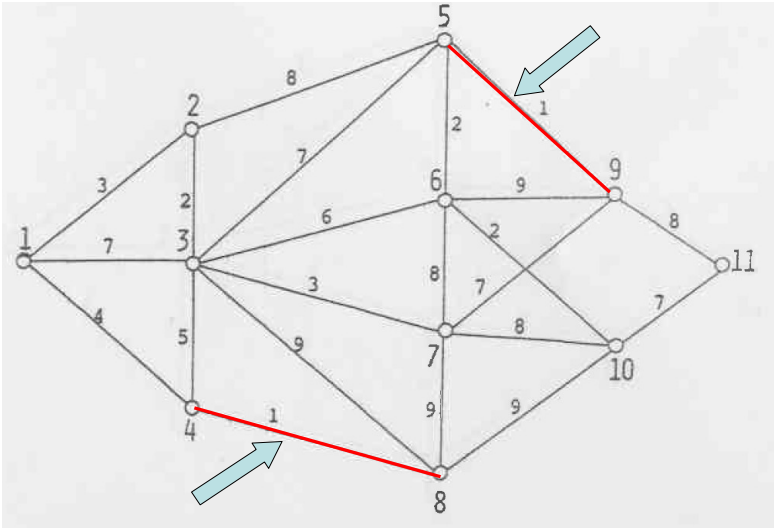
Dana je naslednja mreža (graf) transportnih poti (slika 63). Poiščite minimalno vpeto drevo s Kruskalovim algoritmom.



Slika 63: Primer grafa transportnih poti

Rešitev

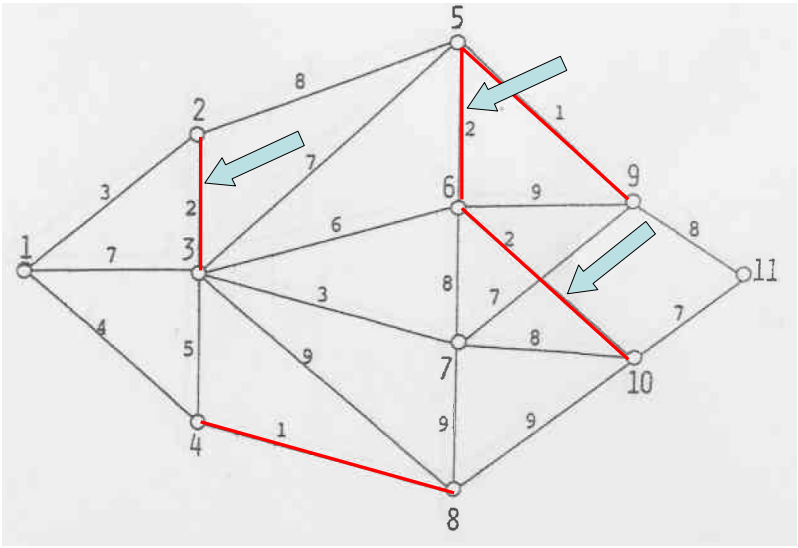
Najprej tvorimo na sliki 63 vse povezave (jih odebelimo!), ki imajo najnižjo utež, torej utež 1, pri čemer dobimo sliko 64:



• najprej utež 1!

Slika 64.: Tvorjenje povezav z utežjo 1

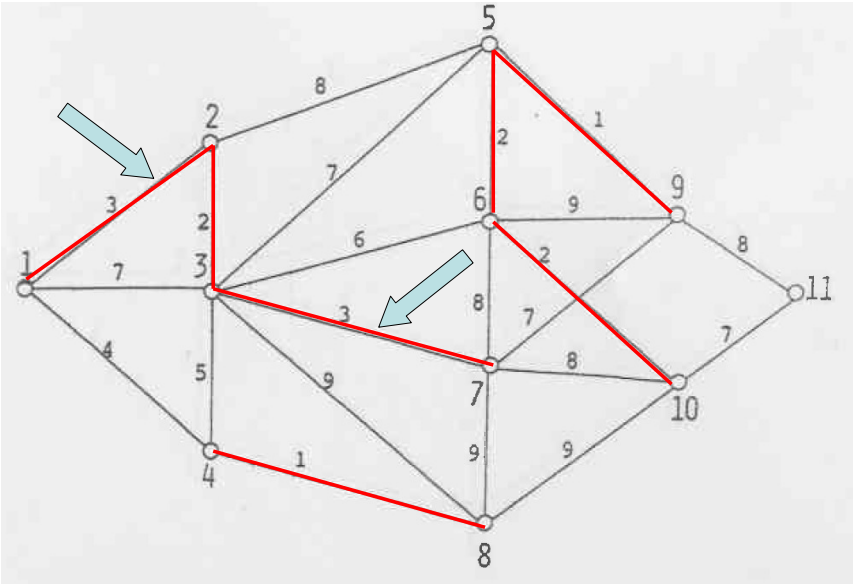
Nato tvorimo (dodamo oz. odebelimo) na sliki 64 vse povezave, ki imajo drugo najnižjo utež, torej utež 2. S tem dobimo sliko 65.



• nato utež 2!

Slika 65: Tvorjenje povezav z utežjo 2

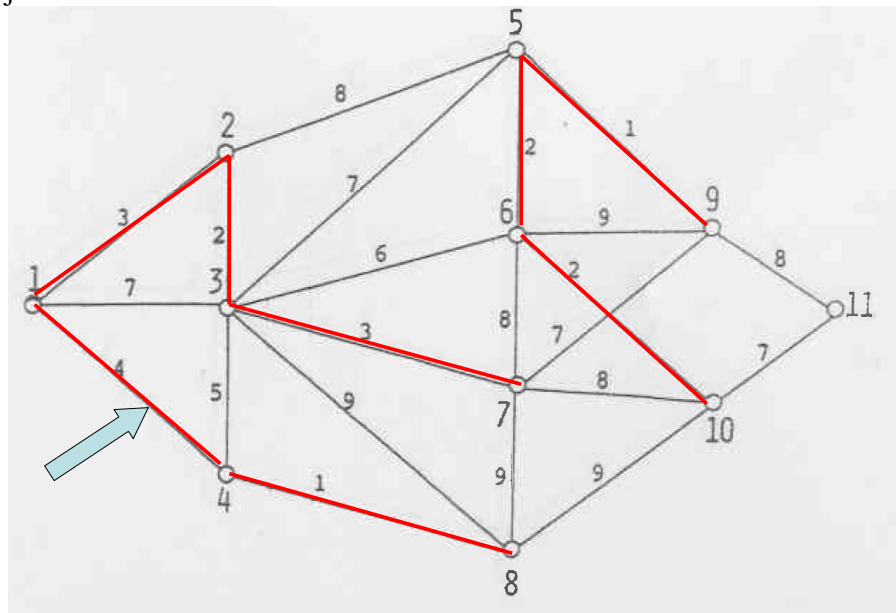
Nato tvorimo (dodamo oz. odebelimo) na sliki 65 vse povezave, ki imajo tretjo najnižjo utež, torej utež 3. S tem dobimo sliko 66:



• nato utež 3!

Slika 66: Tvorjenje povezav z utežjo 3

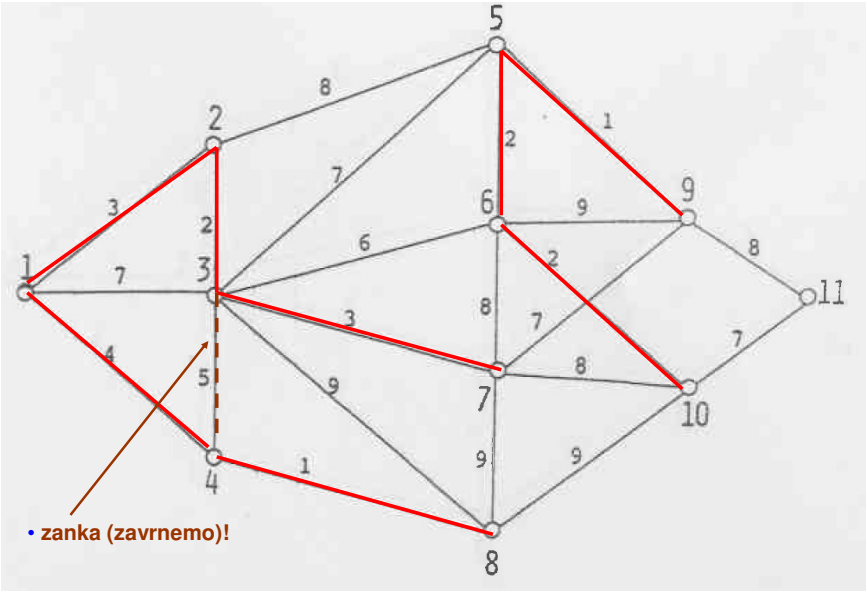
Nato tvorimo (dodamo oz. odebelimo) na sliki 66 vse povezave, ki imajo četrto najnižjo utež, torej utež 4. S tem dobimo sliko 67.



• nato utež 4!

Slika 67: Tvorjenje povezav z utežjo 4

Nato tvorimo na sliki 67 vse povezave, ki imajo peto najnižjo utež, torej utež 5, pri čemer dobimo sliko 68.

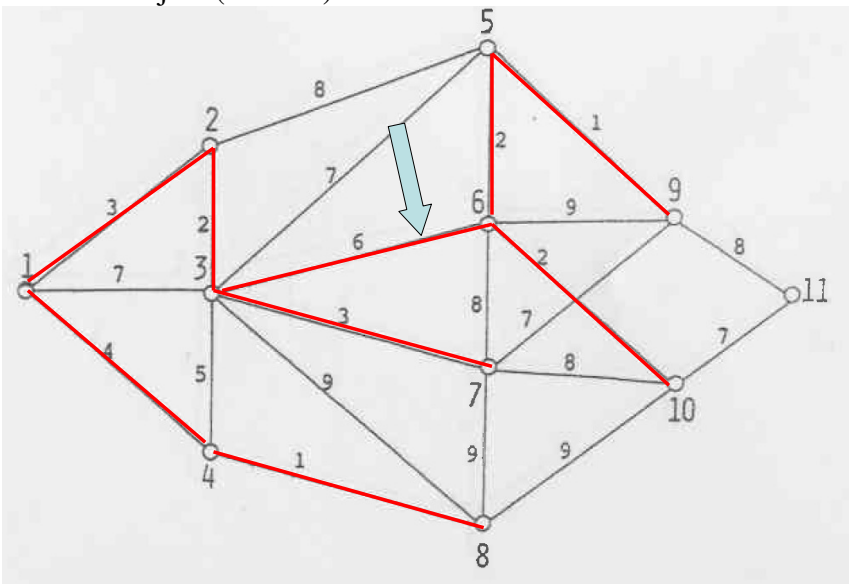


• nato utež 5!

Slika 68.: Tvorjenje povezav z utežjo 5

Ker bi dotična (dodana) povezava na sliki 68 tvorila zanko glede na že sprejete povezave, jo zavrnemo.

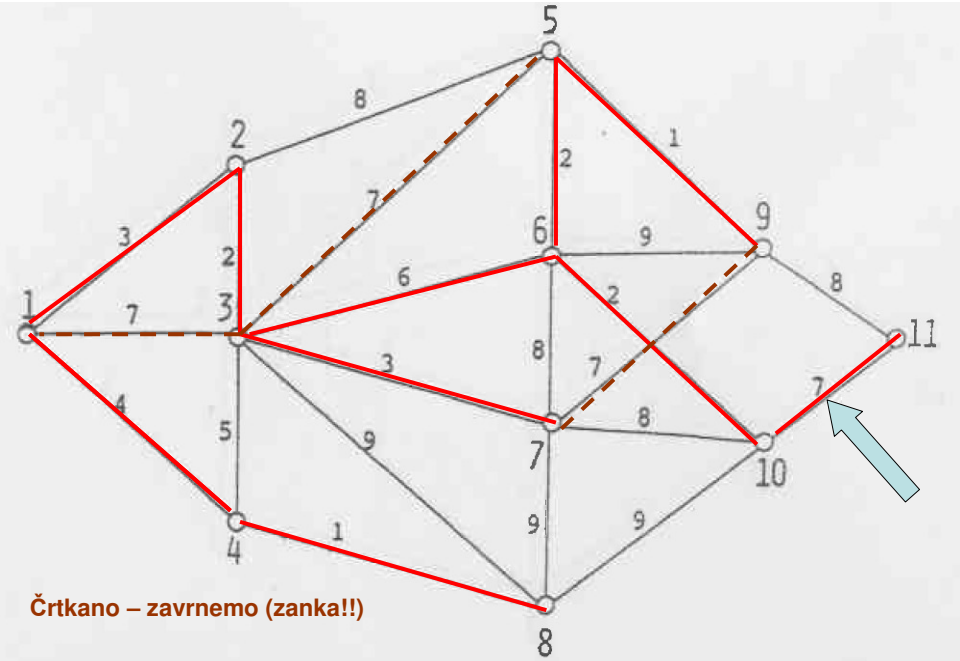
Sledijo povezave z utežjo 6 (slika 69):



• nato utež 6!

Slika 69: Tvorjenje povezav z utežjo 6

Sledijo povezave z utežjo 7 (slika 70):

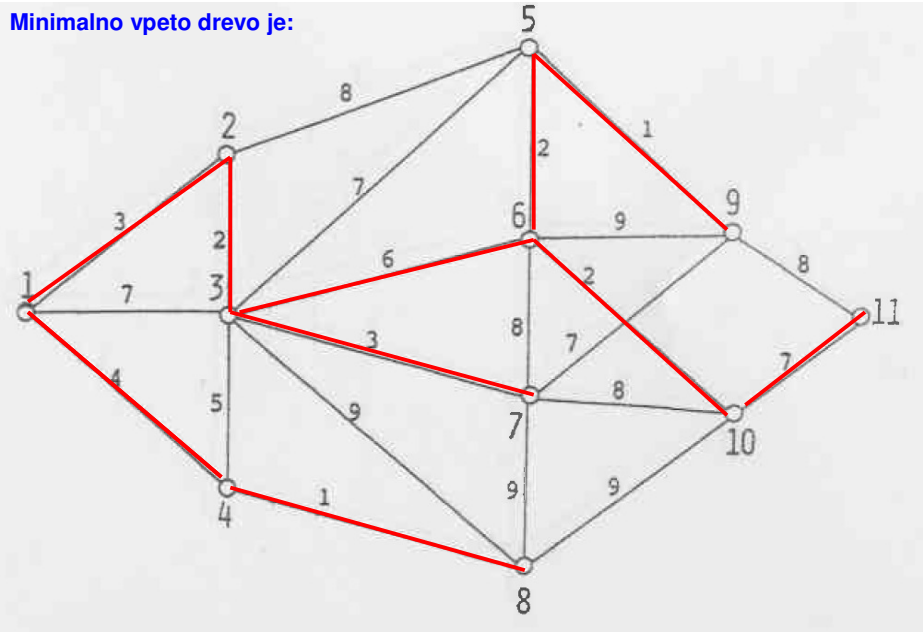


Slika 70: Tvorjenje povezav z utežjo 7

Kot lahko vidimo na sliki 70, bi v tem primeru kar tri nove povezave z utežjo 7 (glej črtkano!) tvorile zanke glede na že sprejete povezave, zato jih zavrnemo. Sprejmemo pa četrto povezavo (označeno s puščico na sliki 70), saj ta pa ne tvori zanke.

Ker smo že uspeli povezati vsa vozlišča na sliki 70, postopek končamo (torej ni potrebno obdelati še uteži 8 in 9, katerih povezave bi gotovo tvorile same zanke).

Minimalno vpeto drevo ima torej obliko, kot jo prikazujejo odebeljene črte na sliki 71.

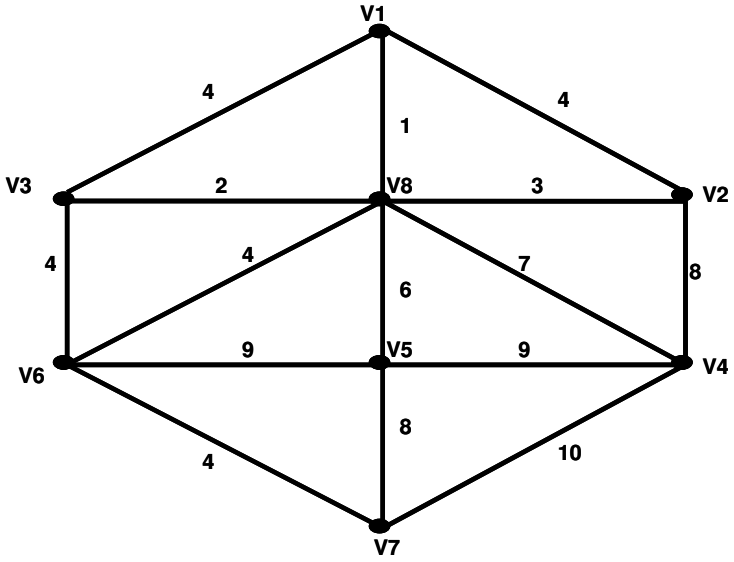


Slika 71: Minimalno vpeto drevo za primer grafa transportnih poti

Če seštejemo uteži vseh povezav na sliki 71, pridemo do njihove skupne dolžine 31. To je minimalna skupna dolžina vseh povezav, ki jo lahko z nekim drevesom vpnemo v dani graf.

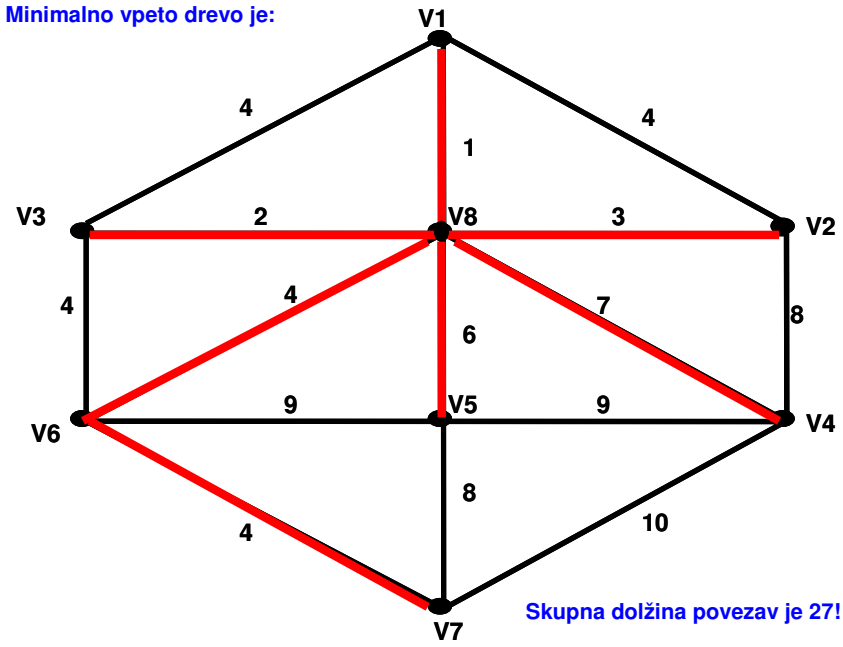
Primer 2:

Dana je naslednja mreža (graf) transportnih poti (slika 72.). Poiščite minimalno vpeto drevo s Kruskalovim algoritmom.



Slika 72: Primer grafa transportnih poti

Rešujemo na popolnoma enak način kot v prejšnjem primeru. Pri tem pridemo do minimalno vpetega drevesa z obliko, prikazano na sliki 73 (odebeljene črte na sliki 73).

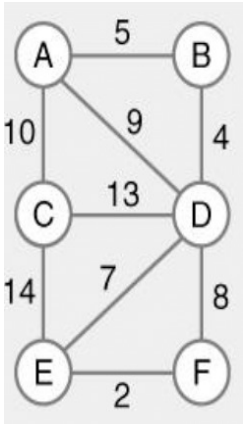


Slika 73: Minimalno vpeto drevo za primer grafa transportnih poti

Če seštejemo uteži vseh povezav na sliki 73, pridemo do njihove skupne dolžine 27. To je minimalna skupna dolžina vseh povezav, ki jo lahko z nekim drevesom vpnemo v dani graf.

Primer 3:

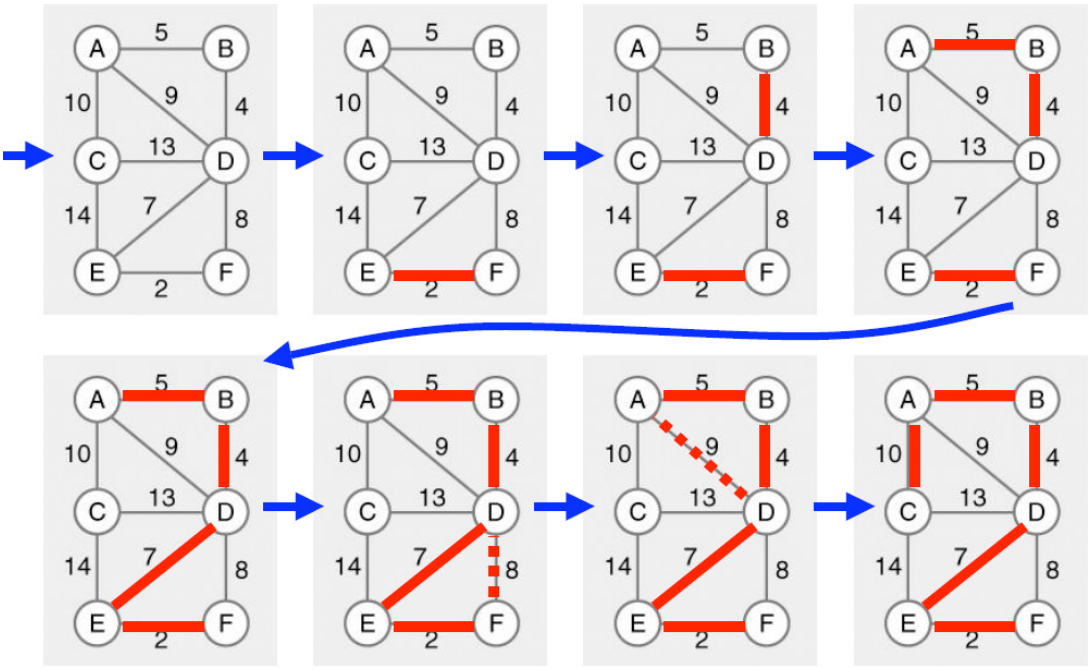
Dana je naslednja mreža (graf) transportnih poti (slika 74). Poiščite minimalno vpeto drevo s Kruskalovim algoritmom.



Slika 74.: Primer grafa transportnih poti

Rešitev:

Kot v prejšnjih primerih tvorimo povezave v posameznih fazah Kruskalovega postopka. Pri tem gremo pri sliki 74. od povezav z nižjimi utežmi do povezav z višjimi utežmi in vselej pazimo, da ne tvorimo kakšne zanke. Slika 75. prikazuje posamezne faze Kruskalovega postopka. Začnemo z najnižjo utežjo 2 (odebeljena črta), ter nadaljujemo z utežmi 4, 5 in 7. Ko pridemo do uteži 8 in 9, njunih povezav ne upoštevamo, saj bi tvorili zanko. Postopek sklenemo z utežjo 10, s čimer nam uspe povezati vsa vozlišča danega grafa v minimalno vpeto drevo (odebeljene črte na zadnji izmed slik na sliki 75.). Kot je razvidno iz slike 75, je minimalna skupna dolžina vseh povezav minimalno vpetega drevesa enaka 28.



Slika 75: Potek tvorjenja povezav minimalno vpetega drevesa v posameznih fazah Kruskalovega postopka

5.4.2 Primov algoritem

Za programiranje na računalniku je Primov algoritem primernejši od Kruskalovega, saj se pri slednjem izvaja potratno urejanje po naraščajočih utežeh ter nenehno odkrivanje morebitnih ciklov.

Primov algoritem se glasi [16]:

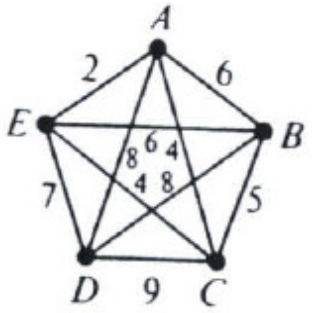
Najmanjše vpeto drevo T povezanega uteženega grafa G konstruiramo s postopkom:

1. *Dodaj poljubno vozlišče v prazno drevo T.*
2. *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 v tem, da lahko deluje neposredno na tabeli uteži namesto na grafu [16]. 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 stolpcih, ki ustrezajo že postavljenim vozliščem v *T*. Metodo ponazorimo z naslednjim primerom [16].

Primer:

S pomočjo Primovega algoritma poišči minimalno vpeto drevo za graf na sliki 76.



Slika 76.: Graf, pri katerem bomo s Primovim algoritmom poiskali minimalno vpeto drevo.

Najprej grafu na sliki 76 priredimo tabelo, ki je prikazana na sliki 77.

	A	B	C	D	E
A	–	6	4	8	2
B	6	–	5	8	6
C	4	5	–	9	4
D	8	8	9	–	7
E	2	6	4	7	–

Slika 77: Tabela uteži, ki jo priredimo grafu

Kot je razvidno iz tabele 77, smo na njej vpisali uteži, s katerimi so povezana posamezna vozlišča grafa na sliki 76. Če določeni vozlišči nista povezani (ali pri primerjavi vozlišča s samim seboj), pa smo to ustrezno označili z znakom '-'.

Najprej izberemo katerokoli vozlišče grafa na sliki 76, denimo je to vozlišče B, ter ga dodamo v prazen graf T. Le-ta preide v obliko, ki jo prikazuje slika 78.

Slika 78: Graf T, sestavljen iz enega samega vozlišča B.

Nato zbrišemo vrstico B iz tabele uteži na sliki 77, pri čemer dobimo tabelo na sliki 79.

	A	B	C	D	E
A	–	6	4	8	2
C	4	5	–	9	4
D	8	8	9	–	7
E	2	6	4	7	–

Slika 79: Tabela uteži, kjer smo zbrisali vrstico B.

Nato v stolpcu vozlišča B, ki je že bilo uporabljeno v T, poiščemo najmanjšo utež. Vidimo, da je to utež povezave med vozliščema B in C z vrednostjo 5. To povezavo dodamo v graf T na sliki 78, pri čemer dobimo sliko 80.

Slika 80: Graf T, sestavljen iz dveh vozlišč B in C.

Nato zbrišemo vrstico C iz tabele uteži na sliki 79, pri čemer dobimo tabelo na sliki 81.

	A	B	C	D	E
A	–	6	(4)	8	2
D	8	8	9	–	7
E	2	6	(4)	7	–

Slika 81: Tabela uteži, kjer smo zbrisali vrstici B in C.

Nato v stolpcih vozlišč B in C, ki sta že bili uporabljeni v T, poiščemo najmanjšo utež. Očitno gre za uteži povezav med vozliščema A in C, ter E in C, z vrednostjo 4. Izberemo eno izmed teh dveh povezav, npr. povezavo CA, ter jo dodamo v graf T na sliki 80, pri čemer dobimo sliko 82.

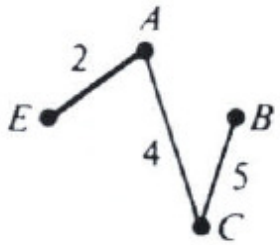
Slika 82: Graf T, sestavljen iz treh vozlišč A, B in C.

Nato zbrišemo vrstico A iz tabele uteži na sliki 81, pri čemer dobimo tabelo na sliki 83.

	A	B	C	D	E
D	8	8	9	–	7
E	(2)	6	4	7	–

Slika 83: Tabela uteži, kjer smo zbrisali vrstice A, B in C.

Nato v stolpcih vozlišč A, B in C, ki so že bili uporabljena v T, poiščemo najmanjšo utež. Očitno gre za utež povezave med vozliščema A in E z vrednostjo 2. To povezavo dodamo v graf T na sliki 82, pri čemer dobimo sliko 84.



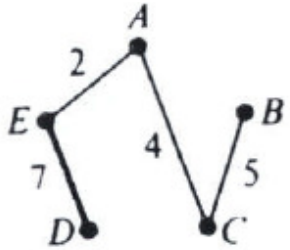
Slika 84: Graf T, sestavljen iz štirih vozlišč A, B, C in E.

Nato zbrišemo vrstico E iz tabele uteži na sliki 83, pri čemer dobimo tabelo na sliki 85.

	A	B	C	D	E
D	8	8	9	–	⑦

Slika 85: Tabela uteži, kjer smo zbrisali vrstice A, B, C in E.

Nato v stolpcih vozlišč A, B, C in E, ki so že bili uporabljena v T, poiščemo najmanjšo utež. Očitno gre za utež povezave med vozliščema D in E z vrednostjo 7. To povezavo dodamo v graf T na sliki 84, pri čemer dobimo sliko 86.



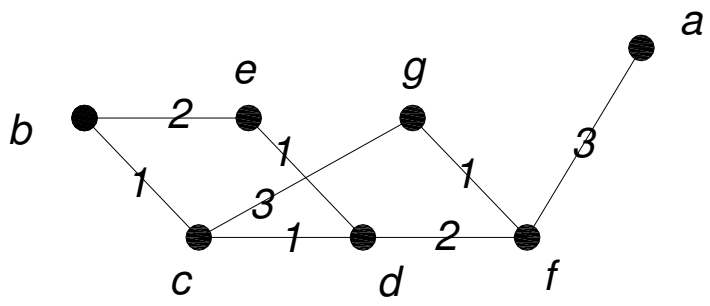
Slika 86: Graf T, sestavljen iz petih vozlišč A, B, C, D in E.

Ker smo uspeli v grafu T na sliki 86 povezati vsa vozlišča originalnega grafa na sliki 76, se je postopek s Primovim algoritmom očitno končal, rezultat na sliki 86 pa je že minimalno vpeto drevo. Skupna utež tega drevesa ima, kot lahko vidimo, vrednost 18.

Uporabo Primovega algoritma za iskanje minimalno vpetega drevesa na kratko ponazorimo še na enem primeru.

Primer:

S pomočjo Primovega algoritma poišči minimalno vpeto drevo za graf na sliki 87.



Slika 87.: Graf, pri katerem bomo s Primovim algoritmom poiskali minimalno vpeto drevo.

Najprej grafu na sliki 87 priredimo tabelo, ki je prikazana na sliki 88.

	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

Slika 88.: Tabela uteži, ki jo priredimo grafu

Kot je razvidno iz tabele na sliki 88, smo na njej vpisali uteži, s katerimi so povezana posamezna vozlišča grafa na sliki 87. Če določeni vozlišči nista povezani, smo to ustrezno označili z znakom '-'. Pri oddaljenosti vozlišč od samih seboj pa smo zapisali vrednost 0.

Najprej izberemo katerokoli vozlišče grafa na sliki 87, denimo je to vozlišče a, ter ga dodamo v prazen graf T. Zato zbrišemo vrstico a iz tabele uteži na sliki 88, pri čemer dobimo tabelo na sliki 89. Označimo tudi stolpec a, odkoder bomo poiskali minimalen

element. Očitno ima slednji vrednost 3 in se nahaja v vrstici vozlišča *f*.

	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	3	-	-	2	-	0	1
g	-	-	3	-	-	1	0

Slika 89.: Tabela uteži, kjer smo zbrisali vrstico a.

To je utež povezave med vozliščema a in f, ki jo dodamo v graf T, ki sedaj vsebuje vozlišči a in f.

Nato zbrišemo vrstico f iz tabele uteži na sliki 89, pri čemer dobimo tabelo na sliki 90. Označimo tudi stolpec f, odkoder bomo poleg stolpca a iskali minimalen element.

	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	-	-
g	-	-	3	-	-	1	0

Slika 90.: Tabela uteži, kjer smo zbrisali vrstici a in f.

Očitno ima minimalni element v stolpcih a in f vrednost 1 in se nahaja v vrstici vozlišča *g*. Zato v drevo T dodamo vozlišče *g* in povezavo *gf* (dolžine 1), ki sedaj vsebuje vozlišča a, f in g. Iz tabele na sliki 90 pa zbrišemo vrstico *g* in označimo stolpec vozlišča *g,* odkoder bomo poleg stolpcev a in f iskali nov minimalen element. Tako dobimo tabelo na sliki 91.

	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	-	-

Slika 91.: Tabela uteži, kjer smo zbrisali vrstice a, f in g.

Očitno ima minimalni element v stolpcih a, f in g vrednost 2 in se nahaja v vrstici vozlišča *d.* Zato v drevo T dodamo vozlišče *d* in povezavo *df* (dolžine 2), ki sedaj vsebuje vozlišča a, f, g in d. Iz tabele na sliki 91 pa zbrišemo vrstico *d* in označimo stolpec vozlišča *d,* odkoder bomo poleg stolpcev a, f in g iskali nov minimalen element. Tako dobimo tabelo na sliki 92.

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

Slika 92.: Tabela uteži, kjer smo zbrisali vrstice a, f, g in d.

Očitno ima minimalni element v stolpcih a, f, g in d vrednost 1 in se nahaja v vrsticah vozlišč *c* in *e.* Gre za dve enakovredni varianti, izberemo si eno, npr. tisto v vrstici c. Zato v drevo T dodamo vozlišče *c* in povezavo *dc* (dolžine 1), ki sedaj vsebuje vozlišča a, f, g, d in c. Iz tabele na sliki 92 pa zbrišemo vrstico *c* in označimo stolpec vozlišča *c,* odkoder bomo poleg stolpcev a, f, g in d iskali nov minimalen element. Tako dobimo tabelo na sliki 93.

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

Slika 93.: Tabela uteži, kjer smo zbrisali vrstice a, f, g, d in c.

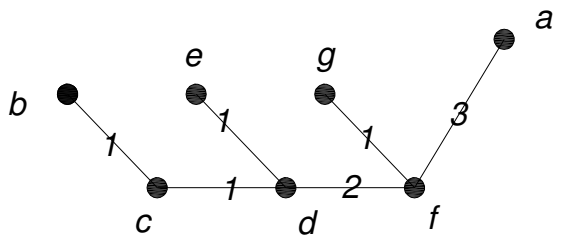
Očitno ima minimalni element v stolpcih a, f, g, d in c vrednost 1 in se nahaja v vrsticah vozlišč *b* in *e.* Zopet gre za dve enakovredni varianti, izberemo si eno, npr. tisto v vrstici b. Zato v drevo T dodamo vozlišče *b* in povezavo *bc* (dolžine 1), ki sedaj vsebuje

vozlišča a, f, g, d, c in b. Iz tabele na sliki 93 pa zbrišemo vrstico *b* in označimo stolpec vozlišča *b,* odkoder bomo poleg stolpcev a, f, g, d in c iskali nov minimalen element. Tako dobimo tabelo na sliki 94.

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

Slika 94.: Tabela uteži, kjer smo zbrisali vrstice a, f, g, d, c in b.

Očitno ima minimalni element v stolpcih a, f, g, d, c in b vrednost 1 in se nahaja v vrsticah edinega preostalega vozlišča *e.* Zato v drevo T dodamo še vozlišče *e* in povezavo *ed* (dolžine 1), ter končamo postopek. Dobljeno minimalno vpeto drevo (v grafu T!) je prikazano na sliki 95 in ima skupno utež 9.



Slika 95.: Minimalno vpeto drevo, dobljeno s Primovim algoritmom

5.5 Iskanje spodnje in zgornje meje uteži minimalnega Hamiltonovega cikla pri problemu trgovskega potnika

Kot smo videli pri problemu trgovskega potnika, slednji želi obiskati nekaj mest in se vrniti domov, pri čemer naj prevozi najmanjšo možno razdaljo. Glede na preprostost požrešne metode za problem iskanja najmanjšega vpetega drevesa bi upali, da bo obstajal preprost algoritem tudi za reševanje problema trgovskega potnika. Na žalost ne poznamo nobenega takšnega algoritma. Seveda bi lahko poskusili preveriti vse mogoče Hamiltonove cikle in izbrati tistega z najmanjšo dolžino. Toda to je brezupna naloga tudi za računalnik, razen če je graf zelo majhen. Za problem razporeditve poslov z (recimo) 100 posli bi imeli že l00! ($\sim 9.3 \cdot 10^{157}$) zaporedij, ki bi jih bilo potrebno pregledati, zato ni vredno niti poskusiti z nobeno metodo, ki bi temeljila na takšnem pregledovanju. Zato smo prisiljeni iskati približne rešitve problema. Metoda, ki v praksi pogostokrat zadovoljivo dobro deluje, je poiskati spodnjo in zgornjo mejo za problem trgovskega potnika, tako, da namesto tega rešimo problem minimalnega vpetega drevesa [16, 19].

5.5.1 Iskanje spodnje meje uteži minimalnega Hamiltonovega cikla pri problemu trgovskega potnika

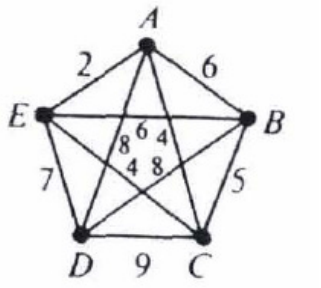
Metodo opravičimo z naslednjim argumentom. Če vzamemo Hamiltonov cikel z najmanjšo utežjo v uteženem polnem grafu in odstranimo vozlišče A in povezave s krajišči v A, dobimo pot, ki gre skozi ostale točke. Takšna pot je vpeto drevo za polni graf na preostalih točkah, uteži Hamiltonovega cikla pa dobimo, če dodamo skupni teži poti uteži odstranjenih dveh povezav.

Izrek:

Spodnjo mejo za rešitev problema trgovskega potnika torej dobimo, če seštejemo utež najmanjšega vpetega drevesa in dve najmanjši uteži povezav s krajiščem v odstranjenem vozlišču A [16, 19].

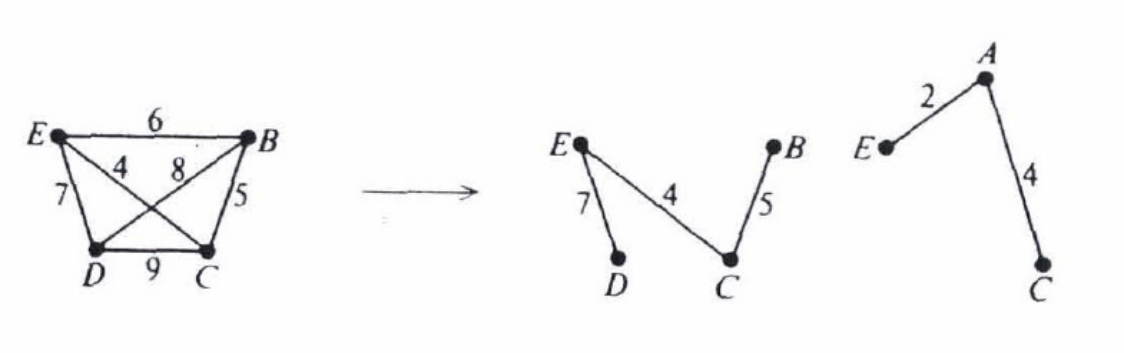
Primer:

Ocenite spodnjo mejo uteži minimalnega Hamiltonovega cikla pri problemu trgovskega potnika za graf na sliki 96.



Slika 96.: Graf, pri katerem bomo ocenili spodnjo mejo uteži minimalnega Hamiltonovega cikla

Če iz grafa na sliki 96 odstranimo vozlišče A, dobimo preostanek grafa z vozlišči B, C, D in E (slika 97 levo).

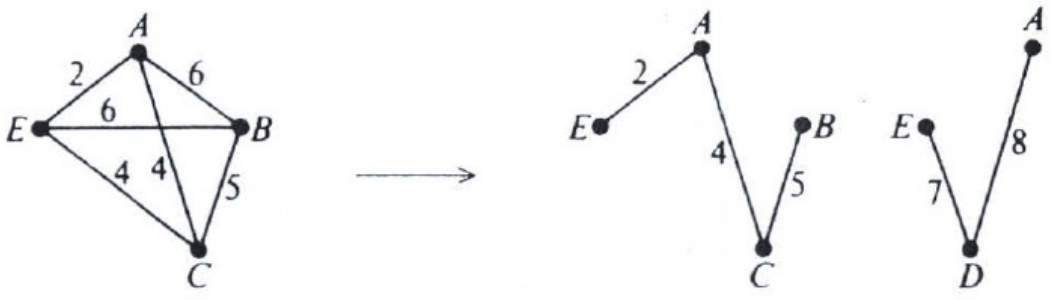


Slika 97.: Ocenjevanje spodnje meje uteži minimalnega Hamiltonovega cikla (odstranjeno vozlišče A)

Če nato za ta preostanek grafa poiščemo minimalno vpeto drevo, dobimo rezultat na sliki 97 v sredini, ki ima skupno utež 16. Poglejmo še, s kakšnimi utežmi je bilo odstranjeno vozlišče A na grafu na sliki 96 povezano z ostalimi vozlišči. Očitno sta imeli povezavi z vozliščema E in C najmanjšo utež, to je vrednost 2 oz. 4 (glej sliko 97 desno).

Iskana spodnja meja skupne uteži minimalnega Hamiltonovega cikla (pri odstranjenem A) je torej:
utež najmanjšega vpetega drevesa + dve najmanjši uteži povezav s krajiščem v odstranjenem vozlišču A = 16 + 2 + 4 = 22.

Kot se izkaže, dobimo še boljši rezultat za spodnjo mejo, če iz grafa na sliki 96 odstranimo vozlišče D. Če to storimo, dobimo preostanek grafa z vozlišči A, B, C, in E (slika 98 levo).



Slika 98.: Ocenjevanje spodnje meje uteži minimalnega Hamiltonovega cikla (odstranjeno vozlišče D)

Če nato za ta preostanek grafa poiščemo minimalno vpeto drevo, dobimo rezultat na sliki 98 v sredini, ki ima skupno utež 11. Poglejmo še, s kakšnimi utežmi je bilo odstranjeno vozlišče D na grafu na sliki 96 povezano z ostalimi vozlišči. Očitno sta imeli povezavi z vozliščem E in vozliščema A ali B najmanjšo utež, to je vrednost 7 oz. 8. Denimo izberemo od slednjih dveh vozlišče A (glej sliko 98 desno), pri čemer dobimo vsoto dveh najmanjših uteži povezav s krajiščem v odstranjenem vozlišču D enako: 7+8 = 15.

Iskana spodnja meja skupne uteži minimalnega Hamiltonovega cikla (pri odstranjenem D) je torej:
utež najmanjšega vpetega drevesa + dve najmanjši uteži povezav s krajiščem v odstranjenem vozlišču D = 11 + 15 = 26.

Primerjajmo sedaj ta rezultat s pravo (optimalno) rešitvijo za problem trgovskega potnika na sliki 96. Kot se izkaže, je slednja enaka minimalnemu Hamiltonovemu ciklu A-C-B-D-E-A s skupno težo 4+5+8+7+2 = 26. Torej smo v tem primeru z ocenjevanjem spodnje meje celo zadeli pravi rezultat za skupno težo minimalnega Hamiltonovega cikla.

5.5.2 Iskanje zgornje meje uteži minimalnega Hamiltonovega cikla pri problemu trgovskega potnika

Če izvedemo iskanje v globino na najmanjšem vpetem drevesu danega grafa, tako, da dobimo sklenjen sprehod, ki obišče vsako vozlišče vsaj enkrat, prehodimo vsako povezavo natanko dvakrat in prehojena pot je enaka dvojni skupni teži najmanjšega vpetega drevesa. Dokazati se da naslednje [16 ,19]:
Vrednost rešitve problema trgovskega potnika je največ dvakrat večja od skupne teže najmanjšega vpetega drevesa.

V prvem primeru uporabe Primovega algoritma v poglavju 5.4.2. smo videli, da ima minimalno vpeto drevo za graf na sliki 96 skupno težo 18. Ocenjena zgornja meja za rešitev problema trgovskega potnika se torej glasi: Zgornja meja = 2 x 18 = 36.

Če povzamemo sklepe poglavij o iskanju spodnje in zgornje meje uteži minimalnega Hamiltonovega cikla pri problemu trgovskega potnika, vsekakor velja naslednje. Ocenjeni meji nam lahko služita kot dobrodošla dodatna orientacija, kako čim enostavneje poiskati takšno rešitev za Hamiltonov cikel, ki se čimbolj približa optimalni rešitvi (minimalni Hamiltonov cikel).

6 PROBLEM NAJKRAJŠE POTI

Problemi najkrajše poti so eni najpomembnejših problemov, ki jih poskušamo reševati v okviru teorije grafov in mrežnih pretokov. Pojavljajo se pri širokem spektru aplikacij, kjer želimo poslati določen material (računalniški paket podatkov, telefonski klic, vozilo, itn) med dvema konkretnima točkama v dani mreži. Velikokrat pa se pojavijo tudi kot podproblemi pri reševanju številnih problemov kombinatorične oz. mrežne optimizacije.

Obstaja več tipov problemov najkrajše poti, kot npr. [15]:

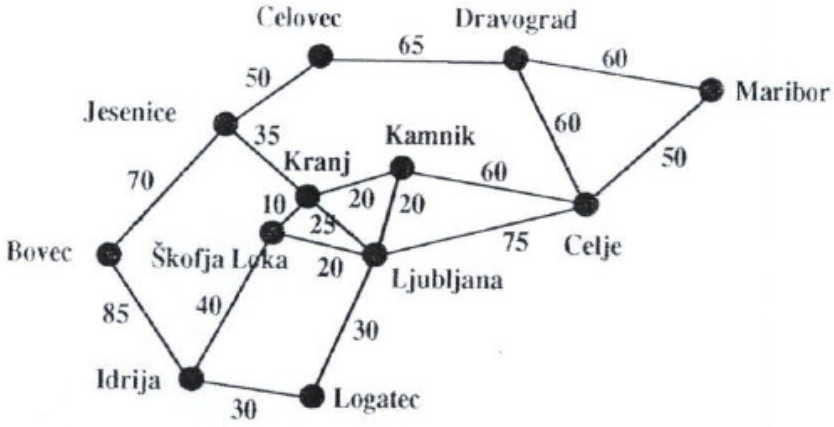
- iskanje najkrajše poti od enega (začetnega oz. izvornega) vozlišča do drugega (končnega oz. ciljnega) vozlišča.
- Iskanje najkrajše poti od enega vozlišča do vseh ostalih vozlišč,
- Iskanje najkrajše poti od vsakega vozlišča do vsakega drugega vozlišča, itn.

Tipični primeri aplikacij, kjer se pojavljajo problemi najkrajše poti, pa so [15]:

- Iskanje najkrajše (najcenejše, najhitrejše) poti vozila med dvema geografskima lokacijama,
- Kompleksni problemi urbanega prometnega planiranja, kjer želimo načrtovati konstrukcijo najkrajših prometnih poti med izvornimi in ciljnimi lokacijami,
- Aplikacije v domeni planiranja proizvodnje, razvrščanja telefonskih klicev, itn.

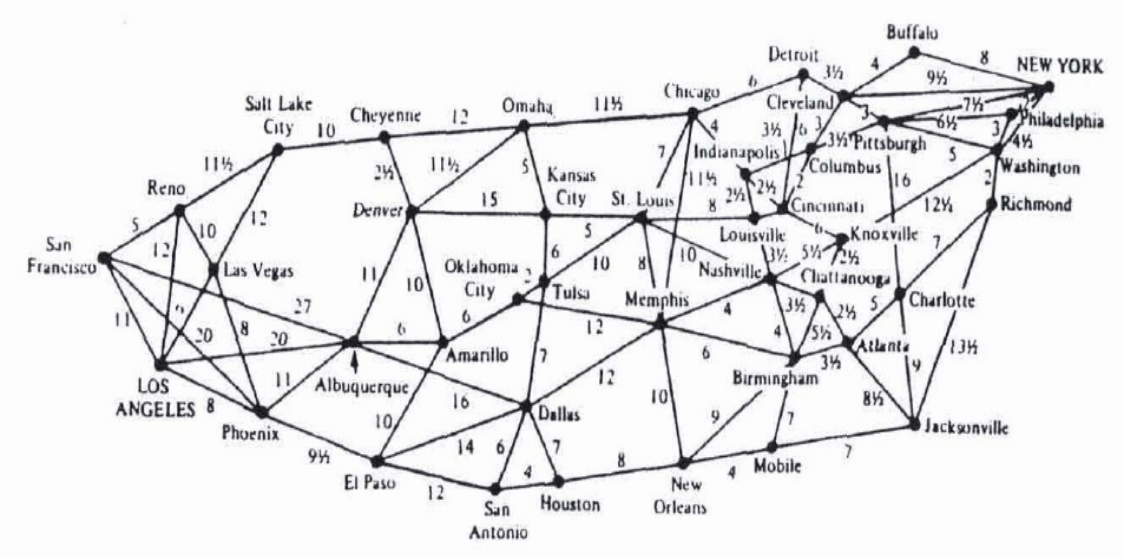
1. Primer problema najkrajše poti:

Popotnik se želi peljati iz Maribora v Bovec v najkrajšem možnem času. Na karti na sliki 99 je označen čas (v minutah), potreben za potovanje med posameznimi pari mest. Katero pot naj izbere popotnik?



Slika 99: Problem iskanja najkrajše poti med mestoma Maribor in Bovec

V tem primeru ni zelo težko najti rešitve z nekaj spretnega ugibanja, toda takšen pristop je mnogo manj uspešen v primerih, ko postane mreža cest bolj in bolj zapletena (glej primer poti med mestoma Los Angeles in New York na sliki 100).

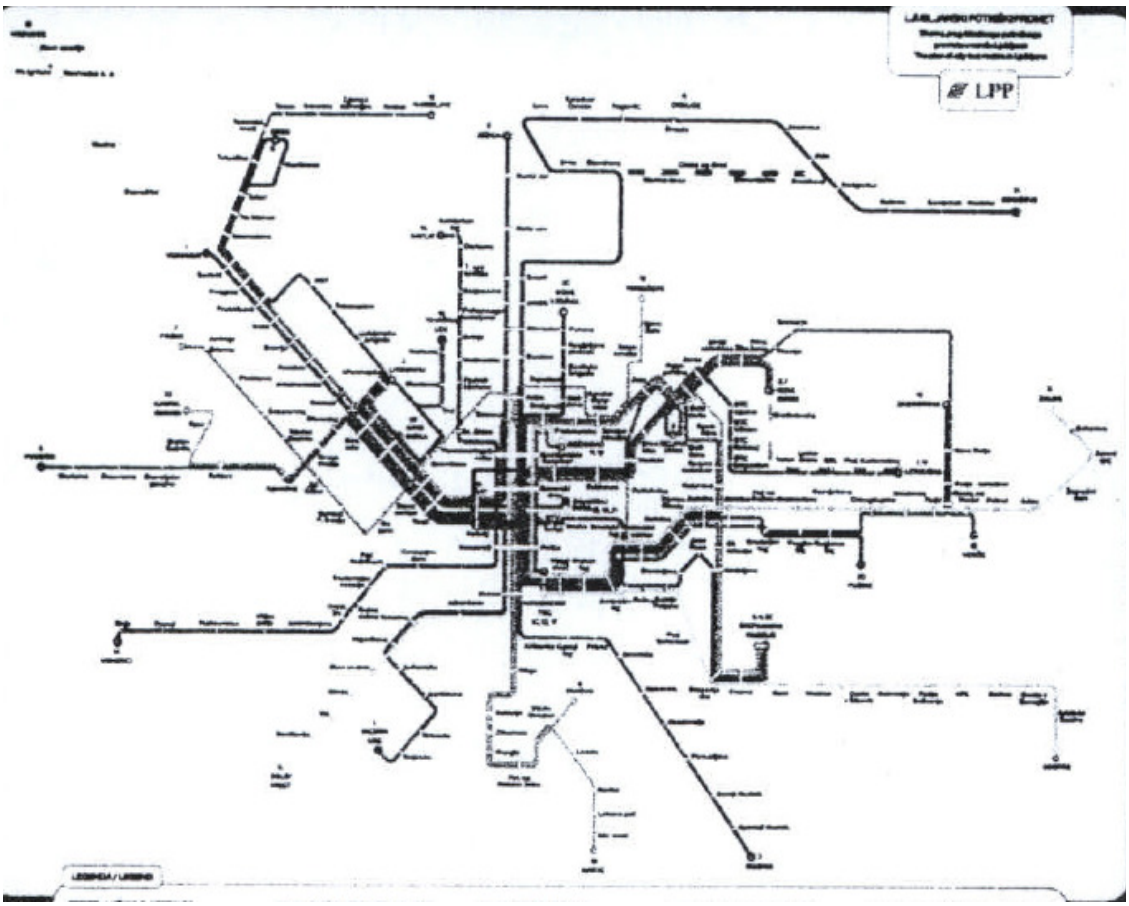


Slika 100: Problem iskanja najkrajše poti med mestoma Los Angeles in New York

2. Primer problema najkrajše poti:

Na sliki 101 je podana shema prog mestnega prometa v Ljubljani [19]. Če nas zanimajo najkrajše poti med deli mesta (med postajami mestnih avtobusov), lahko na osnovi sheme definiramo graf z utežmi na povezavah. Utež lahko smiselno določimo na več načinov, odvisno od tega, v kakšnem smislu nas najkrajše poti zanimajo. Na primer, lahko se

vprašamo, kolikokrat je potrebno zamenjati avtobus, da pridemo iz Trnovega do tovarne Lek. Drugo najkrajšo pot (verjetno) dobimo, če nas zanima minimalno število vmesnih postajališč na poti.



Slika 101: Shema prog mestnega prometa v Ljubljani in problem najkrajše poti

V nadaljevanju bomo opisali algoritem, ki ga lahko učinkovito uporabimo za iskanje najkrajše poti med katerimkoli parom vozlišč v grafu [15, 16, 19].

6.1 Dijkstrin algoritem za iskanje najkrajših poti

Dijkstrin algoritem je bil odkrit s strani matematika Edsgerja Dijkstre leta 1956 in objavljen v članku leta 1959. Algoritem je namenjen za reševanje enoizvorskih problemov najkrajše poti za grafe z nenegativnimi utežmi na povezavah, pri čemer poišče

najkrajšo pot (oz. drevo najkrajših poti). Ta algoritem je sicer pogostokrat uporabljan tudi pri problematiki usmerjanja v mreži.

Algoritem lahko poišče najkrajšo pot od enega vozlišča do vseh ostalih vozlišč. Seveda pa je primeren tudi za iskanje najkrajše poti od enega (izvornega) vozlišča do drugega (ciljnega) vozlišča. Tako algoritem v primeru, če imamo npr. graf, kjer so vozlišča mesta, povezave pa poti med njimi, poišče najkrajšo pot med enim mestom in vsemi ostalimi mesti.

6.1.1 Princip delovanja Dijkstrinega algoritma

Denimo je naloga tega algoritma poiskati najkrajšo pot od vozlišča S do vozlišča T v danem omrežju [15]. V ta namen se premikamo po omrežju z leve proti desni in računamo razdalje od S do vsakega od vmesnih vozlišč, ki jih obiščemo. Na vsakem koraku algoritma pogledamo vsa vozlišča, ki jih lahko dosežemo preko usmerjene povezave iz vozlišča, v katerem trenutno smo, in jim priredimo začasno razdaljo, ki je najkrajša razdalja od S do pa do teh vozlišč po poteh, ki smo jih obravnavali do tedaj. Končno vsako vozlišče dobi stalno oznako (poimenovano potencial, običajno označeno v kvadratnem okvirčku), ki je gotovo najkrajša razdalja od S do dotičnega vozlišča. Ko tudi vozlišče T dobi potencial, smo določili najkrajšo razdaljo med vozliščema S in T.

Med izvajanjem postopka je lahko vozlišče v enem izmed treh stanj [15, 16]:

- novo vozlišče (ki ga z algoritmom še nismo obravnavali),
- znano vozlišče (ki ni več novo), ter
- obdelano vozlišče (z dokončno izračunano razdaljo).

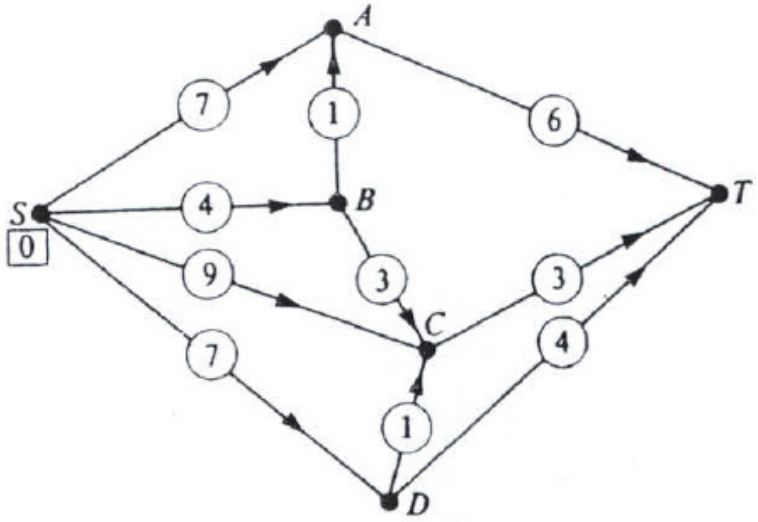
Ko novo vozlišče v algoritmu prvič srečamo kot soseda obravnavanega vozlišča, ga proglasimo za znanega in mu dodelimo začasno oznako, dotlej najkrajšo razdaljo med njim in izvornim vozliščem. Iz množice znanih vozlišč nato vedno izberemo vozlišče z najmanjšo začasno oznako in to oznako proglasimo za razdaljo, poimenovano potencial. Vsem sosedom tega vozlišča nato določimo nove oznake tako, da razdaljo, ki je bila

izračunana z doslej pregledanimi potmi, popravimo, če smo našli krajšo pot (glede na izvorno vozlišče).

Delovanje algoritma, ki ga lahko uporabljamo tako za grafe kot digrafe, bomo osvetili na naslednjem primeru digrafa [15, 16].

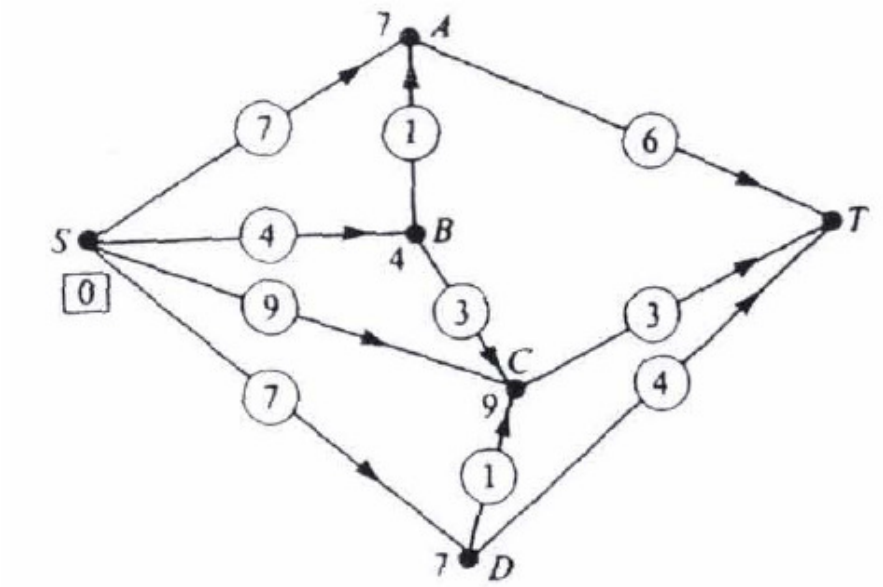
Primer:

Za dani digraf na sliki 102 poišči najkrajšo pot med izvornim vozliščem S in ciljnim vozliščem T z uporabo Dijkstrinega algoritma.



Slika 102: Digraf, kjer bomo z Dijkstrinim algoritmom poiskali najkrajšo pot med vozliščema S in T.

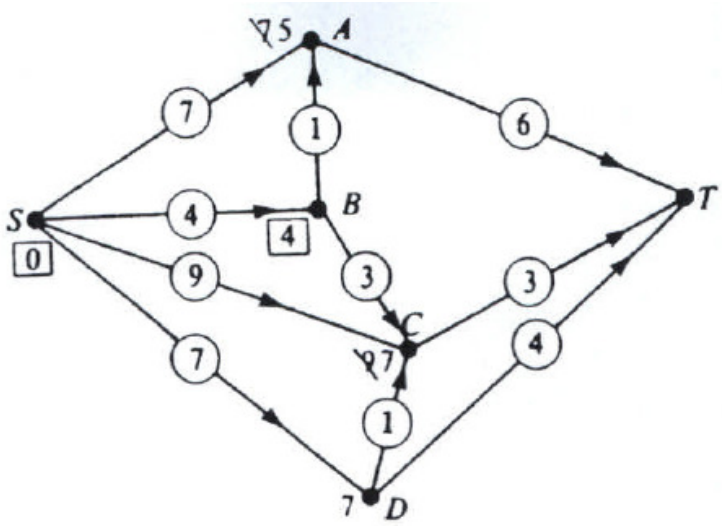
Na začetku priredimo vozlišču S potencial 0, saj je najkrajša razdalja od S do S enaka 0. Potem gledamo vsa vozlišča, ki jih lahko dosežemo iz S (to so vozlišča A, B, C in D), in vsakemu od njih določimo začasno oznako. Ta je vsota potenciala v S in razdalje od S do dotičnega vozlišča. Tako dobimo začasne razdalje 7, 4, 9, in 7 do vozlišč A, B, C in D (glej sliko 103).



Slika 103: Dodelitev začasnih razdalj 7, 4, 9, in 7 vozliščem A, B, C in D

Zdaj vzamemo najmanjšo začasno razdaljo (najmanjšo oznako, ki še ni potencial) in jo proglasimo za potencial. V tem primeru je takšna razdalja v vozlišču B, zato temu vozlišču določimo potencial 4 (označimo 4 v kvadratnem okvirčku, glej na sliki 104). To je res najkrajša razdalja od S do B, saj bi vsaka druga pot od S do B morala preko enega od vozlišč A, C ali D in je do tja dolžina poti gotovo večja od 4.

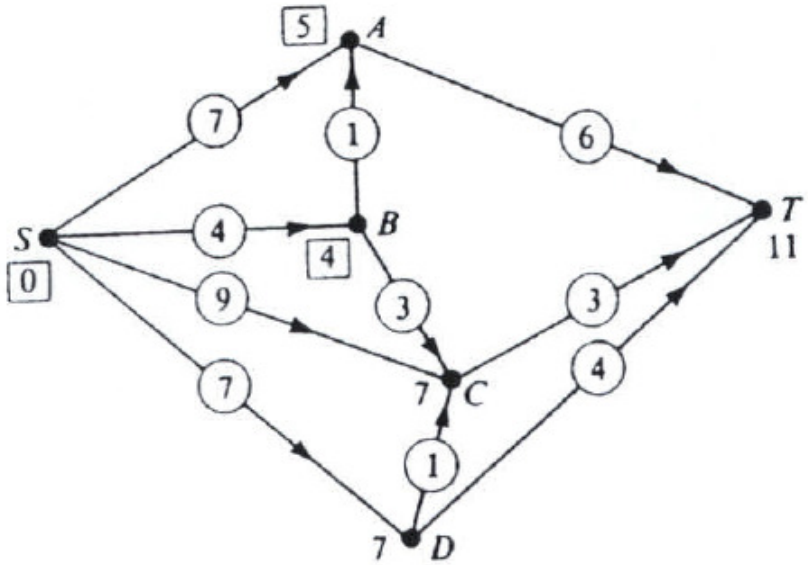
Ker smo v B ravnokar priredili potencial, zdaj gledamo vsa vozlišča, ki jih lahko dosežemo neposredno iz B (to sta vozlišči A in C). Vsakemu od teh vozlišč priredimo novo začasno razdaljo, enako vsoti potenciala v B in razdalje od B do teh vozlišč, vendar samo v primeru, če je nova začasna razdalja manjša od oznake, ki sta jo vozlišči že imeli od prej. Ker to velja tako za vozlišče A, kot C, slednjima priredimo novi začasni oznaki 4 + 1 = 5 oz. 4 + 3 = 7, saj sta novi vrednosti gotovo manjši od prejšnjih začasnih oznak. Prireditev novih začasnih oznak je na sliki 104 označena tako, da prečrtamo vrednosti 7 in 9, ter ju nadomestimo z vrednostima 5 oz. 7.



Slika 104: Dodelitev potenciala 4 vozlišču B, dodelitev novih začasnih oznak 5 in 7 vozliščema A in C.

Zdaj vzamemo na sliki 104 najmanjšo začasno oznako in jo proglasimo za potencial. V tem primeru je takšna razdalja v vozlišču A, zato temu vozlišču določimo potencial 5 (označimo 5 v kvadratnem okvirčku, glej na sliki 105).

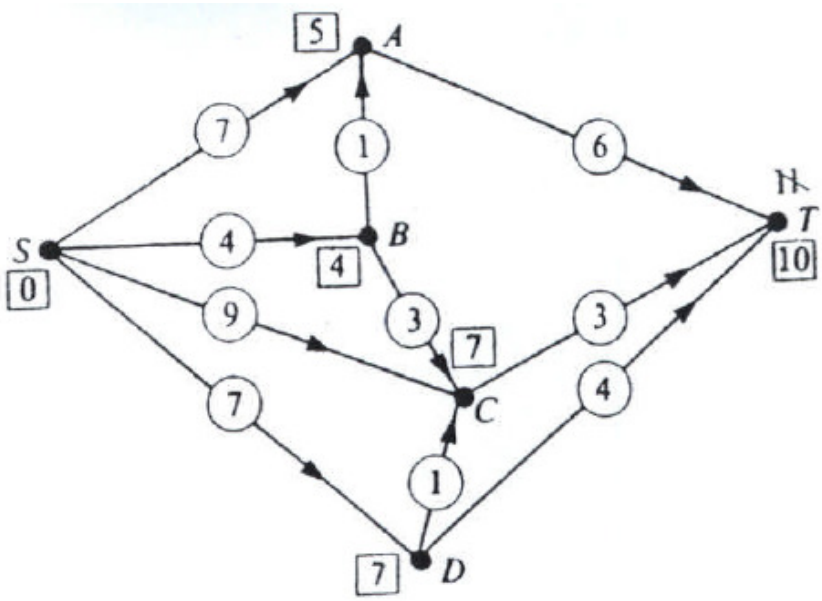
Ker smo v A ravnokar priredili potencial, zdaj gledamo vsa vozlišča, ki jih lahko dosežemo neposredno iz A (to je le vozlišče T). Temu vozlišču priredimo začasno razdaljo, enako vsoti potenciala v A in razdalje od A do T, pri čemer dobimo začasno oznako 5+6 = 11 (glej sliko 105).



Slika 105: Dodelitev potenciala 5 vozlišču A in dodelitev začasne oznake 11 vozlišču T.

Zdaj vzamemo na sliki 105 najmanjšo začasno oznako in jo proglasimo za potencial. V tem primeru sta takšni razdalji v vozliščih C in D, zato tema vozliščema določimo potencial 7 (označimo 7 v kvadratnem okvirčku, glej na sliki 106).

Ker smo v C in D ravnokar priredili potencial, zdaj gledamo vsa vozlišča, ki jih lahko dosežemo neposredno iz C in D (to je le vozlišče T). Vozlišče T ima začasno oznako 11, premik iz vozlišča C v T bi dal slednjemu oznako 7+3 = 10, premik iz vozlišča D v T pa bi dal slednjemu oznako 7+4 = 11. Seveda izberemo premik iz vozlišča C v T, saj da slednjemu najmanjšo začasno oznako 10. Prireditev nove začasne oznake 10 vozlišču T je na sliki 106 označena tako, da prečrtamo vrednosti 11, ter jo nadomestimo z vrednostjo 10. Slednja je tudi potencial v točki T (okvirček na sliki 106), saj smo prišli s postopkom do konca. Ker ima vozlišče T potencial 10, je to očitno tudi najkrajša razdalja med vozliščema S in T.



Slika 106: Dodelitev potencialov 7 vozliščema C in D in dodelitev potenciala 10 vozlišču T.

Strukturo najkrajše poti med S in T pa določimo tako, da potujemo iz vozlišča T nazaj proti vozlišču S. Pri tem upoštevamo le tiste povezave oz. segmente poti, kjer je razlika višjega in nižjega potenciala med določenima vozliščema enaka razdalji med tema vozliščema.

Očitno velja to za naslednje povezave (glej sliko 106):

$$\begin{aligned} potencial\ pri\ T\ - potencial\ pri\ C\ &= razdalja\ od\ C\ do\ T \\ potencial\ pri\ C\ - potencial\ pri\ B\ &= razdalja\ od\ B\ do\ C \\ potencial\ pri\ B\ - potencial\ pri\ S\ &= razdalja\ od\ S\ do\ B \end{aligned}$$

Torej bodo te povezave gotovo sestavni del najkrajše poti med S in T, ki se potemtakem glasi: S-B-C-T v skupni dolžini 10.

Povzemimo opisani postopek Dijkstrinega algoritma [15, 16, 19].

Dijkstrin algoritem za iskanje najkrajše poti

Iščemo najkrajšo pot med vozliščema S in T v danem omrežju.

KORAK 1: Določi točki S potencial 0. Določi vsakemu vozlišču V, ki je dosegljivo neposredno iz S, oznako, enako razdalji med S in V. Izberi najmanjšo od teh oznak in jo proglasi za potencial.

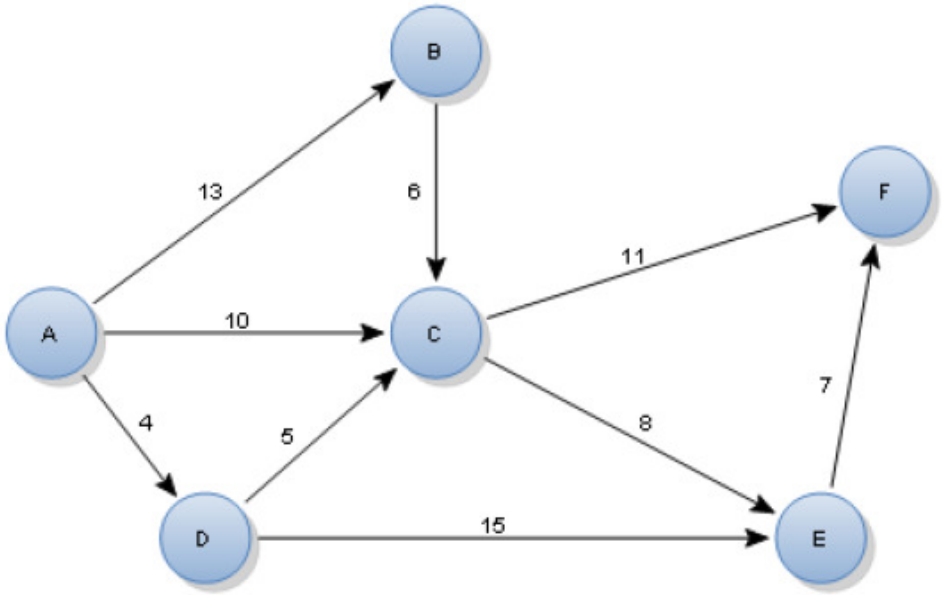
PONAVLJAJ: Obravnavaj vozlišča V, ki jim je bil pravkar določen potencial. Za vsakega od vozlišč V poglej vsakega od vozlišč W, ki jih je mogoče doseči neposredno iz V in določi vozliščem W začasno oznako: (potencial pri V) + (razdalja od V do W), razen, če vozlišče W ni že imelo manjše oznake. Ko so bila vsa vozlišča W označena, izberi najmanjše oznake in jih proglasi za potenciale. Ponovi z novimi potenciali.

KONČAJ, ko tudi vozlišče T dobi potencial. To je najkrajša razdalja od S do T.

DOLOČITEV NAJKRAJŠE POTI: Pojdi od T nazaj proti S in izberi tiste povezave V-W, za katere velja: potencial pri W − potencial pri V = razdalja od W do V

Primer:

Za dani digraf na sliki 107 poišči najkrajšo pot med izvornim vozliščem A in ciljnim vozliščem F z uporabo Dijkstrinega algoritma.



Slika 107: Digraf, kjer bomo z Dijkstrinim algoritmom poiskali najkrajšo pot med vozliščema A in F.

Na začetku priredimo vozlišču A potencial 0, saj je najkrajša razdalja od A do A enaka 0. Potem gledamo vsa vozlišča, ki jih lahko dosežemo iz A (to so vozlišča B, C in D), in vsakemu od njih določimo začasno oznako. Ta je vsota potenciala v A in razdalje od A do dotičnega vozlišča. Tako dobimo začasne razdalje 13, 10 in 4 do vozlišč B, C in D.

Zdaj vzamemo najmanjšo začasno razdaljo (najmanjšo oznako, ki še ni potencial) in jo proglasimo za potencial. V tem primeru je takšna razdalja v vozlišču D, zato temu vozlišču določimo potencial 4.

Ker smo v D ravnokar priredili potencial, zdaj gledamo vsa vozlišča, ki jih lahko dosežemo neposredno iz D (to sta vozlišči C in E). Vsakemu od teh vozlišč priredimo (novo) začasno razdaljo, vozlišču C vrednost 9 (namesto 10), vozlišču E pa vrednost 19.

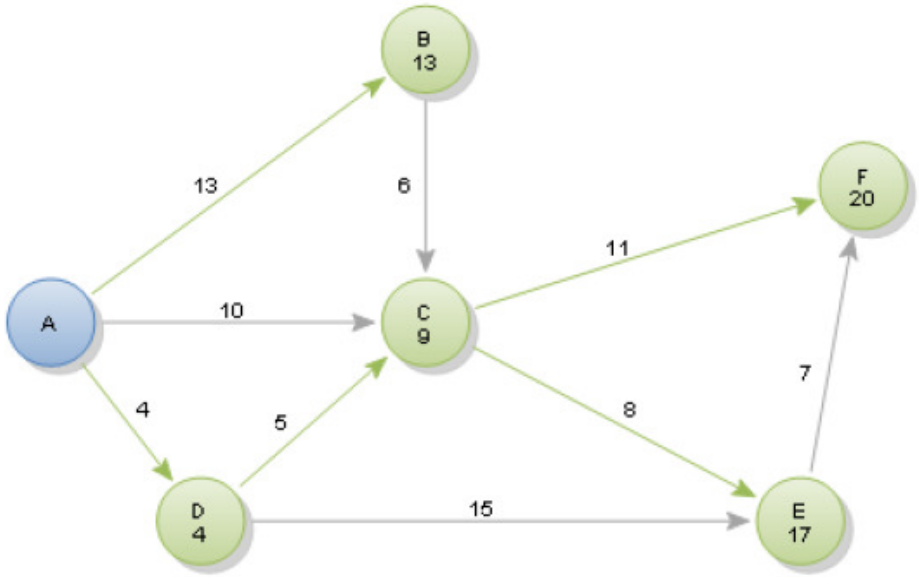
Zdaj vzamemo najmanjšo začasno oznako in jo proglasimo za potencial. V tem primeru je takšna razdalja v vozlišču C, zato temu vozlišču določimo potencial 9.

Ker smo v C ravnokar priredili potencial, zdaj gledamo vsa vozlišča, ki jih lahko dosežemo neposredno iz C (to sta vozlišči E in F). Vsakemu od teh vozlišč priredimo (novo) začasno razdaljo, vozlišču E vrednost 17 (namesto 19), vozlišču F pa vrednost 20.

Zdaj vzamemo najmanjšo začasno oznako in jo proglasimo za potencial. V tem primeru je takšna razdalja v vozlišču B, zato temu vozlišču določimo potencial 13. Ker smo v B ravnokar priredili potencial, zdaj gledamo vsa vozlišča, ki jih lahko dosežemo neposredno iz B (to je vozlišče C). Ker pa ima slednje že dodeljen potencial (9), oznaka iz B (13+6 = 19) ne more prinesti izboljšave, zato pustimo pri vozlišču C nedotaknjen potencial 9.

Ponovno gledamo najmanjšo začasno oznako in vidimo, da jo ima vozlišče E (17). Temu priredimo potencial 17 in gledamo vsa vozlišča, ki jih lahko dosežemo neposredno iz E (to je vozlišče F). Vozlišče F ima začasno oznako 20, povezava iz E pa bi prinesla začasno oznako 17+7 = 24. Ker je slednja opcija slabša, pustimo vozlišču F začasno oznako 20 in jo spremenimo v potencial (saj smo prišli do konca).

V tem trenutku smo vsem vozliščem dodelili potenciale, kar je razvidno na sliki 108 (potenciali so označeni pod številkami vozlišč v krogcih, potencial od A je 0).



Slika 108: Digraf z vsemi določenimi potenciali.

Strukturo najkrajše poti med A in F določimo tako, da potujemo iz vozlišča F nazaj proti vozlišču A. Pri tem upoštevamo le tiste povezave oz. segmente poti, kjer je razlika višjega in nižjega potenciala med določenima vozliščema enaka razdalji med tema vozliščema.

Očitno velja to za naslednje povezave (glej sliko 108):

$$\begin{aligned} potencial\ pri\ F\ - potencial\ pri\ C\ &= razdalja\ od\ C\ do\ F \\ potencial\ pri\ C\ - potencial\ pri\ D\ &= razdalja\ od\ C\ do\ D \\ potencial\ pri\ D\ - potencial\ pri\ A\ &= razdalja\ od\ A\ do\ D \end{aligned}$$

Torej bodo te povezave gotovo sestavni del najkrajše poti med A in F, ki se potemtakem glasi: A-D-C-F v skupni dolžini 20.

7 PROBLEM MAKSIMALNEGA PRETOKA SKOZI OMREŽJE

Pri tej problematiki se ukvarjamo s problemom, kako ugotoviti, kolikšen maksimalen promet določenega blaga lahko spravimo skozi dano omrežje. Tovrstni problemi se pri mrežni optimizaciji pogostokrat pojavljajo, z njimi pa želimo izračunati maksimalno količino pretoka določenega blaga, ki jo lahko transportiramo od vhoda v omrežje (izvor) do izhoda iz omrežja (ponor) [6]. Seveda je potrebno pri obravnavi problema upoštevati tudi omejitve, ki jih predstavljajo maksimalne kapacitete posameznih povezav, ki pomenijo maksimalen pretok blaga, ki ga lahko spravimo skozi določeno povezavo (npr. v določeni časovni enoti).

Pri obravnavi pretoka blaga skozi vozlišča običajno upoštevamo varianto, da se blago v posameznih vozliščih ne more kopičiti (skladiščiti), pač pa gre nemudoma naprej. V posameznih vozliščih potemtakem upoštevamo zakon, da je vsota pritekajočih tokov (pretokov blaga) v vozlišče enaka vsoti odtekajočih tokov (pretokov blaga) iz vozlišča [6]. Gre za ravnotežni zakon, ki deluje na enakih načelih kot Kirchoffov zakon za tokove v elektrotehniki, zato ga zaradi nazornosti poimenujmo z istim imenom. Če kot tokove obravnavamo odločitvene spremenljivke, lahko Kirhoffov zakon (Kirchoffovo ravnotežno enačbo) za neko vozlišče Vi matematično formuliramo na naslednji način [6]:

$$\begin{aligned} &\textit{vsota pritekajočih tokov v } V_i = \textit{vsota odtekajočih tokov iz } V_i \\ &\sum_{i=1}^{N} x_{vh}(i) = \sum_{i=1}^{M} x_{izh}(i) \end{aligned} \tag{7.1.}$$

kjer so $x_{vh}(i)$ odločitvene spremenljivke, ki predstavljajo vhodne tokove, ki pritekajo v vozlišče Vi, $x_{izh}(i)$ so odločitvene spremenljivke, ki predstavljajo izhodne tokove, ki odtekajo iz vozlišča Vi, N je število vhodnih tokov, M pa je število izhodnih tokov. Seveda tokovi $x_{vh}(i)$ in $x_{izh}(i)$ pritekajo v opazovano vozlišče iz sosednjih vozlišč oz. odtekajo k sosednjim vozliščem v omrežju.

Naštejmo nekaj tipičnih primerov aplikacij, kjer se pojavlja problem maksimalnega pretoka:

- Maksimizacija pretoka skozi distribucijsko mrežo podjetja od tovarn do potrošnikov,

- Maksimizacija pretoka skozi oskrbno mrežo podjetja od dobaviteljev do tovarn,
- Maksimizacija pretoka nafte skozi sistem cevovodov,
- Maksimizacija pretoka vode skozi vodovodni sistem,
- Maksimizacija pretoka vozil skozi transportno mrežo, itn.

V splošnem obstajata dva značilna pristopa, kako rešiti problem maksimalnega pretoka:

- Prevedba na problem linearnega programiranja in rešitev s simpleksno metodo, ter
- Rešitev problema maksimalnega pretoka s pomočjo takoimenovane metode povečanja poti (augmenting path), ki temelji na residualnih mrežah in povečanih poteh (Ford Fulkersonov algoritem).

V nadaljevanju si bomo pogledali pristop, kako problem maksimalnega pretoka prevesti na problem linearnega programiranja.

7.1 Prevedba problema maksimalnega pretoka na problem linearnega programiranja

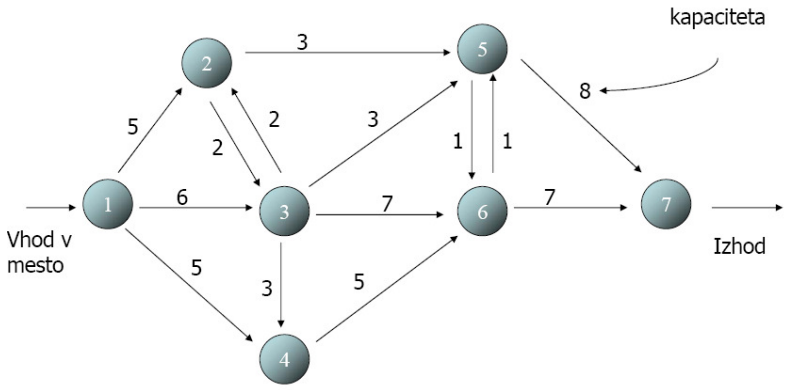
Kot bomo videli na primerih v nadaljevanju, lahko problem maksimalnega pretoka učinkovito prevedemo v problem linearnega programiranja [6, 17]. V ta namen bomo uvedli tudi takoimenovano umetno povratno zanko iz izhoda iz omrežja nazaj na vhod v omrežje, katere pretok blaga skoznjo bo enak vsoti vseh pretokov (skupni pretočnosti), ki gredo skozi omrežje. Če bomo nato z iskanjem optimalne rešitve v linearnem programu dosegli, da se maksimizira odločitvena spremenljivka, ki predstavlja pretok skozi povratno zanko, je logično, da bomo s tem dosegli tudi, da se maksimizira celoten pretok blaga (skupna pretočnost) skozi omrežje [6, 17].

Podrobnosti tega postopka bodo postale bolj razumljive pri obravnavi primerov v nadaljevanju.

Primer 1:

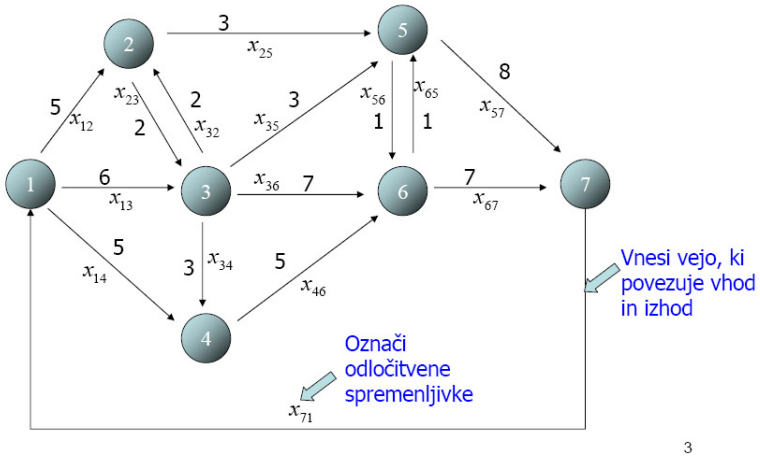
Za prvi primer uporabimo neko mestece, ves blagovni promet pa gre skozi mesto (se ne zadržuje v njem). Slika 109 prikazuje shemo (omrežje) mesta, označene so tudi

maksimalne kapacitete povezav, ki govorijo o maksimalnem možnem pretoku blaga skozi povezavo v določeni časovni enoti (npr. število ton blaga na minuto).



Slika 109.: Mreža mesteca, kjer želimo ugotoviti, kolikšen maksimalen promet blaga lahko spravimo skozi dano omrežje.

Najprej moramo na sliki 109. vpeljati vse odločitvene spremenljivke x_{ij}, ki predstavljajo pretok blaga iz vozlišča i v vozlišče j, označimo tudi navidezno povratno zanko iz izhoda iz omrežja nazaj na vhod v omrežje, katere pretok blaga skoznjo x_{71} je enak vsoti vseh pretokov, ki gredo skozi omrežje (skupni pretočnosti skozi omrežje). Tako pridemo do slike 110.



Slika 110.: Vpeljava odločitvenih spremenljivk x_{ij}, ki predstavljajo pretok blaga iz vozlišča i v vozlišče j. Vpeljava navidezne povratne zanke iz izhoda iz omrežja nazaj na vhod v omrežje.

Kriterijska funkcija, ki jo bomo maksimizirali, je:

$$\max \quad x_{71} = f(x_{ij}) \tag{7.2.}$$

in je seveda posredno funkcija vseh pretokov v omrežju, saj je pretok blaga skozi povratno zanko x_{71} enak vsoti vseh pretokov, ki gredo skozi omrežje (skupni pretočnosti skozi omrežje).

Ravnotežje tokov za vsako vozlišče zapišemo z naslednjimi Kirchoffovimi ravnotežnimi enačbami , pri čemer za vsa vozlišča uporabimo izraz (7.1.) (glej tudi sliko 110.):

$$\begin{aligned}
&x_{71} = x_{12} + x_{13} + x_{14} && (\textit{vozlišče } 1)\\
&x_{12} + x_{32} = x_{23} + x_{25} && (\textit{vozlišče } 2)\\
&x_{13} + x_{23} = x_{32} + x_{34} + x_{35} + x_{36} && (\textit{vozlišče } 3)\\
&x_{14} + x_{34} = x_{46} && (\textit{vozlišče } 4)\\
&x_{25} + x_{35} + x_{65} = x_{56} + x_{57} && (\textit{vozlišče } 5)\\
&x_{36} + x_{46} + x_{56} = x_{65} + x_{67} && (\textit{vozlišče } 6)\\
&x_{57} + x_{67} = x_{71} && (\textit{vozlišče } 7)
\end{aligned} \qquad (7.3.)$$

torej za vsa vozlišča opazujemo tokove, ki pritekajo vanje, ter tokove, ki odtekajo iz njih. Pri tem na levih straneh straneh enačb tvorimo vsoto pritekajočih tokov, na desnih straneh enačb pa vsoto odtekajočih tokov, za vsako vozlišče posebej.

Upoštevati moramo še preostale omejitve, kjer poudarimo maksimalno kapaciteto vsake povezave, skozi kateri gre lahko največji možen pretok blaga (glej sliki 109. in 110.). Tako dobimo:

$$\begin{aligned}
&x_{12} \le 5, \quad x_{13} \le 6, \ x_{14} \le 5, \ x_{23} \le 2, \ x_{25} \le 3,\\
&x_{32} \le 2, \quad x_{34} \le 3, \ x_{35} \le 3, \ x_{36} \le 7, \ x_{46} \le 5,\\
&x_{56} \le 1, \quad x_{57} \le 8, \ x_{65} \le 1, \ x_{67} \le 7\\
&x_{ij} \ge 0
\end{aligned} \qquad (7.4.)$$

V izrazu (7.4.) smo seveda tudi upoštevali, da pretoki blaga ne morejo biti negativni.

Če sedaj strnemo izraze (7.2.), (7.3.) in (7.4.) ter jih zapišemo v kompaktni pregledni obliki, dobimo klasičen problem linearnega programiranja:

$$\max \; x_{71}$$

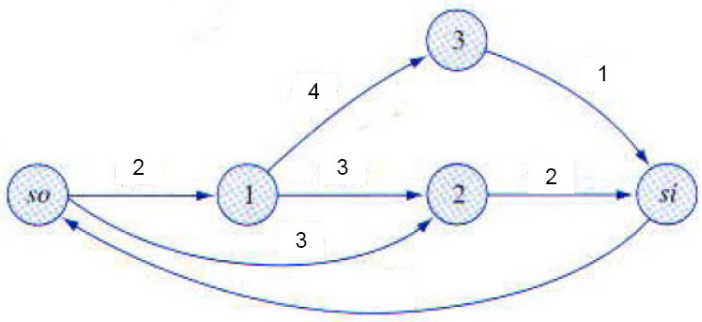
$$\begin{aligned}
&x_{71} = x_{12} + x_{13} + x_{14} && (\textit{vozlišče } 1)\\
&x_{12} + x_{32} = x_{23} + x_{25} && (\textit{vozlišče } 2)\\
&x_{13} + x_{23} = x_{32} + x_{34} + x_{35} + x_{36} && (\textit{vozlišče } 3)\\
&x_{14} + x_{34} = x_{46} && (\textit{vozlišče } 4)\\
&x_{25} + x_{35} + x_{65} = x_{56} + x_{57} && (\textit{vozlišče } 5)\\
&x_{36} + x_{46} + x_{56} = x_{65} + x_{67} && (\textit{vozlišče } 6)\\
&x_{57} + x_{67} = x_{71} && (\textit{vozlišče } 7)
\end{aligned} \qquad (7.5.)$$

$$\begin{aligned}
&x_{12} \le 5, \quad x_{13} \le 6, \; x_{14} \le 5, \; x_{23} \le 2, \; x_{25} \le 3,\\
&x_{32} \le 2, \quad x_{34} \le 3, \; x_{35} \le 3, \; x_{36} \le 7, \; x_{46} \le 5,\\
&x_{56} \le 1, \quad x_{57} \le 8, \; x_{65} \le 1, \; x_{67} \le 7\\
&x_{ij} \ge 0
\end{aligned}$$

Seveda je potrebno izraze (7.5.) še predelati v primerno obliko za reševanje z računalniškim programom. Dobljena optimalna rešitev, ki bo upoštevajoč ravnotežne enačbe in omejitve kapacitet vej maksimizirala odločitveno spremenljivko x_{71}, bo omogočila, da bomo lahko izračunali maksimalen pretok blaga, ki ga lahko pošljemo skozi mesto. V skladu s tem se bodo seveda tudi odločitvene spremenljivke x_{ij} v omrežju nastavile na takšno optimalno vrednost, da bo dosežen maksimalen pretok blaga.

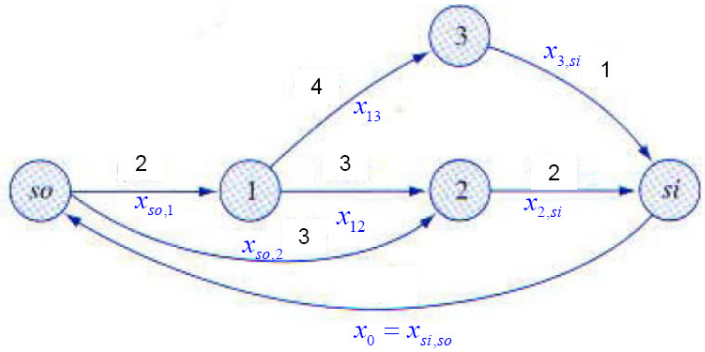
Primer 2:

Za dano mrežo nastavite matematični model za problem maksimalnega pretoka skozi omrežje v obliki linearnega programa! Pri tem so na povezavah na sliki označene kapacitete posameznih pretokov (**v mio sodčkov nafte na uro**).



Slika 111.: Podana mreža naftne infrastrukture, kjer želimo ugotoviti, kolikšen maksimalen promet sodčkov nafte lahko spravimo skozi omrežje

Najprej moramo na sliki 111. vpeljati vse odločitvene spremenljivke x_{ij}, ki predstavljajo pretok blaga iz vozlišča i v vozlišče j, označimo tudi navidezno povratno zanko iz izhoda iz omrežja nazaj na vhod v omrežje, katere pretok blaga skoznjo x_0 je enak vsoti vseh pretokov, ki gredo skozi omrežje (skupni pretočnosti skozi omrežje). Tako pridemo do slike 112.



Slika 112.: Vpeljava odločitvenih spremenljivk x_{ij}, ki predstavljajo pretok sodčkov nafte iz vozlišča i v vozlišče j. Vpeljava navidezne povratne zanke iz izhoda iz omrežja nazaj na vhod v omrežje.

Kriterijska funkcija, ki jo bomo maksimizirali, je:

$$\max \quad x_0 = \max \quad x_{si,so} = \quad f(x_{ij}) \tag{7.6.}$$

in je seveda posredno funkcija vseh pretokov v omrežju, saj je pretok blaga skozi povratno zanko x_0 enak vsoti vseh pretokov, ki gredo skozi omrežje (skupni pretočnosti skozi omrežje).

Ravnotežje tokov za vsako vozlišče zapišemo z naslednjimi Kirchoffovimi ravnotežnimi enačbami , pri čemer za vsa vozlišča uporabimo izraz (7.1.) (glej tudi sliko 112.):

$$\begin{aligned}
&x_0 = x_{so,1} + x_{so,2} && (vozlišče\ \ so)\\
&x_{so,1} = x_{12} + x_{13} && (vozlišče\ \ 1)\\
&x_{12} + x_{so,2} = x_{2,si} && (vozlišče\ \ 2)\\
&x_{13} = x_{3,si} && (vozlišče\ \ 3)\\
&x_{2,si} + x_{3,si} = x_0 && (vozlišče\ \ si)
\end{aligned} \tag{7.7.}$$

torej za vsa vozlišča opazujemo tokove, ki pritekajo vanje, ter tokove, ki odtekajo iz njih. Pri tem na levih straneh straneh enačb tvorimo vsoto pritekajočih tokov, na desnih straneh enačb pa vsoto odtekajočih tokov, za vsako vozlišče posebej.

Upoštevati moramo še preostale omejitve, kjer poudarimo maksimalno kapaciteto vsake povezave, skozi kateri gre lahko največji možen pretok blaga (glej sliki 111. oz. 112.). Tako dobimo:

$$\begin{aligned} x_{so,1} &\le 2 \\ x_{so,2} &\le 3 \\ x_{12} &\le 3 \\ x_{13} &\le 4 \\ x_{2,si} &\le 2 \\ x_{3,si} &\le 1 \\ x_{ij} &\ge 0 \end{aligned} \qquad (7.8.)$$

V izrazu (7.8.) smo seveda tudi upoštevali, da pretoki blaga ne morejo biti negativni.

Če sedaj strnemo dobljene izraze ter jih zapišemo v kompaktni pregledni obliki, dobimo klasičen problem linearnega programiranja:

$$\begin{aligned} &\max\ x_0 \\ &\textit{ravnotežne enačbe}: \\ &x_0 = x_{so,1} + x_{so,2} \quad (\textit{vozlišče so}) \\ &x_{so,1} = x_{12} + x_{13} \quad (\textit{vozlišče 1}) \\ &x_{12} + x_{so,2} = x_{2,si} \quad (\textit{vozlišče 2}) \\ &x_{13} = x_{3,si} \quad (\textit{vozlišče 3}) \\ &x_{2,si} + x_{3,si} = x_0 \quad (\textit{vozlišče si}) \\ &\textit{omejitve}: \\ &x_{so,1} \le 2 \\ &x_{so,2} \le 3 \\ &x_{12} \le 3 \\ &x_{13} \le 4 \\ &x_{2,si} \le 2 \\ &x_{3,si} \le 1 \\ &x_{ij} \ge 0 \end{aligned} \qquad (7.9.)$$

Izkaže se, da je optimalna rešitev linearnega programa (7.9.) naslednja:

$$\begin{aligned} x_{so,1}^{*} &= 2 \\ x_{so,2}^{*} &= 1 \\ x_{12}^{*} &= 1 \\ x_{13}^{*} &= 1 \\ x_{2,si}^{*} &= 2 \\ x_{3,si}^{*} &= 1 \\ x_0^{*} &= x_{so,1}^{*} + x_{so,2}^{*} = x_{2,si}^{*} + x_{3,si}^{*} = 3 \end{aligned} \qquad (7.10.)$$

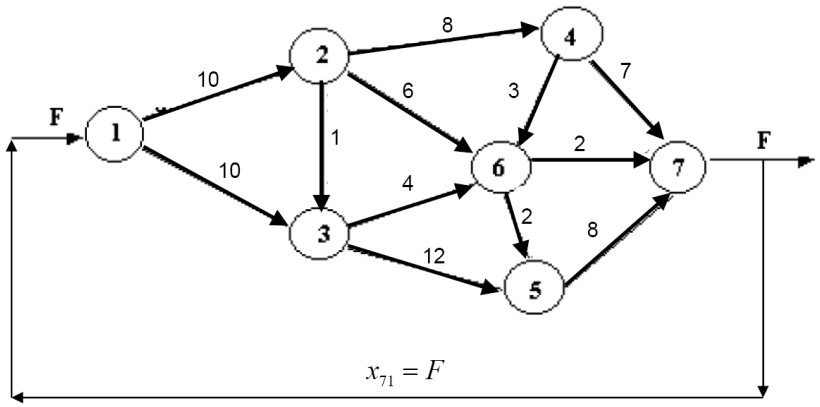
in jo dobimo s simpleksnim programom za reševanje problemov linearnega programiranja v enem izmed ustreznih orodij (Scilab, Matlab, LINGO, itn).

Iz optimalne rešitve (7.10.) pa se da povleči naslednji sklep:

Maksimalen možen pretok je 3 mio sodčkov nafte na uro od izvora so do ponora si, pri čemer se pošlje 1 mio sodčkov po poti s0-1-2-si, 1 mio sodčkov po poti s0-1-3-si, ter 1 mio sodčkov po poti s0-2-si

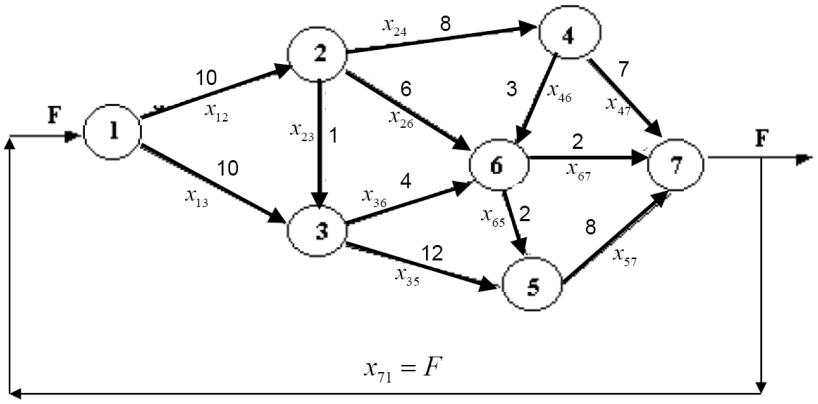
Primer 3:

Za dano mrežo nastavite matematični model za problem maksimalnega pretoka skozi omrežje v obliki linearnega programa! Pri tem so na povezavah na sliki 113 označene kapacitete posameznih pretokov.



Slika 113.: Podana mreža, kjer želimo ugotoviti, kolikšen maksimalen promet blaga lahko spravimo skozi omrežje

Najprej moramo na sliki 113. vpeljati vse odločitvene spremenljivke x_{ij}, ki predstavljajo pretok blaga iz vozlišča i v vozlišče j, označimo tudi navidezno povratno zanko iz izhoda iz omrežja nazaj na vhod v omrežje, katere pretok blaga skoznjo x_{71} je enak vsoti vseh pretokov, ki gredo skozi omrežje (skupni pretočnosti skozi omrežje). Tako pridemo do slike 114.



Slika 114.: Vpeljava odločitvenih spremenljivk x_{ij}, ki predstavljajo pretok blaga iz vozlišča i v vozlišče j. Vpeljava navidezne povratne zanke iz izhoda iz omrežja nazaj na vhod v omrežje.

Kriterijska funkcija, ki jo bomo maksimizirali, je:

$$\max \quad x_{71} = f(x_{ij}) \tag{7.11.}$$

in je seveda posredno funkcija vseh pretokov v omrežju, saj je pretok blaga skozi povratno zanko x_{71} enak vsoti vseh pretokov, ki gredo skozi omrežje (skupni pretočnosti skozi omrežje).

Ravnotežje tokov za vsako vozlišče kot v prejšnjih primerih zapišemo s Kirchoffovimi ravnotežnimi enačbami, pri čemer za vsa vozlišča uporabimo izraz (7.1.) (glej tudi sliko 114.). Poleg tega moramo upoštevati še preostale omejitve, kjer poudarimo maksimalno kapaciteto vsake povezave, skozi kateri gre lahko največji možen pretok blaga (glej sliki 113. oz. 114.). Če torej strnemo vse izraze (kriterijska funkcija, Kirchoffove enačbe, omejitve glede maksimalnih kapacitet povezav) ter jih zapišemo v kompaktni pregledni obliki, dobimo naslednji problem linearnega programiranja:

$$\max \; x_{71}$$

$$\begin{aligned}
x_{71} &= x_{12} + x_{13} && (\textit{vozlišče } 1)\\
x_{12} &= x_{23} + x_{24} + x_{26} && (\textit{vozlišče } 2)\\
x_{13} + x_{23} &= x_{35} + x_{36} && (\textit{vozlišče } 3)\\
x_{24} &= x_{46} + x_{47} && (\textit{vozlišče } 4)\\
x_{35} + x_{65} &= x_{57} && (\textit{vozlišče } 5)\\
x_{26} + x_{36} + x_{46} &= x_{65} + x_{67} && (\textit{vozlišče } 6)\\
x_{47} + x_{57} + x_{67} &= x_{71} && (\textit{vozlišče } 7)
\end{aligned} \qquad (7.12.)$$

$$\begin{aligned}
&x_{12} \le 10, \quad x_{13} \le 10, \; x_{23} \le 1, \; x_{24} \le 8, \; x_{26} \le 6,\\
&x_{35} \le 12, \quad x_{36} \le 4, \; x_{46} \le 3, \; x_{47} \le 7, \; x_{57} \le 8,\\
&x_{65} \le 2, \quad x_{67} \le 2\\
&x_{ij} \ge 0
\end{aligned}$$

7.2 Reševanje problema maksimalnega pretoka s Ford Fulkersonovim algoritmom

Algoritem je bil razvit s strani L.R. Forda in D.R. Fulkersona leta 1956. Gre za pogosto uporabljan algoritem, ko poskušamo določiti maksimalen pretok, ki gre lahko skozi neko omrežje na poti od izvornega do ciljnega vozlišča. Algoritem poskuša najti takšne uspešne poti med izvorom in ponorom, ki lahko pošljejo več dodatnega pretoka med dotičnima vozliščema [12]. To pomeni, da ima vsak od teh poti zmožnost povečanja skupnega pretoka skozi omrežje, zato jim pravimo tudi razširjene poti (augmenting paths) [12].

Pri danem omrežju je prvi korak, da poiščemo ustrezno razširjeno pot od izvora do ponora, pri čemer je pretok preko te poti očitno enak minimumu kapacitet na povezavah vzdolž te poti. Takšnemu pretoku pogosto pravimo tudi razširjen pretok (augmenting flow). Ko je dotični pretok poslan skozi omrežje, nato skušamo ugotoviti, če se da skozenj poslati še kaj dodatnega pretoka preko drugih razširjenih poti. To preverjanje izvedemo z vpeljavo pojma takoimenovanega residualnega omrežja. Pri tem omrežju so kapacitete nesaturiranih povezav (kjer je nivo pretoka nižji od kapacitete) reducirane z

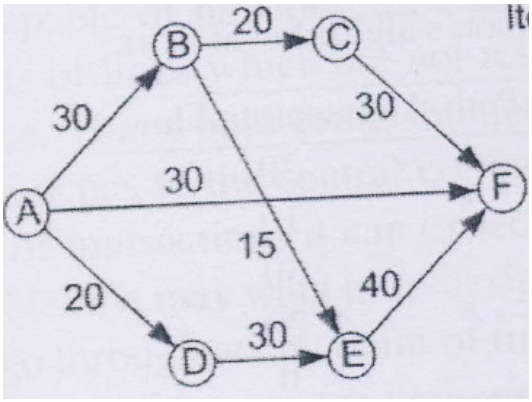
vrednostjo razširjenega pretoka [12]. Kar se pa tiče saturiranih povezav, so le-te odstranjene iz residualnega omrežja in nadomeščene s povezavami v nasprotni smeri, ki imajo enako kapaciteto kot originalne saturirane povezave [12].

Postopek se nadaljuje z iskanjem preostalih razširjenih poti, za katere bi se lahko poslalo še več razširjenih pretokov skozi residualno omrežje. Algoritem se konča, ko ni več možno poslati nobenega novega razširjenega pretoka skozi residualno omrežje. Maksimalni pretok kot končni rezultat pa nato dobimo tako, da seštejemo vse razširjene (delne) pretoke, ki smo jih generirali v posameznih fazah postopka [12].

Princip delovanja algoritma bomo ponazorili na naslednjem primeru [12].

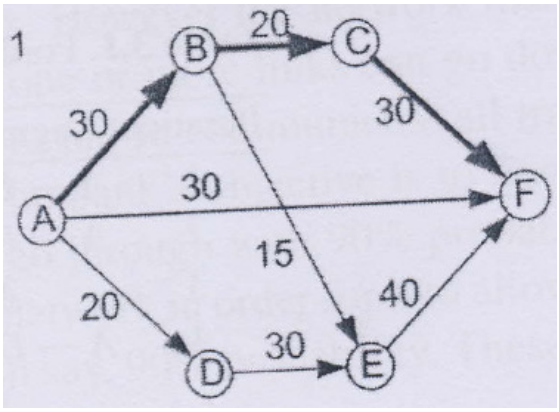
Primer:

Za omrežje na sliki 115 poiščite maksimalen možen pretok (od A do F) s pomočjo Ford Fulkersonovega algoritma.



Slika 114.: Omrežje, kjer bomo poiskali maksimalen pretok s Ford Fulkersonovim algoritmom.

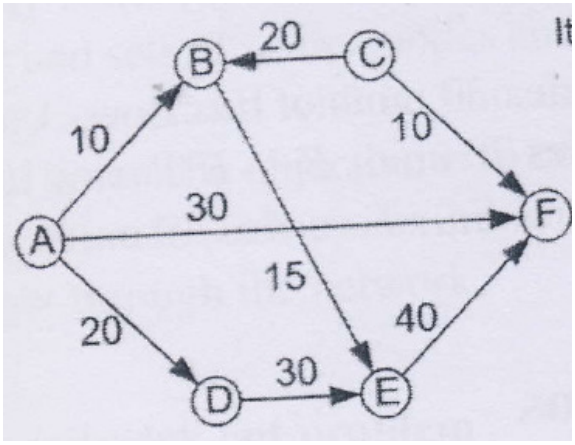
Na tem omrežju (ki je tudi začetno residualno omrežje) poiščimo prvo razširjeno pot, denimo je to pot preko vozlišč A-B-C-F, ki je ilustrirana na sliki 115.



Slika 115.: Prva razširjena pot (označena z odebeljeno črto).

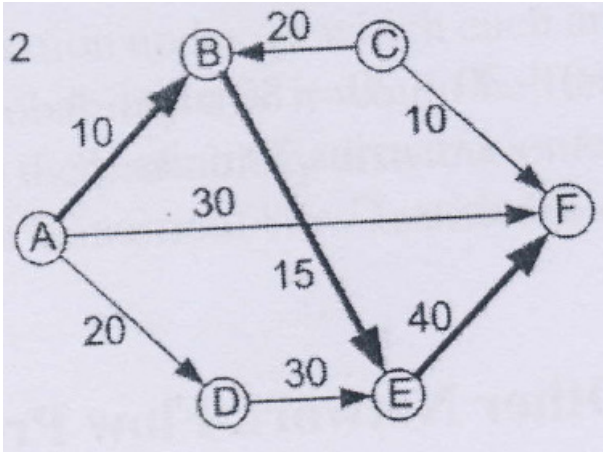
Razširjen pretok skozi to razširjeno pot je: min(30, 20, 30) = 20 enot. Pri tem sta povezavi med A in B in C in F nesaturirani (ker je razširjen pretok manjši od njune kapacitete 30), povezava B-C pa je saturirana (saj je razširjen pretok enak kapaciteti 20).

Kot posledico tega dobimo residualno omrežje v naslednji iteraciji, prikazano na sliki 116.



Slika 116.: Residualno omrežje v naslednji, 2. iteraciji

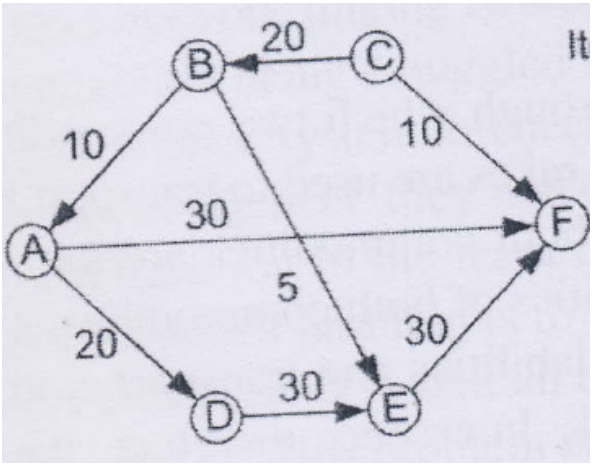
Kot je razvidno iz slike 116, se je kapaciteta nesaturiranih povezav A-B in C-F zmanjšala za 20, saturirana povezava B-C pa je obrnila smer. V nadaljevanju na residualnem omrežju na sliki 116 poiščimo novo razširjeno pot, denimo je to pot preko vozlišč A-B-E-F, ki je ilustrirana na sliki 117.



Slika 117.: Druga razširjena pot (označena z odebeljeno črto).

Razširjen pretok skozi to razširjeno pot je: min(10, 15, 40) = 10 enot. Pri tem je povezava med A in B saturirana, povezavi med B in E ter E in F sta pa nesaturirani.

Kot posledico tega dobimo residualno omrežje v naslednji iteraciji, prikazano na sliki 118.

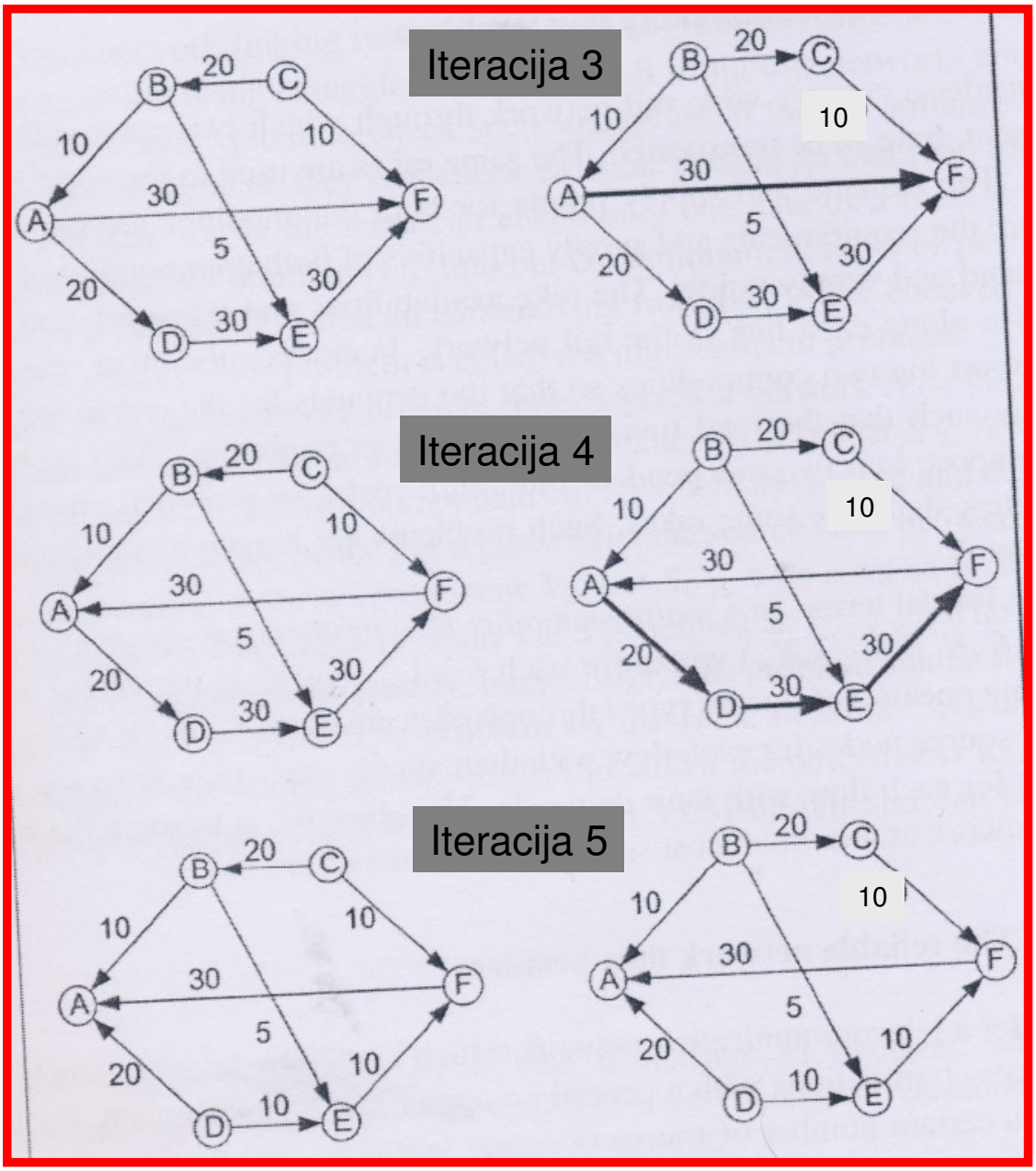


Slika 118.: Residualno omrežje v naslednji, 3. iteraciji

Kot je razvidno iz slike 118, se je kapaciteta nesaturiranih povezav B-E in E-F zmanjšala za 10, saturirana povezava A-B pa je obrnila smer.

V nadaljevanju na residualnem omrežju na sliki 118 zopet poiščemo novo razširjeno pot, jo ustrezno označimo, ugotovimo nov razširjen pretok, ter pridemo do novega residualnega omrežja. Postopek na takšen način nadaljujemo toliko časa, dokler ni več mogoče na residualnem omrežju najti nobene nove razširjene poti. Slika 119 ponazarja nadaljnje iteracije (3., 4., 5.) postopka, ki se očitno konča po petih iteracijah. Pri tem so

na levih slikah slike 119 označena nova residualna omrežja v posameznih iteracijah, na desnih slikah pa nove razširjene poti na teh omrežjih. Seveda le-te ni več mogoče dobiti v zadnji iteraciji (zadnja desna slika), zato tudi ni nobene označbe z odebeljenim tiskom.



Slika 119.: Nadaljnje iteracije Ford Fulkersonovega postopka (na levi so residualna omrežja, na desni pa označene razširjene poti)

Kot je razvidno iz slike 119, smo v 3. iteraciji tvorili razširjen pretok v višini 30 enot, v 4. iteraciji v višini 20 enot, v peti iteraciji pa smo končali postopek, saj ni več možno najti nobene razširjene poti od izvora do ponora.

Maksimalen pretok dobimo kot vsoto vseh razširjenih (delnih) pretokov v posameznih iteracijah postopka. Njegova vrednost je: 20+10+30+20 = 80 enot. Takšna je torej maksimalna vrednost pretoka, ki ga lahko spravimo skozi omrežje na sliki 114.

Zanima nas tudi, na kakšne dele se ta pretok razdeli, ko potuje preko posameznih povezav omrežja na sliki 114. To lahko ugotovimo tako, da tvorimo tabelo na sliki 120 (glej tudi slike 115 do 119).

Iteracija	razširjena pot	razširjen pretok
1	$A \rightarrow B \rightarrow C \rightarrow F$	20
2	$A \rightarrow B \rightarrow E \rightarrow F$	10
3	$A \rightarrow F$	30
4	$A \rightarrow D \rightarrow E \rightarrow F$	20
5	/	0

Slika 120.: Razširjene poti in razširjeni pretoki v posameznih iteracijah postopka

Očitno povezava A-B nosi 20+10 = 30 enot pretoka, A-F 30 enot, A-D 20 enot, B-C 20 enot, B-E 10 enot, D-E 20 enot, C-F 20 enot, ter E-F 10+20 = 30 enot. Tako se torej maksimalen pretok porazdeli po posameznih povezavah omrežja na sliki 114.

8 ALGORITMI ZA REŠEVANJE PROBLEMA TRGOVSKEGA POTNIKA

Problem trgovskega potnika (PTP) sodi v skupino zelo zahtevnih NP - težkih problemov s področja kombinatorične optimizacije, ki se pojavlja v številnih aplikacijah s področja operacijskih raziskav in računalniških znanosti [15, 16, 19]. Kot smo spoznali v prejšnjih poglavjih, matematično gledano ta problem preprosto formuliramo takole: V uteženem polnem grafu z n vozlišči je potrebno izmed (n-1)! različnih možnosti za Hamiltonov cikel poiskati minimalni Hamiltonov cikel. Torej je potrebno poiskati cikel, ki predstavlja takšen minimalen sprehod po grafu, pri katerem je vsaka točka, razen prve in zadnje, zajeta natanko enkrat. Pri tem je utež sprehoda vsota uteži njegovih povezav.

V praktičnem smislu PTP npr. pomeni, kako poiskati takšno krožno pot, da bomo obiskali vsa mesta in da bo hkrati ta naš sprehod čim cenejši. Pogoj pa je seveda, da poznamo, koliko nas stane prehod iz enega mesta v drugo.

PTP je izredno kompleksen problem, zato težimo k takšnim algoritmom, ki dovolj hitro najdejo približno optimalno pot, z natančnostjo, ki je zadovoljiva z vidika danega problema. Načinov iskanja rešitve takšnih problemov je sicer več, najbolj optimalne rešitve pa večinoma ne zagotavlja noben način [15, 16, 19]. Dobimo lahko le približne rešitve. Torej, takšne rešitve niso nujno najboljše, so pa sprejemljive in dobljene z razumno vloženim trudom.[1]

Problem trgovskega potnika ima že dolgo zgodovino v svetu operacijskih raziskav. Ko so se v začetku petdesetih pojavili prvi pomebnejši rezultati v kombinatorični optimizaciji in operacijskih raziskavah (linearno programiranje, prirejenje, pretoki, razporejanje

[1] Če bi problem trgovskega potnika reševali tako, da bi preiskali vse možnosti, bi se mnogi deli poti ponavljali in izračunavali bi veliko neoptimalnih možnosti. Pregledovanje vseh možnosti bi bilo predrago, ne glede na to, kako zmogljiv računalnik bi imeli. Če predpostavimo, da za en obisk vozlišča potrebujemo 1 nanosekundo, bi za pregled grafa z 10 vozlišči potrebovali dobre pol minute, za pregled grafa s 15 vozlišči, pa bi potrebovali že več kot pol leta. Iskanje optimalne rešitve je pogosto prezahtevno, zato se zadovoljimo s približnimi rešitvami, katerih kakovost je še zadovoljiva, četudi običajno ne tudi popolnoma optimalna [15, 16, 19].

poslov,...), je bila trdovratnost PTP deležna toliko večje pozornosti. Prvi večji preboj v iskanju dokazljivo optimalne poti se je zgodil leta 1954, ko so G. B. Dantzig, D. R. Fulkerson in S. M. Johnson rešili problem trgovskega potnika na množici 49 mest v Združenih Državah Amerike. Naslednji dramatični uspehi so sledili leta 1980, ko sta Crowder in Padberg našla dokazljivo optimalno pot za 318 mest [15, 16, 19].

Takšen problem pa se ne pojavlja samo pri trgovskem potniku v dobesednem smislu, pač pa se z njim soočamo tudi marsikje drugod. Tako npr. pri vozniku dostavnega kamiona, ki razvaža blago v trgovine, poštarju, ki mora raznositi pošto, učencu, ki raznaša vabila za prireditev, delavcu, ki dela s strojem različne izdelke, pri čemer čas, odvisen za pripravo stroja, zavisi od tega, kaj želi delati in kaj je delal prej, itn. Ta problem lahko prepoznamo tudi pri pripravljanju hrane, kjer moramo čez različne faze pripravljanja, kjer se pri predelavi izgubljajo vitamini, mi pa želimo, da bo izguba vitaminov najmanjša. Problem trgovskega potnika se lahko pojavi tudi v številnih primerih prometa in transporta, kjer se pod pojmom "trgovski potnik" lahko razumejo letala, ladje, tovornjaki, avtobusi, itn. V vozliščih, ki jih ta prevozna sredstva obiskujejo, pa se lahko vrši prevzem ali oddaja blaga ali/in potnikov [15, 16, 19].

Čeprav gre za zelo zahteven problem, je bilo za njegovo reševanje razvitih že veliko tehnik in metod, tako hevrističnih (ki poiščejo rešitev blizu optimalne), kot tudi popolnoma optimalnih (poiščejo točno rešitev). Nekatere najbolj poznane tehnike za reševanje problema trgovskega potnika so [15, 16, 19]:

- Požrešni algoritem,
- Algoritem najbližjega soseda,
- Algoritmi za lokalno iskanje,
- Pristop s takoimenovano tehniko 'Tabu search',
- Pristop z metodo Razveji in omeji (Branch & Bound),
- Pristop z dinamičnim programiranjem,
- Pristop s sestopanjem, itn.

Za te algoritme je značilno, da temeljijo bodisi na gradnji cikla, bodisi na razporejanju delov cikla, ali pa na širitvi cikla v smislu minimalnega obhoda [15, 16, 19].

Seveda obstaja več kategorij problematike trgovskega potnika, tako npr. ločimo med simetričnim in nesimetričnim problemom, problem ima lahko tudi določene omejitve, v nekaterih primerih pride do časovnih oken, ki jih je potrebno upoštevati, in podobno [15, 16, 19].

Pojasnimo še nekoliko natančneje razliko med popolnoma optimalnimi in hevrističnimi rešitvami [15, 16, 19]. Ker je PTP po svoji naravi kombinatorni problem, je potrebno iz velike množice možnih rešitev (npr. možnih poti prevoznih sredstev) izluščiti kar najbolj optimalno rešitev. Pri iskanju slednje običajno težimo k minimizaciji skupne stroškovne funkcije oz. minimalnim transportnim stroškom. Če imamo na razpolago dovolj računalniškega časa, bomo pri reševanju PTP bržkone izbrali kakšnega od eksaktnih algoritmov, ki nas bo vodil do popolnoma optimalne rešitve. Pri tem se poraba računalniškega časa meri v številu elementarnih računskih operacij, ki jih je potrebno opraviti v najslabšem možnem primeru, da bi dosegli optimalno rešitev. V splošnem ločimo med "dobrimi" in "slabimi" algoritmi. Prvi spadajo v kategorijo takoimenovanih polinomskih algoritmov, kjer je računska kompleksnost proporcionalna določeni polinomski funkciji predstavnikov dimenzije problema. V nasprotju s temi algoritmi, pa velja za slabe algoritme eksponentna (nepolinomska) odvisnost od predstavnikov dimenzije problema. Med hevristične algoritme spadajo metode, ki so zasnovane na izkušnjah in intuiciji, s katerimi je mogoče najti "dobre rešitve" v "sprejemljivem" času računanja z računalnikom. Naziv "hevristični" izhaja iz Grške terminologije in pomeni nekaj odkriti ali najti. V večini zelo kompleksnih kombinatoričnih problemov se vsekakor izkaže, da nas zgolj hevristični algoritmi lahko v sprejemljivem času pripeljejo do dobre rešitve [15, 16, 19].

V nadaljevanju si bomo pogledali nekaj značilnih metod, ki se najbolj pogosto uporabljajo pri reševanju klasičnega PTP [15, 16, 19].

8.1 Reševanje PTP z metodo razveji in omeji

Dan je graf G z matriko C, kjer je C_{ij} (nenegativna) vrednost povezave med vozliščema i in j, ki zavzame vrednost ∞, če povezave med vozliščema ni. Poiskati moramo krožno pot (cikel) v grafu na takšen način, da vsako vozlišče obiščemo natanko enkrat in je vsota povezav minimalna.

V splošnem za strategijo reševanja velja [15, 16, 19]:

- imamo n vozlišč in moramo vsa obiskati,
- poznamo vrednost (ceno, čas,…) prehoda iz enega vozlišča v drugega,
- želimo čim ugodneje obiskati vsa vozlišča in priti na začetek.
- lahko začnemo v kateremkoli vozlišču.

Pri metodi Razveji in omeji se uporablja ocenjevanje obetavnosti vozlišč, postopek sta leta 1958 predlagala Eastman in Croes, objavili pa leta 1963 Little, Murty, Sweeney in Karel [15].

Prvi korak postopka je, da izdelamo spisek vseh možnih povezav in njihovih cen, ter iz tega sestavimo matriko sosednosti (glej poglavje 2), v kateri so vse možne povezave in njihove vrednosti (povezave iz grafa oz. njihove uteži torej prenesemo v matriko sosednosti).

Za uporabo metode razveji in omeji potrebujemo primerno oceno spodnje meje dolžine krožne poti, ki jo dobimo na osnovi matrike sosednosti. Iz vozlišča 1 do naslednjega vozlišča krožne poti trgovski potnik potrebuje vsaj toliko, kot je dolga najkrajša povezava iz vozlišča 1. Ker mora iz vozlišča 1 v natančno eno od preostalih vozlišč, lahko najmanjši element v matriki sosednosti odštejemo od vseh ostalih v določeni vrstici te matrike. Podobno velja za vse ostale vrstice.

Iz vsakega od vozlišč stopi trgovski potnik natanko enkrat. Toda tudi v vsako od vozlišč (razen v začetno vozlišče) pride trgovski potnik natanko enkrat, kar ga stane vsaj toliko,

kot je oddaljeno najbližje vozlišče. Zato lahko tudi po stolpcih v matriki sosednosti odštejemo vrednosti najmanjših elementov. Pravimo, da smo pri prehodu iz enega v drugo vozlišče opravili **redukcijo matrike sosednosti** oz. smo jo reducirali [15, 16, 19].

Reduciramo jo lahko na dva načina, ki pa ne ponujata nujno enakega rezultata oz. enake cene. Oba načina redukcije pa sta pravilna. Poznamo torej [15, 16, 19]:

- Prvi način: najprej redukcija po vrsticah, nato redukcija po stolpcih.
- Drugi način: najprej redukcija po stolpcih, nato redukcija po vrsticah.

Ključni princip metode je, da tvorimo vsote vseh odštetih vrednosti (v stolpcih in vrsticah) pri redukciji spreminjajoče se matrike sosednosti tekom izgradnje ocene za spodnjo mejo dolžine krožne poti v originalnem grafu oz. v prvotnem grafu. To seveda tudi pomeni, da vse krožne poti stanejo vsaj toliko, kolikor je spodnja meja njihove dolžine.

Ker sme trgovski potnik v vsako vozlišče samo po enkrat, ima v vozlišču 1 na izbiro n-1 preostalih poti za dosego cilja, nato n-2, itn. Vse možne krožne poti trgovskega potnika lahko s pomočjo redukcije spreminjajoče se matrike sosednosti in ocenjevanja spodnje meje dolžine krožne poti opišemo v takoimenovanem **permutacijskem drevesu globine n ali drevesu stanj** [15, 16, 19]. Ko v tem drevesu po določeni veji prodiramo v njegovo notranjost, pri tem dodajamo na novo obiskana vozlišča, pri vsakem prehodu vozlišča spremenimo vrednost reducirane matrike sosednosti, ter tako gradimo oceno spodnje meje dolžine krožne poti.

Drevo stanj nam torej pomaga, da se reševanje problema PTP vrši veliko bolj pregledno in da takoj opazimo, katera vrednost spodnje meje dolžine (delne) krožne poti je najmanjša, kako pridemo do nje in če ima katera druga dolžina (delne) krožne poti morebiti enako vrednost. Če na primer neko krožno pot gradimo v določeno smer že do četrtega vozlišča in nato ocenimo, da bi dala neka druga začetna povezava v določeno

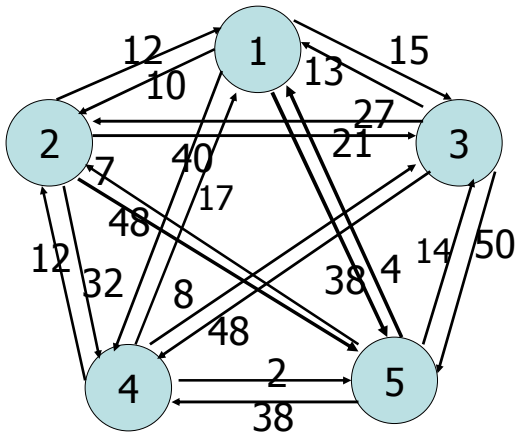
vozlišče manjšo vrednost ocene spodnje meje dolžine krožne poti, lahko spremenimo smer in nadaljujemo po tej drugi začetni povezavi.

Na koncu nas seveda zanima samo vrednost najcenejše krožne poti in pripadajoča veja v drevesu stanj, zato vse ostale veje odstranimo iz drevesa stanj. Razvijemo ga torej tako, da vedno potujemo v smeri najbolj obetavne veje. Ko dokončno razvijemo drevo stanj oz. ko pridemo do najcenejše rešitve, ki obide vsa vozlišča, oklestimo iz njega »slabe veje«. To so tiste veje, ki imajo ceno določene krožne poti večjo od tiste najbolj obetavne (najcenejše) krožne poti. Seveda imamo najcenejšo ocenjeno krožno pot (cikel) natanko tedaj, ko upoštevamo vsa vozlišča na najbolj obetavni veji drevesa stanj, ki je edine nismo oklestili.

V nadaljevanju si bomo ogledali primer, ki nam bo pomagal dodatno osvetliti princip delovanja metode Razveji in omeji.

Primer:

Rešite problem trgovskega potnika za graf na sliki 121 s pomočjo metode Razveji in Omeji, pri čemer poskušajte poiskati najcenejšo pot skozi vsa vozlišča.



Slika 121.: Graf, kjer želimo rešiti PTP s pomočjo metode Razveji in omeji

Iz grafa na sliki 121 poskušajmo najprej tvoriti matriko sosednosti, kar storimo v tabeli 1.

	1	2	3	4	5
1	∞	10	15	40	38
2	12	∞	21	32	48
3	13	27	∞	48	50
4	17	12	8	∞	2
5	4	7	24	38	∞

Tabela 1.: Matrika sosednosti originalnega grafa

Kot je razvidno iz slike 121 in tabele 1, prehod iz levega v desni element posamezne vrstice pomeni prehod iz enega v drugo vozlišče (razen elementov v diagonali, saj iz istega vozlišča ne moremo preiti nazaj v isto vozlišče, zato oznaka ∞). Nato reduciramo matriko v tabeli 1 po vrsticah in nato po stolpcih. To pomeni, da najprej odštejemo od vseh elementov matrike v tabeli 1 najmanjšo vrednost v vrsticah, kar ponazarja matrika v tabeli 2. Nato pa še od slednje odštejemo od vseh elementov najmanjšo vrednost v stolpcih, kar ponazarja matrika v tabeli 3.

	1	2	3	4	5	
1	∞	10	15	40	38	***-10***
2	12	∞	21	32	48	***-12***
3	13	27	∞	48	50	***-13***
4	17	12	8	∞	2	***-2***
5	4	7	24	38	∞	***-4***

Tabela 2.: Odštetje najmanjšega elementa v vrsticah

	1	2	3	4	5
1	∞	0	5	30	28
2	0	∞	9	20	36
3	0	14	∞	35	37
4	15	10	6	∞	0
5	0	3	20	34	∞
	-0	***-0***	***-5***	***-20***	***-0***

Tabela 3.: Odštetje najmanjšega elementa v stolpcih

Ko obe odštetji naredimo, dobimo kot rezultat reducirano matriko v tabeli 4.

	1	2	3	4	5
1	∞	0	0	10	28
2	0	∞	4	0	36
3	0	14	∞	15	37
4	15	10	1	∞	0
5	0	3	15	14	∞

Tabela 4.: Reducirana matrika sosednosti

Začetno spodnjo mejo (SM) dolžine krožne poti dobimo tako, da seštejemo odštete elemente tako v tabeli 2, kot tudi tabeli 3. Pri tem dobimo:

$$SM = 10+12+13+2+4+0+0+5+20+0 = 66 \qquad (8.1.)$$

Izraz (8.1.) pomeni, da v prvotnem grafu na sliki 121 vse krožne poti stanejo vsaj 66 enot, ni pa nujno, da obstaja res takšna krožna pot, ki bi stala 66 enot. To vrednost sedaj vzamemo kot izhodišče v drevesu stanj. Seveda velja, da se bo ta vrednost povečevala, ko bomo napredovali po drevesu stanj in obiskovali vedno več vozlišč.

Tvorjenje delnih krožnih poti na osnovi dveh obiskanih vozlišč

V naslednjem koraku nadaljujemo postopek na osnovi reducirane matrike sosednosti v tabeli 4, pri čemer nadaljujemo izgradnjo krožne poti na osnovi povezav 1-2, 1-3, 1-4 in 1-5, kjer so 1, 2, 3, 4 in 5 seveda vozlišča iz slike 121.

Krožna pot s povezavo 1-2:

Sedaj v matriki v tabeli 4 vse elemente vrstice 1 in stolpca 2 postavimo na ∞, prav tako postavimo na ∞ tudi element druge vrstice in prvega stolpca (povezava 2-1), saj se iz vozlišča 2 še ne smemo vrniti v vozlišče 1. Nato dobljeno matriko zopet reduciramo po vrsticah in stolpcih, kar ponazarja tabela 5, pri čemer v primeru samih ∞ vrednosti tudi odštejemo vrednost 0.

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	*-0*
2	∞	∞	4	0	36	*-0*
3	0	∞	∞	15	37	*-0*
4	15	∞	1	∞	0	*-0*
5	0	∞	15	14	∞	*-0*
	-0	*-0*	***-1***	*-0*	*-0*	

Tabela 5.: Nadaljnje odštetje najmanjšega elementa v vrsticah in stolpcih

Dobimo reducirano matriko za povezavo 1-2, ki je prikazana na tabeli 6.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	**3**	0	36
3	0	∞	**∞**	15	37
4	15	∞	**0**	∞	0
5	0	∞	**14**	14	∞

Tabela 6.: Reducirana matrika za povezavo 1-2

Novo trenutno spodnjo mejo dolžine krožne poti (za povezavo 1-2) dobimo tako, da izrazu (8.1.) prištejemo vsoti odštetih elementov v vrsticah oz. stolpcih tabele 5, ter vrednost elementa iz tabele 4, kjer sta se križali vrstica in stolp ∞ vrednosti - vrednost povezave 1-2 (ki je pa 0!). Tako dobimo:

$$SM = 66+0+0+1+0+0+0+0+0+0+0=67 \qquad (8.2.)$$

Skupaj nas torej vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-2, stane 67 enot ali več.

Krožna pot s povezavo 1-3:

Sedaj v matriki v tabeli 4 vse elemente vrstice 1 in stolpca 3 postavimo na ∞, prav tako postavimo na ∞ tudi element tretje vrstice in prvega stolpca (povezava 3-1), saj se iz vozlišča 3 še ne smemo vrniti v vozlišče 1. Nato dobljeno matriko zopet reduciramo po vrsticah in stolpcih, kar ponazarja tabela 7.

∞	∞	∞	∞	∞	-0
0	∞	∞	0	36	-0
∞	14	∞	15	37	***-14***
15	10	∞	∞	0	-0
0	3	∞	14	∞	-0
-0	-0	-0	-0	-0	

Tabela 7.: Nadaljnje odštetje najmanjšega elementa v vrsticah in stolpcih

Dobimo reducirano matriko za povezavo 1-3, ki je prikazana na tabeli 8.

∞	∞	∞	∞	∞
0	∞	∞	0	36
∞	**0**	∞	**1**	**23**
15	10	∞	∞	0
0	3	∞	14	∞

Tabela 8.: Reducirana matrika za povezavo 1-3

Novo trenutno spodnjo mejo dolžine krožne poti (za povezavo 1-3) dobimo tako, da izrazu (8.1.) prištejemo vsoti odštetih elementov v vrsticah oz. stolpcih tabele 7, ter vrednost elementa iz tabele 4, kjer sta se križali vrstica in stolp ∞ vrednosti - vrednost povezave 1-3 (ki je pa 0!). Tako dobimo:

$$SM = 66+0+0+14+0+0+0+0+0+0+0=80 \qquad (8.3.)$$

Skupaj nas torej vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-3, stane 80 enot ali več, kar je dražje kot pri povezavi 1-2 (le 67 enot).

Krožna pot s povezavo 1-4:

Sedaj v matriki v tabeli 4 vse elemente vrstice 1 in stolpca 4 postavimo na ∞, prav tako postavimo na ∞ tudi element četrte vrstice in prvega stolpca (povezava 4-1), saj se iz vozlišča 4 še ne smemo vrniti v vozlišče 1. Nato dobljeno matriko zopet reduciramo po vrsticah in stolpcih, kar ponazarja tabela 9.

∞	∞	∞	∞	∞	-0
0	∞	4	∞	36	-0
0	14	∞	∞	37	-0
∞	10	1	∞	0	-0
0	3	15	∞	∞	-0
-0	***-3***	***-1***	-0	-0	

Tabela 9.: Nadaljnje odštetje najmanjšega elementa v vrsticah in stolpcih

Dobimo reducirano matriko za povezavo 1-4, ki je prikazana na tabeli 10.

∞	∞	∞	∞	∞
0	∞	**3**	∞	36
0	**11**	**∞**	∞	37
∞	**7**	**0**	∞	0
0	**0**	**14**	∞	∞

Tabela 10.: Reducirana matrika za povezavo 1-4

Novo trenutno spodnjo mejo dolžine krožne poti (za povezavo 1-4) dobimo tako, da izrazu (8.1.) prištejemo vsoti odštetih elementov v vrsticah oz. stolpcih tabele 9, ter vrednost elementa iz tabele 4, kjer sta se križali vrstica in stolp ∞ vrednosti - vrednost povezave 1-4 (ki je 10!). Tako dobimo:

$$SM = 66+0+3+1+0+0+0+0+0+0+0+10=80 \qquad (8.4.)$$

Skupaj nas torej vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-4, stane 80 enot ali več, torej enako kot pri povezavi 1-3.

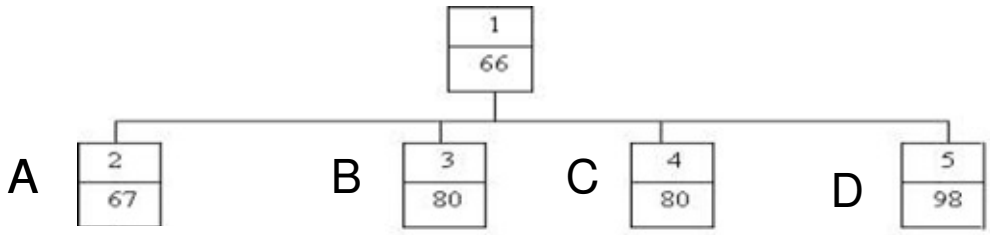
Krožna pot s povezavo 1-5:

Postopek je popolnoma enak kot pri povezavah 1-2, 1-3 in 1-4. Kot se izkaže, dobimo naslednji rezultat za oceno spodnje meje te delne krožne poti:

$$SM = 66+4+28=98 \qquad (8.5.)$$

kjer je vrednost 4 doprinesla vsota odštetih elementov v vrsticah oz. stolpcih, vrednost 28 pa je vrednost elementa iz tabele 4, kjer se križata vrstica in stolp ∞ vrednosti - vrednost povezave 1-5.

Če primerjamo izraze (8.2.), (8.3.), (8.4.) in (8.5.), opazimo, da je med delnimi krožnimi potmi, ki jih tvorijo povezave 1-2 (67 enot), 1-3 (80 enot), 1-4 (80 enot) in 1-5 (98 enot), očitno najobetavnejša povezava 1-2, saj stane 67 enot ali več. Na sliki 122 je ilustrirano drevo stanj, če prodremo v globino do 2. nivoja, to je do dveh obiskanih vozlišč, pri čemer upoštevamo izraze (8.1.), (8.2.), (8.3.), (8.4.) in (8.5.).



Slika 121.: Trenutno drevo stanj, kjer so označene ocene spodnje meje delne krožne poti pri dveh obiskanih vozliščih (dva nivoja drevesa stanj)

Kot je razvidno iz slike 121, so v kvadratih zgoraj označene številke vozlišč, ki smo jih obiskali, spodaj pa ocena SM, ki govori o ceni, ki jo moramo plačati, da obiščemo posamezne kombinacije dveh vozlišč pri izhodišču iz vozlišča 1.

Načeloma moramo sedaj v trenutnem drevesu stanj na sliki 121 nadaljevati s prodiranjem v njegovo globino, tvoriti še njegove nadaljnje nivoje, pri tem dodajati še nadaljnja na novo obiskana vozlišča, ter tako graditi nadaljnjo oceno spodnje meje dolžine krožne poti.

Denimo najprej nadaljujemo s prodiranjem v tretji nivo drevesa preko bloka A na sliki 121, torej nadaljujemo iz povezave 1-2 (saj zaenkrat zgleda naobetavnejša) in ustvarimo nove povezave: 1-2-3, 1-2-4 in 1-2-5.

Tvorjenje delnih krožnih poti na osnovi treh obiskanih vozlišč z izhodiščem iz povezave 1-2

Krožna pot s povezavo 1-2-3:

Sedaj v matriki v tabeli 6 vse elemente vrstice 2 in stolpca 3 postavimo na ∞, prav tako postavimo na ∞ tudi element tretje vrstice in prvega stolpca (povezava 3-1), saj se iz vozlišča 3 še ne smemo vrniti v vozlišče 1. Nato dobljeno matriko zopet reduciramo po vrsticah in stolpcih, kar ponazarja tabela 11.

∞	∞	∞	∞	∞	
∞	∞	∞	∞	∞	
∞	∞	∞	15	37	***-15***
15	∞	∞	∞	0	
0	∞	∞	14	∞	

Tabela 11.: Reduciranje matrike pri krožni poti s povezavo 1-2-3

Če najprej odštejemo najmanjši element v vseh vrsticah tabele 11, dobimo same ničle pri tem odštetju, razen vrednosti 15 pri 3. vrstici. Če nato odštejemo najmanjši element še v vseh stolpcih, pa dobimo same ničle pri tem odštetju (in nič več ne spremenimo).

Tako dobimo reducirano matriko za povezavo 1-2-3, ki je prikazana na tabeli 12.

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	**0**	**22**
15	∞	∞	∞	0
0	∞	∞	14	∞

Tabela 12.: Reducirana matrika za povezavo 1-2-3

Novo trenutno spodnjo mejo dolžine krožne poti (za povezavo 1-2-3) dobimo tako, da izrazu (8.2.) prištejemo vsoti odštetih elementov v vrsticah oz. stolpcih tabele 11 (le vrednost 15!), ter vrednost elementa iz tabele 6, kjer je prišlo do novega križanja vrstice in stolpa ∞ vrednosti - vrednost povezave 2-3 (ki je 3!). Tako dobimo:

$$SM = 67+15+3=85 \qquad (8.6.)$$

Skupaj nas torej vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-2-3, stane 85 enot ali več.

Krožna pot s povezavo 1-2-4:

Sedaj v matriki v tabeli 6 vse elemente vrstice 2 in stolpca 4 postavimo na ∞, prav tako postavimo na ∞ tudi element četrte vrstice in prvega stolpca (povezava 4-1), saj se iz vozlišča 4 še ne smemo vrniti v vozlišče 1. Nato dobljeno matriko zopet reduciramo po vrsticah in stolpcih, kar ponazarja tabela 13.

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	37
∞	∞	0	∞	0
0	∞	14	∞	∞

Tabela 13.: Reduciranje matrike pri krožni poti s povezavo 1-2-4

Če najprej odštejemo najmanjši element v vseh vrsticah tabele 13, dobimo same ničle pri tem odštetju. Če nato odštejemo najmanjši element še v vseh stolpcih, zopet dobimo same ničle pri tem odštetju. Torej tabela 13 že kar predstavlja reducirano matriko za povezavo 1-2-4.

Novo trenutno spodnjo mejo dolžine krožne poti (za povezavo 1-2-4) dobimo tako, da izrazu (8.2.) prištejemo vsoti odštetih elementov v vrsticah oz. stolpcih tabele 13 (ki je pa 0!), ter vrednost elementa iz tabele 6, kjer je prišlo do novega križanja vrstice in stolpa ∞ vrednosti - vrednost povezave 2-4 (ki je pa tudi 0!). Tako dobimo:

$$SM = 67+0+0=67 \qquad (8.7.)$$

Skupaj nas torej vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-2-4, stane 67 enot ali več. Če primerjamo izraza (8.2.) in (8.7.), vidimo, da nas dodajanje novega vozlišča 4 povezavi 1-2 ni prav nič stalo, saj je SM še vedno 67.

Krožna pot s povezavo 1-2-5:

Sedaj v matriki v tabeli 6 vse elemente vrstice 2 in stolpca 5 postavimo na ∞, prav tako postavimo na ∞ tudi element pete vrstice in prvega stolpca (povezava 5-1), saj se iz vozlišča 5 še ne smemo vrniti v vozlišče 1. Nato dobljeno matriko zopet reduciramo po vrsticah in stolpcih, kar ponazarja tabela 14.

∞	∞	∞	∞	∞	
∞	∞	∞	∞	∞	
0	∞	∞	15	∞	
15	∞	0	∞	∞	
∞	∞	14	14	∞	***-14***

Tabela 14.: Reduciranje matrike pri krožni poti s povezavo 1-2-5

Če najprej odštejemo najmanjši element v vseh vrsticah tabele 14, dobimo same ničle pri tem odštetju, razen vrednosti 14 pri 5. vrstici. Če nato odštejemo najmanjši element še v vseh stolpcih, pa dobimo same ničle pri tem odštetju (in nič več ne spremenimo).

Tako dobimo reducirano matriko za povezavo 1-2-5, ki je prikazana na tabeli 15.

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	15	∞
15	∞	0	∞	∞
∞	∞	**0**	**0**	∞

Tabela 15.: Reducirana matrika za povezavo 1-2-5

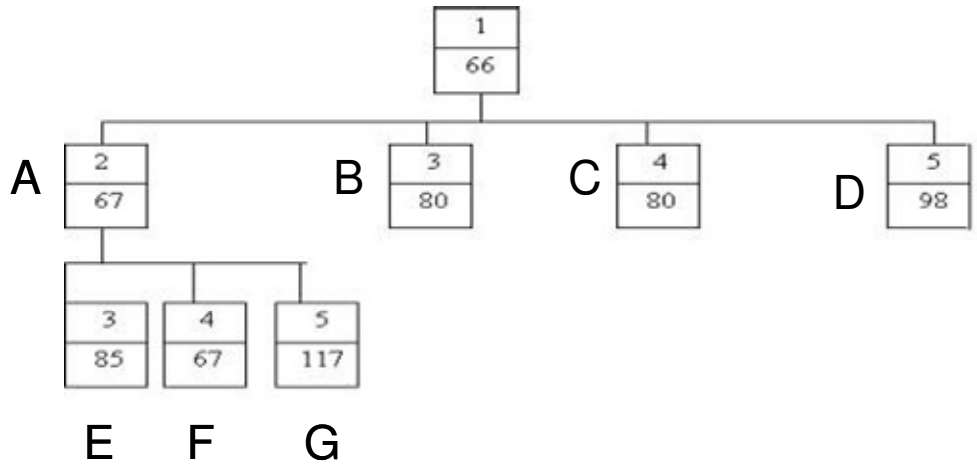
Novo trenutno spodnjo mejo dolžine krožne poti (za povezavo 1-2-5) dobimo tako, da izrazu (8.2.) prištejemo vsoti odštetih elementov v vrsticah oz. stolpcih tabele 14 (le vrednost 14!), ter vrednost elementa iz tabele 6, kjer je prišlo do novega križanja vrstice in stolpa ∞ vrednosti - vrednost povezave 2-5 (ki je 36!). Tako dobimo:

$$SM = 67+14+36=117 \qquad (8.8.)$$

Skupaj nas torej vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-2-5, stane 117 enot ali več.

Na sliki 122 je ilustrirano drevo stanj, če prodremo v globino do 3. nivoja (preko bloka A na sliki 121), to je do treh obiskanih vozlišč, pri čemer dodatno upoštevamo izraze (8.6.), (8.7.) in (8.8.).

Kot je razvidno iz slike 122, smo s pravkar opisanim manevrom ustvarili tri nove bloke E, F in G. Izmed povezav 1-2-3, 1-2-4 in 1-2-5 pa je očitno najbolj obetavna povezava 1-2-4, saj je cena zanjo 67 enot (za povezavi 1-2-3 in 1-2-5 pa bi bila cena 85 oz. 117 enot).



Slika 122.: Trenutno drevo stanj, kjer so označene ocene spodnje meje delne krožne poti pri treh obiskanih vozliščih, pot preko bloka A (trije nivoji drevesa stanj)

Zato si v nadaljevanju še poglejmo, kakšne rezultate bi dobili, če bi izračune nadaljujevali s prodiranjem v četrti nivo drevesa preko bloka F na sliki 122, torej če bi nadaljevali iz povezave 1-2-4 (saj zaenkrat zgleda naobetavnejša) in ustvarili novi povezavi: 1-2-4-3 in 1-2-4-5.

Tvorjenje delnih krožnih poti na osnovi štirih obiskanih vozlišč z izhodiščem iz povezave 1-2-4

Krožna pot s povezavo 1-2-4-3:

Sedaj v matriki v tabeli 13 vse elemente vrstice 4 in stolpca 3 postavimo na ∞, prav tako postavimo na ∞ tudi element tretje vrstice in prvega stolpca (povezava 3-1), saj se iz vozlišča 3 še ne smemo vrniti v vozlišče 1. Nato dobljeno matriko zopet reduciramo po vrsticah in stolpcih, kar ponazarja tabela 16.

∞	∞	∞	∞	∞	
∞	∞	∞	∞	∞	
∞	∞	∞	∞	37	***-37***
∞	∞	∞	∞	∞	
0	∞	∞	∞	∞	

Tabela 16.: Reduciranje matrike pri krožni poti s povezavo 1-2-4-3

Če najprej odštejemo najmanjši element v vseh vrsticah tabele 16, dobimo same ničle pri tem odštetju, razen vrednosti 37 pri 3. vrstici. Če nato odštejemo najmanjši element še v vseh stolpcih, pa dobimo same ničle pri tem odštetju (in nič več ne spremenimo).

Tako dobimo reducirano matriko za povezavo 1-2-4-3, ki je prikazana na tabeli 17.

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	**0**
∞	∞	∞	∞	∞
0	∞	∞	∞	∞

Tabela 17.: Reducirana matrika za povezavo 1-2-4-3

Novo trenutno spodnjo mejo dolžine krožne poti (za povezavo 1-2-4-3) dobimo tako, da izrazu (8.7.) prištejemo vsoti odštetih elementov v vrsticah oz. stolpcih tabele 16 (le vrednost 37!), ter vrednost elementa iz tabele 13, kjer je prišlo do novega križanja vrstice in stolpa ∞ vrednosti - vrednost povezave 4-3 (ki je pa 0!). Tako dobimo:

$$SM = 67+37+0=104 \qquad (8.9.)$$

Skupaj nas torej vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-2-4-3, stane 104 enote ali več.

Krožna pot s povezavo 1-2-4-5:

Sedaj v matriki v tabeli 13 vse elemente vrstice 4 in stolpca 5 postavimo na ∞, prav tako postavimo na ∞ tudi element pete vrstice in prvega stolpca (povezava 5-1), saj se iz vozlišča 5 še ne smemo vrniti v vozlišče 1. Nato dobljeno matriko zopet reduciramo po vrsticah in stolpcih, kar ponazarja tabela 18.

∞	∞	∞	∞	∞	
∞	∞	∞	∞	∞	
0	∞	∞	∞	∞	
∞	∞	∞	∞	∞	
∞	∞	14	∞	∞	**-14**

Tabela 18.: Reduciranje matrike pri krožni poti s povezavo 1-2-4-5

Če najprej odštejemo najmanjši element v vseh vrsticah tabele 18, dobimo same ničle pri tem odštetju, razen vrednosti 14 pri 5. vrstici. Če nato odštejemo najmanjši element še v vseh stolpcih, pa dobimo same ničle pri tem odštetju (in nič več ne spremenimo).

Tako dobimo reducirano matriko za povezavo 1-2-4-5, ki je prikazana na tabeli 19.

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	**0**	∞	∞

Tabela 19.: Reducirana matrika za povezavo 1-2-4-5

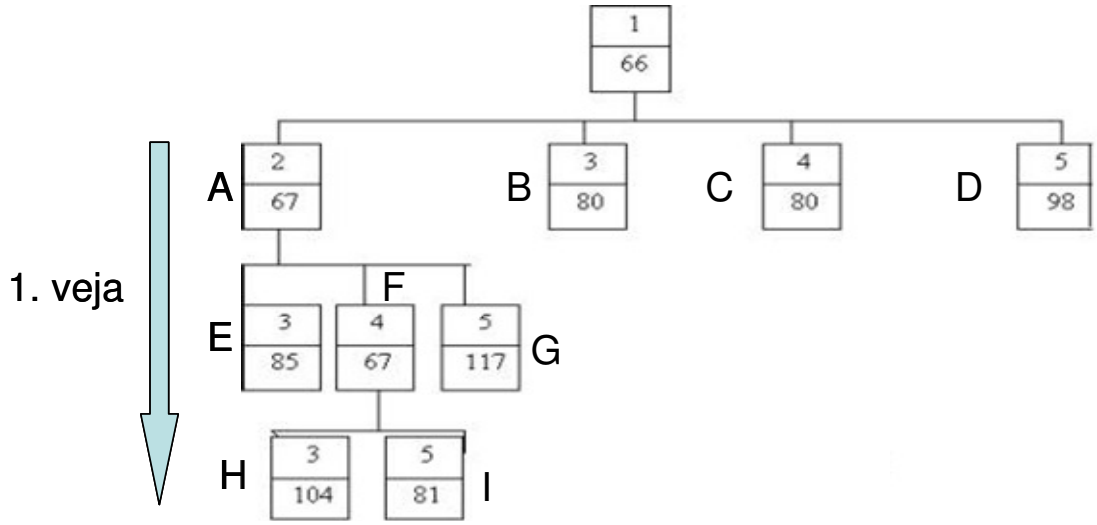
Novo trenutno spodnjo mejo dolžine krožne poti (za povezavo 1-2-4-5) dobimo tako, da izrazu (8.7.) prištejemo vsoti odštetih elementov v vrsticah oz. stolpcih tabele 18 (le vrednost 14!), ter vrednost elementa iz tabele 13, kjer je prišlo do novega križanja vrstice in stolpa ∞ vrednosti - vrednost povezave 4-5 (ki je pa 0!). Tako dobimo:

$$SM = 67+14+0=81 \qquad (8.10.)$$

Skupaj nas torej vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-2-4-5, stane 81 enot ali več.

Na sliki 123 je ilustrirano trenutno drevo stanj, če prodremo v globino do 4. nivoja (preko bloka F na sliki 122), to je do štirih obiskanih vozlišč, pri čemer dodatno upoštevamo izraza (8.9.) in (8.10.).

Kot je razvidno iz slike 123, smo s pravkar opisanim manevrom ustvarili dva nova bloka H in I. Izmed povezav 1-2-4-3 in 1-2-4-5 pa je očitno bolj obetavna povezava 1-2-4-5, saj je cena zanjo 81 enot (za povezavo 1-2-4-3 pa bi bila cena 104 enote).



Slika 123.: Trenutno drevo stanj, kjer so označene ocene spodnje meje delne krožne poti pri štirih obiskanih vozliščih, pot preko blokov A in F (štirje nivoji drevesa stanj)

Žal se izkaže, da nadaljnji razvoj po 1. veji slike 123 ne bi dal najboljših rezultatov. Zato se moramo v nadaljevanju odločiti za prodiranje vglobino drevesa po kakšni drugi veji, denimo se odločimo, da bomo nadaljevali z razvojem krožne poti preko bloka B in povezave 1-3.

Tvorjenje delnih krožnih poti na osnovi treh obiskanih vozlišč z izhodiščem iz povezave 1-3

Krožna pot s povezavo 1-3-2:

Na osnovi enakih postopkov kot pri tvorjenju prejšnjih delnih krožnih poti bi dobili reducirano matriko, ki je prikazana v tabeli 20.

∞	∞	∞	∞	∞
∞	∞	∞	0	36
∞	∞	∞	∞	∞
15	∞	∞	∞	0
0	∞	∞	14	∞

Tabela 20.: Reducirana matrika za povezavo 1-3-2

Pri tem bi nas stala, kot se izkaže, vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-3-2, naslednjo vrednost:

$$SM = 80+0+0=80 \qquad (8.11.)$$

torej ne bi bila nič dražja od povezave 1-3 (glej izraz (8.3.)).

Krožna pot s povezavo 1-3-4:

Na osnovi enakih postopkov kot pri tvorjenju prejšnjih delnih krožnih poti bi dobili reducirano matriko, ki je prikazana v tabeli 21.

∞	∞	∞	∞	∞
0	∞	∞	∞	36
∞	∞	∞	∞	∞
∞	10	∞	∞	0
0	3	∞	∞	∞

-3

∞	∞	∞	∞	∞
0	∞	∞	∞	36
∞	∞	∞	∞	∞
∞	**7**	∞	∞	0
0	**0**	∞	∞	∞

Tabela 21.: Reducirana matrika za povezavo 1-3-4

Pri tem bi nas stala, kot se izkaže, vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-3-4, naslednjo vrednost:

$$SM = 80 + 3 + 1 = 84 \qquad (8.12.)$$

torej bi bila za 4 enote dražja od povezave 1-3 (glej izraz (8.3.)).

Krožna pot s povezavo 1-3-5:

Na osnovi enakih postopkov kot pri tvorjenju prejšnjih delnih krožnih poti bi dobili reducirano matriko, ki je prikazana v tabeli 22.

∞	∞	∞	∞	∞	
0	∞	∞	0	∞	
∞	∞	∞	∞	∞	
15	10	∞	∞	∞	**-10**
∞	3	∞	14	∞	**-3**

∞	∞	∞	∞	∞
0	∞	∞	0	∞
∞	∞	∞	∞	∞
5	**0**	∞	∞	∞
∞	**0**	∞	**11**	∞

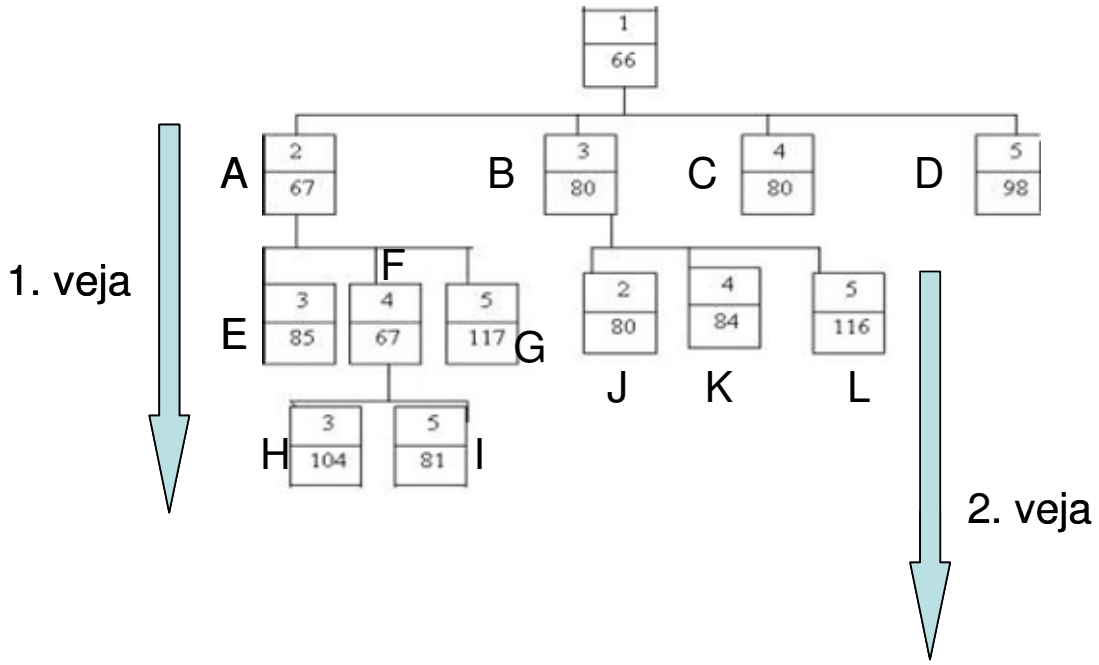
Tabela 22.: Reducirana matrika za povezavo 1-3-5

Pri tem bi nas stala, kot se izkaže, vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-3-5, naslednjo vrednost:

$$SM = 80 + 13 + 23 = 116 \qquad (8.13.)$$

torej bi bila za 36 enot dražja od povezave 1-3 (glej izraz (8.3.)).
Na sliki 124 je ilustrirano trenutno drevo stanj, če prodremo v globino do 3. nivoja (preko bloka B na sliki 123), to je do treh obiskanih vozlišč preko 2. veje drevesa, pri čemer dodatno upoštevamo izraze (8.11.), (8.12.) in (8.13.).

Kot je razvidno iz slike 124, smo s pravkar opisanim manevrom ustvarili tri nove bloke J, K in L. Izmed povezav 1-3-2, 1-3-4 in 1-3-5 pa je očitno najbolj obetavna povezava 1-3-2, saj je cena zanjo le 80 enot.



Slika 124.: Trenutno drevo stanj, kjer so označene ocene spodnje meje delne krožne poti. Razvoj 2. veje drevesa preko bloka B.

Zato si v nadaljevanju še poglejmo, kakšne rezultate bi dobili, če bi izračune nadaljevali s prodiranjem v četrti nivo drevesa preko bloka J na sliki 124, torej če bi nadaljevali iz povezave 1-3-2 (saj zaenkrat zgleda naobetavnejša) in ustvarili novi povezavi: 1-3-2-4 in 1-3-2-5.

Tvorjenje delnih krožnih poti na osnovi štirih obiskanih vozlišč z izhodiščem iz povezave 1-3-2

Krožna pot s povezavo 1-3-2-4:

Na osnovi enakih postopkov kot pri tvorjenju prejšnjih delnih krožnih poti bi dobili reducirano matriko, ki je prikazana v tabeli 23.

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	0
0	∞	∞	∞	∞

Tabela 23.: Reducirana matrika za povezavo 1-3-2-4

Pri tem bi nas stala, kot se izkaže, vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-3-2-4, naslednjo vrednost:

$$SM = 80 + 0 + 0 = 80 \qquad (8.14.)$$

torej ne bi bila nič dražja od povezave 1-3 (glej izraz (8.3.)) oz. povezave 1-3-2 (glej izraz (8.11.)).

Krožna pot s povezavo 1-3-2-5:

Na osnovi enakih postopkov kot pri tvorjenju prejšnjih delnih krožnih poti bi dobili reducirano matriko, ki je prikazana v tabeli 24.

∞	∞	∞	∞	∞	
∞	∞	∞	∞	∞	
∞	∞	∞	∞	∞	
15	∞	∞	∞	∞	**-15**
∞	∞	∞	14	∞	**-14**

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	∞
∞	∞	∞	**0**	∞

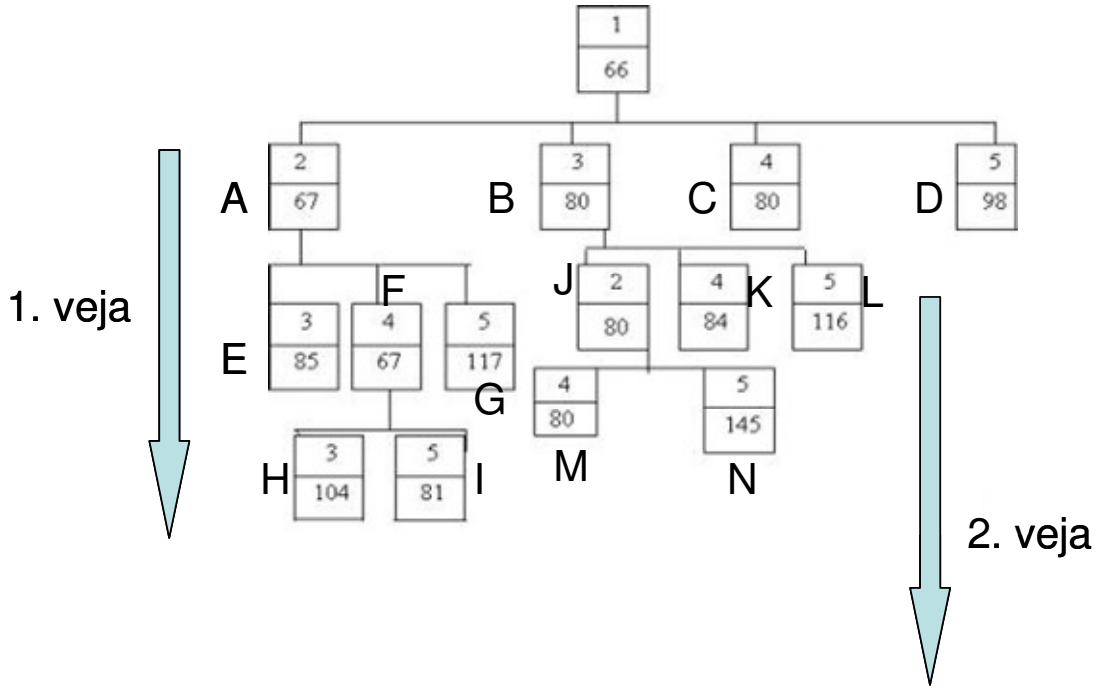
Tabela 24.: Reducirana matrika za povezavo 1-3-2-5

Pri tem bi nas stala, kot se izkaže, vsaka krožna pot v našem problemu, ki vsebuje povezavo 1-3-2-5, naslednjo vrednost:

$$SM = 80 + 29 + 36 = 145 \qquad (8.15.)$$

Na sliki 125 je ilustrirano trenutno drevo stanj, če prodremo v globino do 4. nivoja (preko bloka J na sliki 124), to je do štirih obiskanih vozlišč preko 2. veje drevesa, pri čemer dodatno upoštevamo izraza (8.14.) in (8.15.).

Kot je razvidno iz slike 125, smo s pravkar opisanim manevrom ustvarili dva nova bloka M in N. Izmed povezav 1-3-2-4 in 1-3-2-5 pa je očitno bolj obetavna povezava 1-3-2-4, saj je cena zanjo še vedno le 80 enot.



Slika 125.: Trenutno drevo stanj, kjer so označene ocene spodnje meje delne krožne poti. Nadaljnji razvoj 2. veje drevesa preko bloka J.

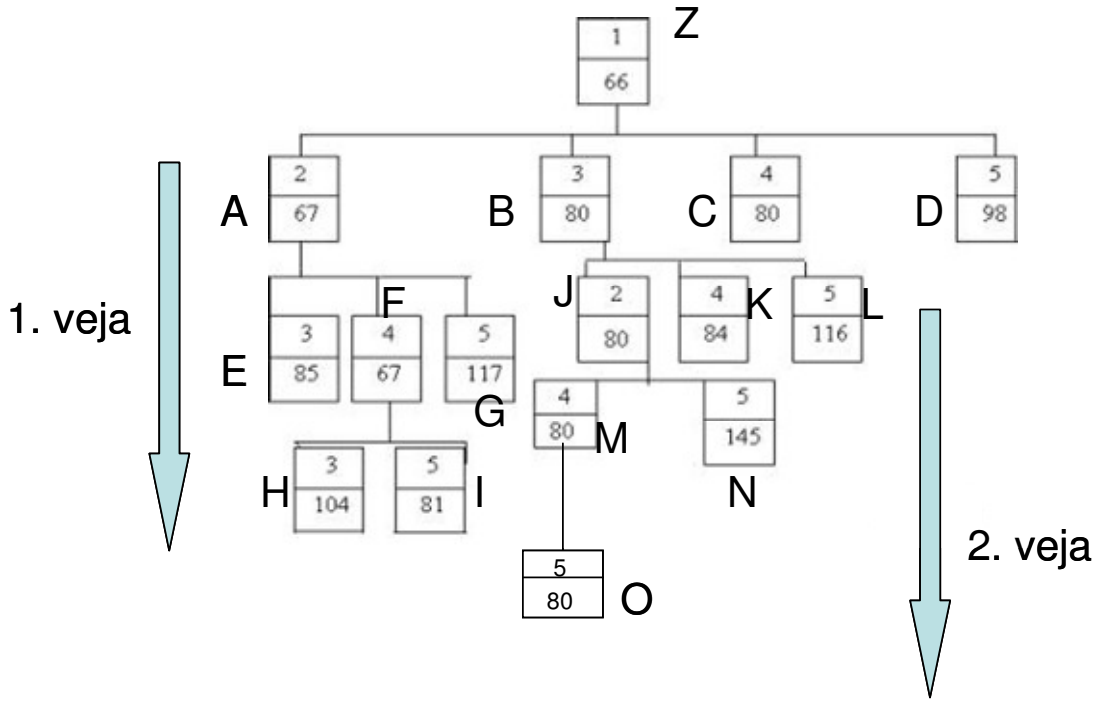
Zato si v nadaljevanju še poglejmo, kakšne rezultate bi dobili, če bi izračune nadaljevali s prodiranjem v peti, zadnji nivo drevesa (saj je 5 vozlišč!) preko bloka M na sliki 125, torej če bi nadaljevali iz povezave 1-3-2-4 (saj zgleda naobetavnejša) in ustvarili še zadnjo novo povezavo: 1-3-2-4-5.

Tvorjenje končne krožne poti na osnovi petih obiskanih vozlišč z izhodiščem iz povezave 1-3-2-4

Krožna pot s povezavo 1-3-2-4-5:

Če opazujemo tabelo 23, kjer so same ∞ vrednosti in dve ničli (četrta vrstica, peti stolp, ter peta vrstica, prvi stolp), vidimo, da nas prehod iz 4. v 5. vozlišče ne bo očitno nič več stal, prav tako tudi ne prehod iz 5. vozlišča nazaj v 1. vozlišče (kar je tokrat dovoljeno, saj smo že obšli vsa vozlišča). SM torej ostaja enaka kot v izrazu (8.14.), torej 80 enot.

Na sliki 126 je ilustrirano trenutno drevo stanj, ki ga dobimo na osnovi tega zadnjega manevra, ki nam je dal kot rezultat izgrajeno 2. vejo drevesa, ki vključuje tudi blok O in končno povezavo 1-3-2-4-5 preko blokov Z-B-J-M-0.

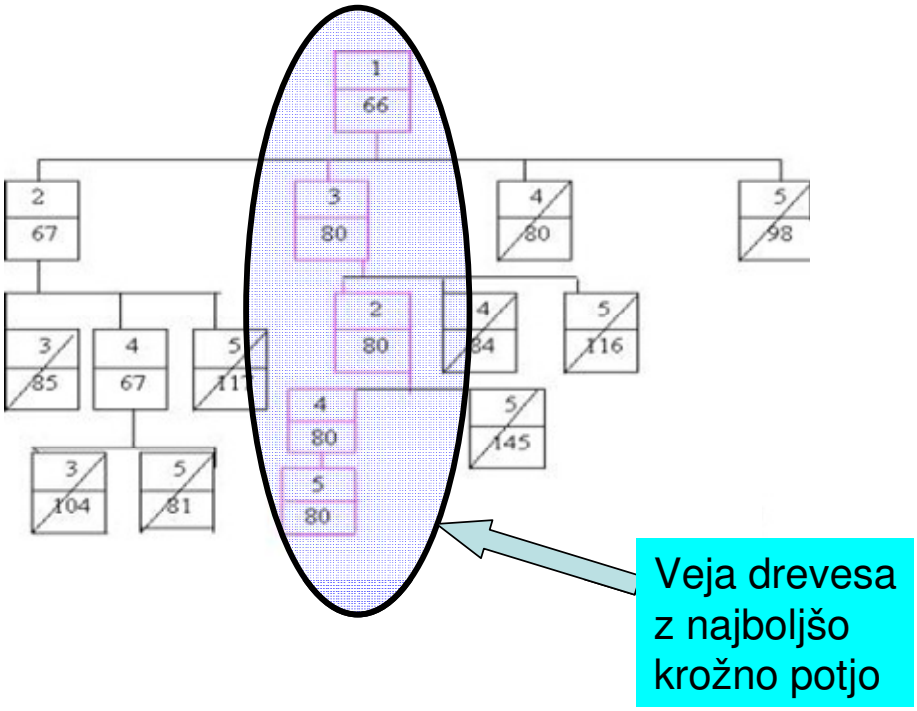


Slika 126.: Trenutno drevo stanj, kjer so označene ocene spodnje meje delne krožne poti. Dokončan razvoj 2. veje drevesa preko bloka M.

Če sedaj opazujemo vse bloke na sliki 126, odkoder bi se še dalo nadaljevati z razvojem krožne poti do petega nivoja drevesa (bloki E, G, H, I, K, L, D, N), vidimo, da bi nas to gotovo vedno stalo dražje kot pri dobljeni povezavi 1-3-2-4-5 s končnim blokom O (kjer

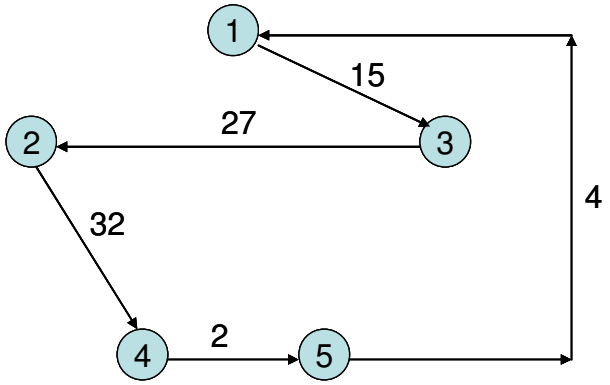
je SM = 80). Tudi v primeru, če bi krožno pot gradili iz bloka C, gotovo ne bi mogli priti do boljšega rezultata, kvečjemu slabšega. Zato sklepamo, da je dobljena krožna pot 1-3-2-4-5-1 (če se še vrnemo iz 5 nazaj v izhodiščno vozlišče 1), gotovo najboljša oz. najcenejša.

Na koncu nas seveda zanima samo vrednost te najcenejše krožne poti in pripadajoča veja v drevesu stanj, zato vse ostale veje odstranimo iz drevesa stanj. Torej, ko dokončno razvijemo drevo stanj oz. ko pridemo do najcenejše rešitve, ki obide vsa vozlišča, oklestimo iz drevesa »slabe veje«. Tako pridemo do končnega izgleda drevesa stanj, prikazanega na sliki 127, kjer so oklestene slabe veje prečrtane, veja z najboljšo krožno potjo pa je posebej poudarjena.



Slika 127.: Končno drevo stanj, kjer smo oklestili slabe veje in pustili le vejo z najboljšo krožno potjo

Na sliki 128 pa je podana rešitev za obravnavani problem trgovskega potnika s slike 121, ki nam da najcenejšo krožno pot preko vseh vozlišč.



$$15 + 27 + 32 + 2 + 4 = 80$$

Slika 128.: Rešitev za obravnavani problem trgovskega potnika

8.2 Reševanje problema PTP s pomočjo metode najbližjega soseda

Algoritem "najbližjega soseda" spada med razred algoritmov za konstrukcijo cikla. Avtorji algoritma so Rosenkrantz, Stearns in Lewis (1977). Algoritem je sestavljen iz naslednjih korakov [15]:

KORAK 1: Poljubno izberi začetno vozlišče cikla.
KORAK 2: Poišči neuporabljeno vozlišče, ki je najbližje zadnjemu vozlišču, vključenem v cikel, ter ga tudi vključi v cikel.
KORAK 3: Ponavljaj korak 2, dokler se ne vključijo vsa vozlišča v cikel, ter se vrni v začetno vozlišče.

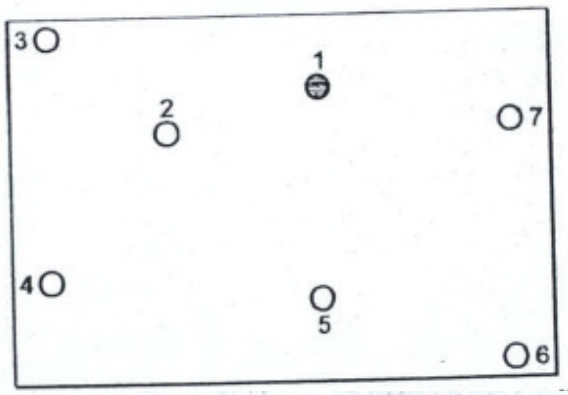
Delovanje algoritma bomo osvetlili na naslednjem primeru.

Primer:

Za graf, katerega vozlišča in matrika sosednosti so prikazana na sliki 129, poišči optimalen cikel trgovskega potnika s pomočjo algoritma najbližjega soseda. Kot izhodišče izberi vozlišče 1.

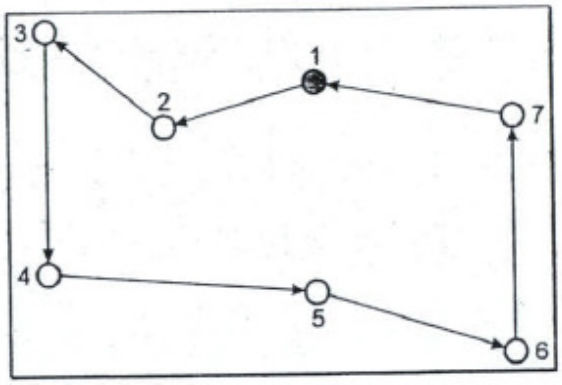
Rešitev:

Začnemo v vozlišču 1 in opazujemo sliko 129. Vozlišču 1 je najbližje vozlišče 2 (utež 75), zato slednjega vključimo v cikel, ki se sedaj glasi: (1, 2). Vozlišču 2 je najbližje vozlišče 3 (utež 90), zato slednjega tudi vključimo v cikel, ki se sedaj glasi: (1, 2, 3). Vozlišču 3 je najbližje vozlišče 4 (utež 150), zato slednjega tudi vključimo v cikel, ki se sedaj glasi: (1, 2, 3, 4). Na podoben način postopek nadaljujemo, ter postopoma vključimo še preostala vozlišča po vrstnem redu: vozlišče 5, vozlišče 6 in vozlišče 7. Na koncu postopka še spojimo vozlišči 7 in 1, torej se vrnemo v izhodiščno vozlišče. Tako smo ustvarili cikel, katerega končna oblika se glasi: (1, 2, 3, 4, 5, 6, 7, 1). Dobljeni cikel je prikazan na sliki 130. Poudarimo še, da pri iskanju najbližjega soseda nismo gledali razdalj do vozlišč, ki smo jih že vključili v cikel, pač pa le do še nevključenih vozlišč.



	1	2	3	4	5	6	7
1	0	75	135	165	135	180	90
2	75	0	90	105	135	210	150
3	135	90	0	150	210	300	210
4	165	105	150	0	135	210	210
5	135	135	210	135	0	90	105
6	180	210	300	210	90	0	120
7	90	150	210	210	105	120	0

Slika 129.: Vozlišča in matrika sosednosti danega grafa

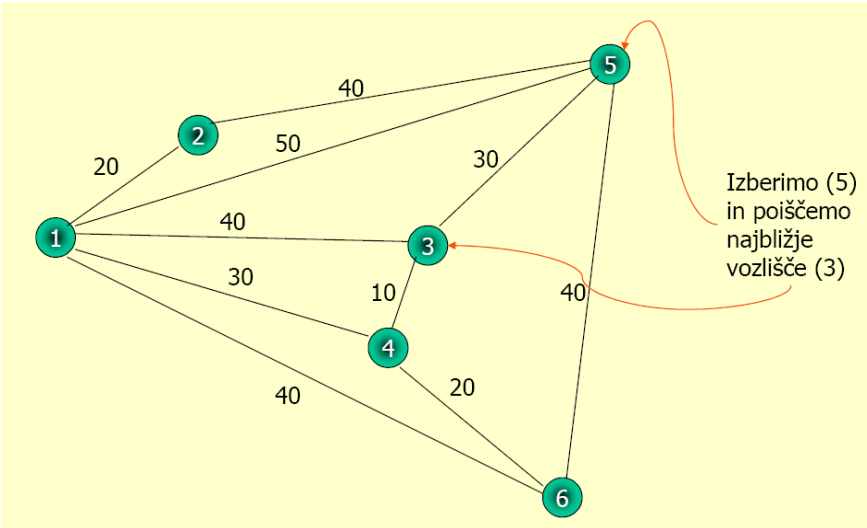


Slika 130.: Rešitev PTP, kjer cikel tvorimo s pomočjo metode najbližjega soseda

Delovanje algoritma osvetlimo še na enem primeru.

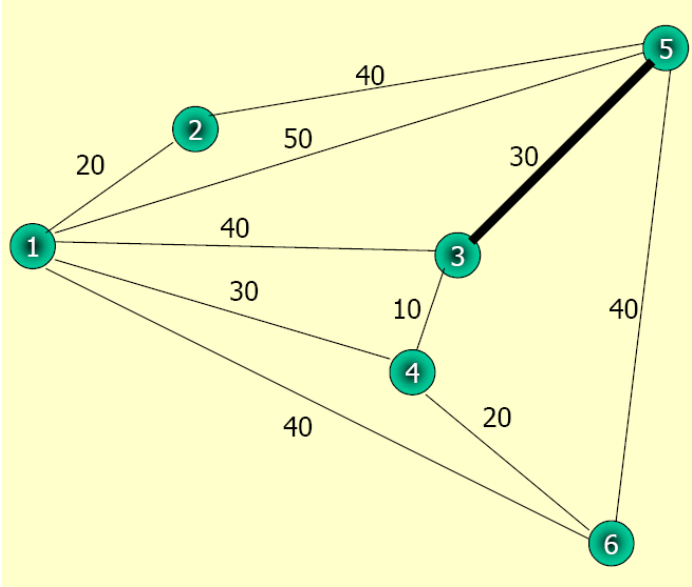
Primer:

Za graf, prikazan na sliki 131, poišči optimalen cikel trgovskega potnika s pomočjo algoritma najbližjega soseda. Kot izhodišče izberi vozlišče 5.



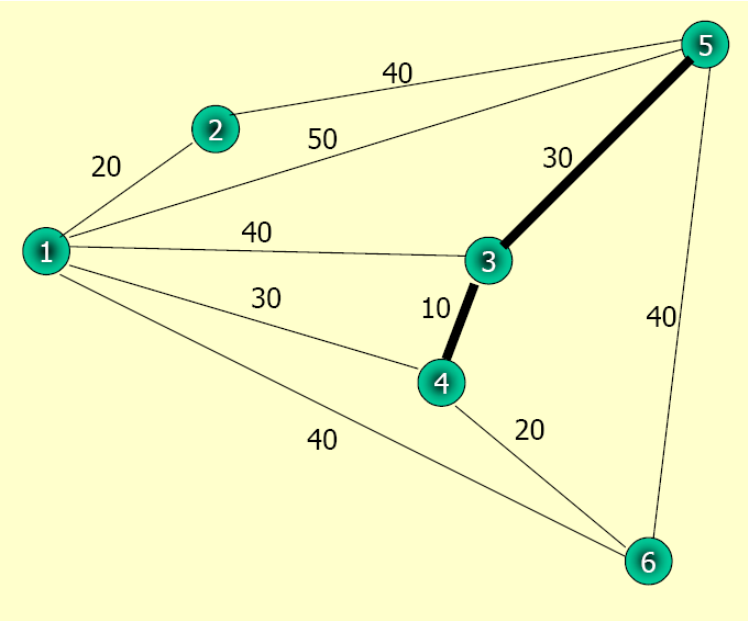
Slika 131.: Dani graf, kjer želimo poiskati rešitev PTP s pomočjo metode najbližjega soseda

Na grafu na sliki 131 poiščemo vozlišču 5 najbližje vozlišče. Očitno je to vozlišče 3, ki ga vključimo v cikel in dobimo sliko 132.



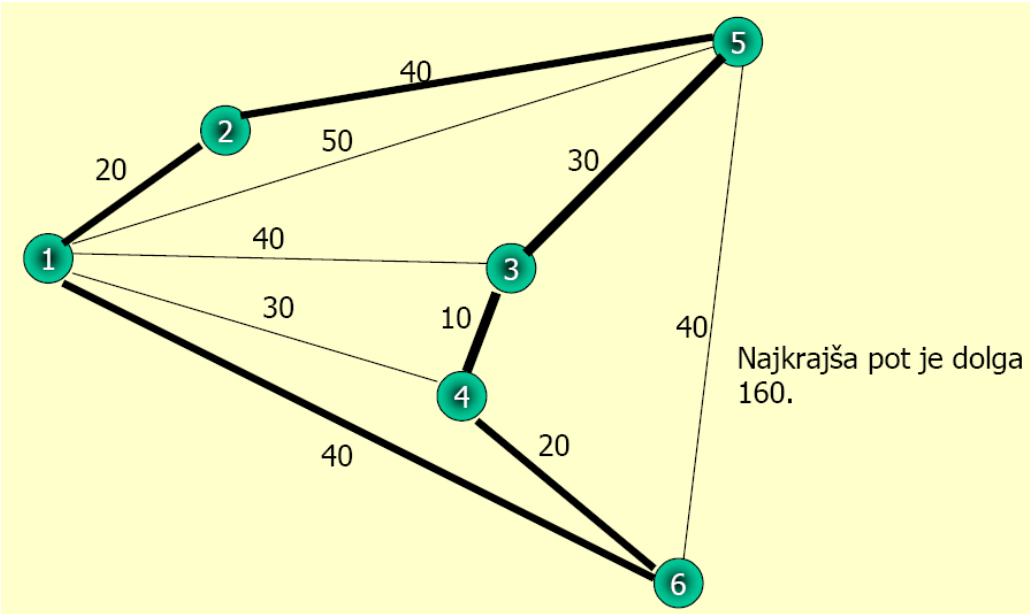
Slika 132.: Vključitev vozlišča 3 v cikel

Na grafu na sliki 132 nato poiščemo vozlišču 3 najbližje (neuporabljeno!) vozlišče. Očitno je to vozlišče 4, ki ga vključimo v cikel in dobimo sliko 133.



Slika 133.: Vključitev vozlišča 4 v cikel

Postopek nadaljujemo, dokler ne obiščemo vseh preostalih vozlišč, ter se vrnemo v izhodiščno vozlišče. Rezultat za dobljeni cikel je prikazan na sliki 134.



Slika 134.: Rešitev PTP, kjer cikel tvorimo s pomočjo metode najbližjega soseda

8.3 Reševanje problemov PTP s pomočjo algoritmov vrivanja (Insertion algorithms)

Rosenkrantz, Stearns in Lewis so razvili večje število hevrističnih algoritmov za reševanje PTP problemov. Med temi algoritmi igrajo pomembno vlogo tudi takoimenovani algoritmi vrivanja (Insertion algorithms) [15]. Pri teh algoritmih se v k-tem izvajanju algoritma preučuje delna pot trgovskega potnika, ki je sestavljena iz k vozlišč. Problem je v tem, kam v to pot vriniti naslednje, še neuporabljeno vozlišče, ter katero vozlišče naj bo to. Tovrstna odločitev se izvede v takoimenovanem "koraku vrivanja". Glede na način, kam vriniti katero vozlišče, so omenjeni avtorji razvili več algoritmov, kot npr.: Algoritem najbližjega vrivanja, Algoritem najcenejšega vrivanja, Algoritem poljubnega vrivanja, Algoritem najdaljšega vrivanja, Algoritem najhitrejšega vrivanja, itn.

V nadaljevanju si bomo pogledali algoritem najbližjega vrivanja, ostale tipe tovrstnih algoritmov pa lahko bralec zasledi v literaturi [15].

8.3.1 Algoritem najbližjega vrivanja pri reševanju PTP problema

Algoritem se sestoji iz naslednjih korakov [15]:

KORAK 1: Konstruiraj podgraf, ki se sestoji samo iz vozlišča i.

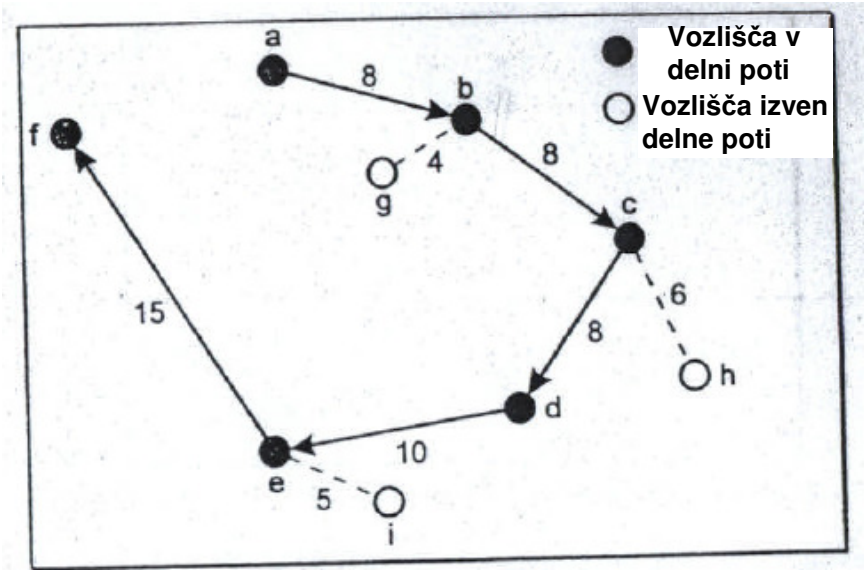
KORAK 2: Poišči tisto vozlišče k, za katerega je razdalja c_{ik} *najmanjša, kar pomeni poiskati tiso vozlišče k, ki je najbliže vozlišču i. Formiraj delno pot trgovskega potnika (TP), ki se glasi: (i, k, i).*

KORAK 3: Poišči tisto vozlišče k, ki ni vključeno v delno pot in je najbližje tej poti (vozlišče je najbližje delni poti, če je razdalja med njim in njemu najbližjem vozlišču (ki je že v delni poti) manjša od katerekoli razdalje med vozlišči izven poti in njim najbližjimi vozlišči znotraj poti).

KORAK 4: Poišči povezavo (i, j), ki je del delne poti in za katero velja najmanjša vrednost veličine $c_{ik} + c_{kj} - c_{ij}$. *Nato vrini vozlišče k med vozlišči i in j.*

KORAK 5: Zaključi z algoritmom, ko so vsa vozlišča vključena v pot trgovskega potnika. V primeru, če kakšno vozlišče še ni vključeno, se vrni na korak 3.

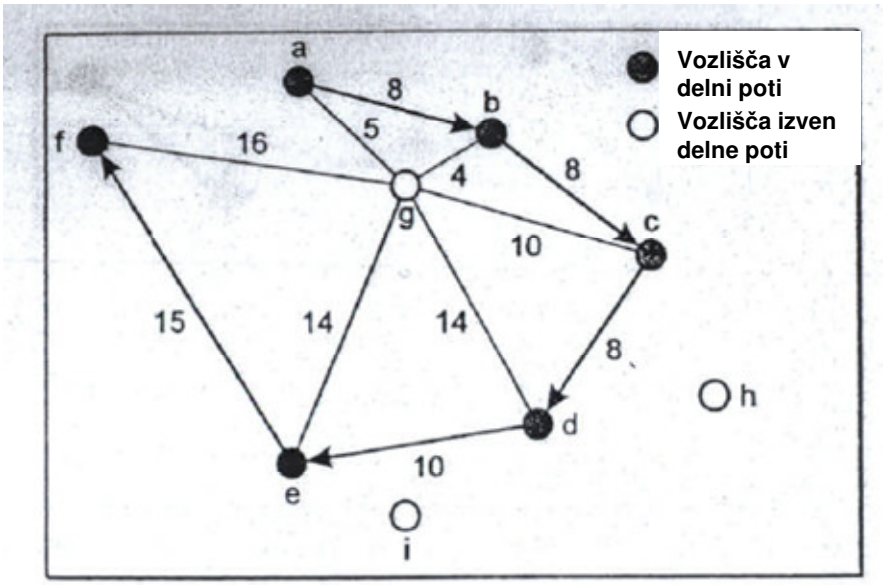
Morda koraka 3 in 4 nista popolnoma razumljiva, zato jih razložimo malce natančneje. Na sliki 135 je prikazana določena delna pot trgovskega potnika, ter nekaj vozlišč, ki še niso vključena v to pot.



Slika 135.: Izbor vozlišča, ki naj bi bilo naslednje vključeno v delno pot TP

Razlaga koraka 3 je naslednja. Delna pot TP se glasi: (a, b, c, d, e, f). Izven te poti se nahajajo vozlišča g, h in i. Iz slike 135 se vidi, da je vozlišče g gotovo najbližje delni poti. Razdalja $c_{bg} = 4$ med vozliščem g in njemu najbližjim vozliščem, ki je že vključeno v delno pot (vozlišče b), je gotovo manjša od razdalj med ostalimi vozlišči izven poti (h oz. i), ter njim najbližjim vozliščem znotraj poti (c oz. e). Ker velja: $c_{bg} = \min\{c_{bg}, c_{ch}, c_{ei}\} = min\{4,6,5\} = 4$, je torej vozlišče g tisto, ki ga je potrebno naslednjega vključiti v delno pot TP.

V koraku 4 pa je potrebno določiti pozicijo, kamor naj bi v delno pot vrinili vozlišče g, ki smo ga določili v koraku 3. Razdalje med vozliščem g in vozlišči, ki so že vključena v delno pot, je prikazana na sliki 136.



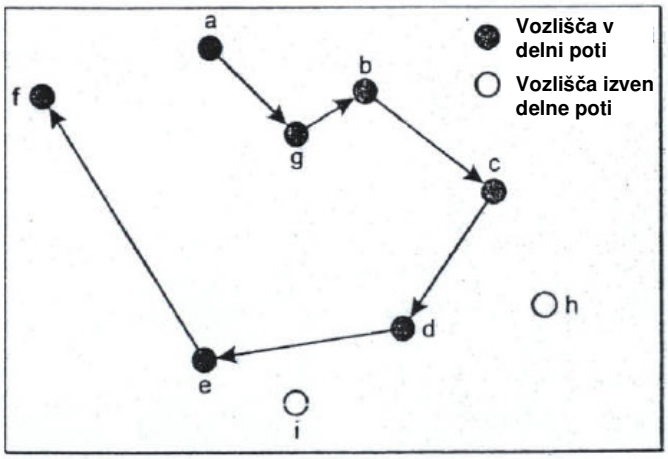
Slika 136.: Določanje pozicije v delni poti TP, kamor naj bi bilo vrinjeno vozlišče g.

V skladu s korakom 4 algoritma je potrebno tvoriti tudi izraze $c_{ik} + c_{kj} - c_{ij}$, ki so za primer iz slike 136 podani v tabeli 25.

$c_{ag} + c_{gb} - c_{ab}$	1
$c_{bg} + c_{gc} - c_{bc}$	6
$c_{cg} + c_{gd} - c_{cd}$	16
$c_{dg} + c_{ge} - c_{de}$	18
$c_{eg} + c_{gf} - c_{ef}$	15

Tabela 25.: Vrednosti veličin $c_{ik} + c_{kj} - c_{ij}$ za sliko 136

Iz tabele 25 je razvidno, da velja: $\min\{c_{ik} + c_{kj} - c_{ij}\} = c_{ag} + c_{gb} - c_{ab} = 1$, zato moramo v skladu s korakom 4 očitno vriniti vozlišče g v delno pot med vozliščema a in b. Na takšen način se bo tudi najmanj povečala dolžina delne poti TP, ko bomo vanjo vrinili novo vozlišče g. Nova delna pot TP po vrinjenju vozlišča g je prikazana na sliki 137.



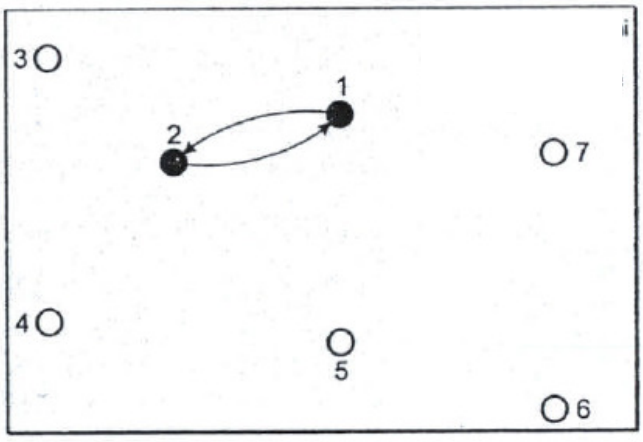
Slika 137.: Nova delna pot TP po vrinjenju vozlišča g.

Princip delovanja obravnavanega algoritma bomo dodatno osvetlili še na naslednjem primeru.

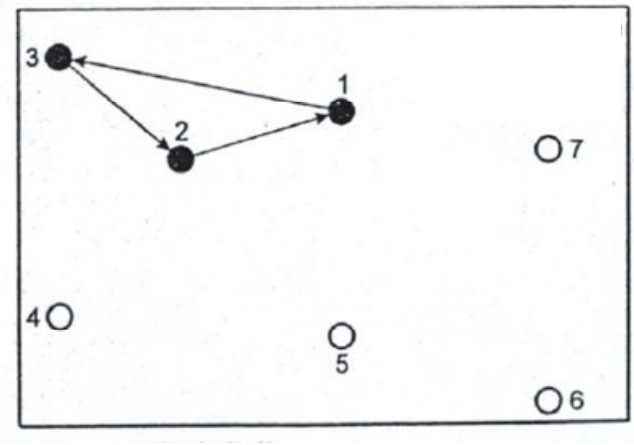
Primer:

Za graf, katerega vozlišča in matrika sosednosti so prikazana na sliki 129, poišči optimalen cikel trgovskega potnika s pomočjo algoritma najbližjega vrivanja. Kot izhodišče izberi vozlišče 1.

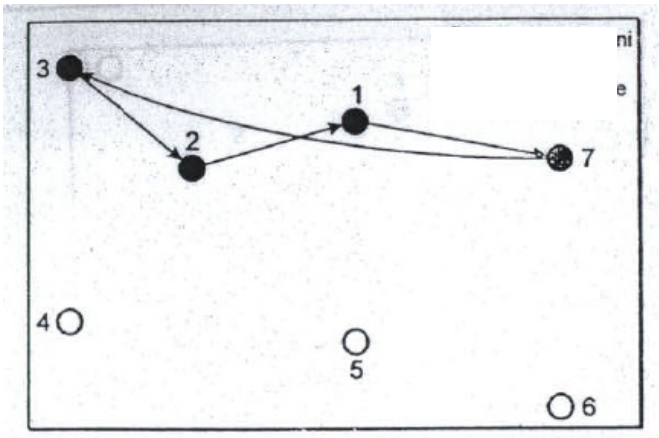
Postopek poteka na popolnoma enak način, kot je bil razložen v prejšnjem primeru. Na slikah 138 do 143 je prikazana izgradnja delne poti TP v posameznih fazah postopka.



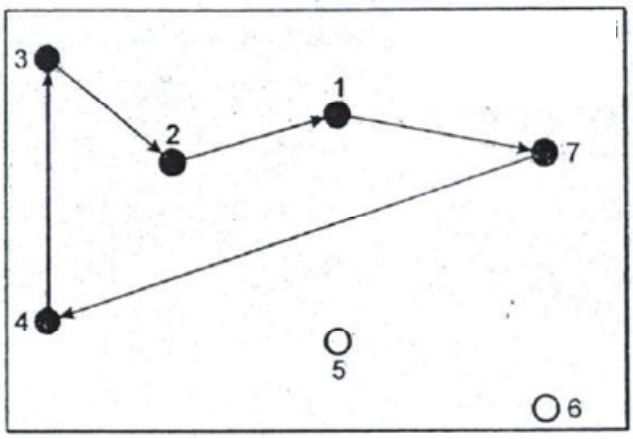
Slika 138.: Delna pot TP (1, 2, 1)



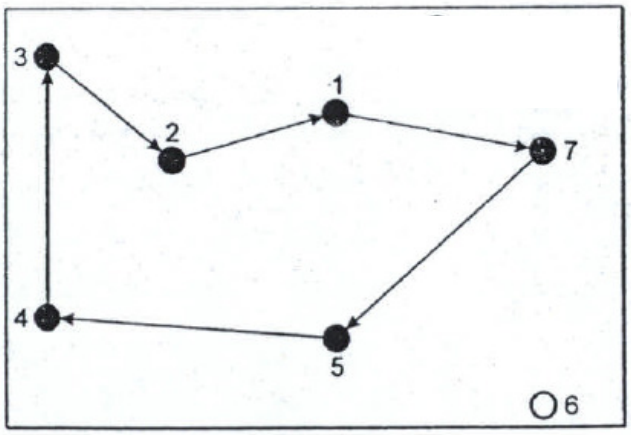
Slika 139.: Delna pot TP (1, 3, 2, 1)



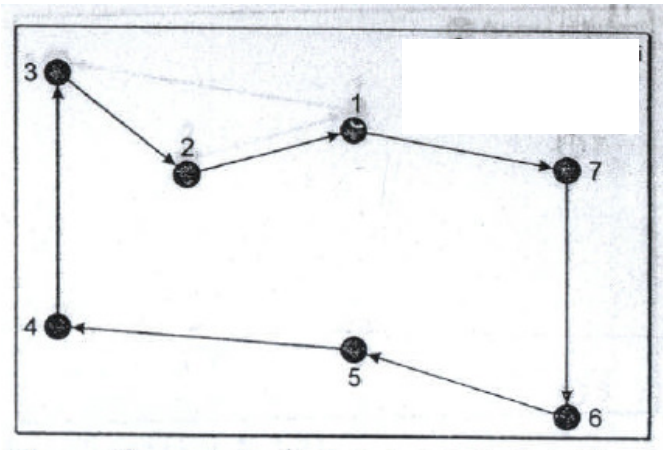
Slika 140.: Delna pot TP (1, 7, 3, 2, 1)



Slika 141.: Delna pot TP (1, 7, 4, 3, 2, 1)



Slika 142.: Delna pot TP (1, 7, 5, 4, 3, 2, 1)

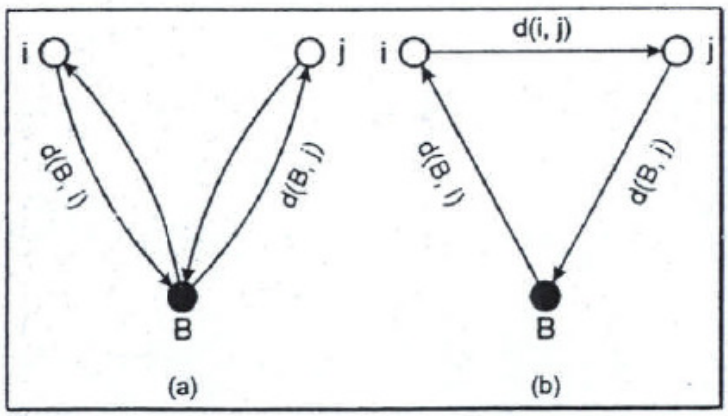


Slika 143.: Delna pot TP (1, 7, 6, 5, 4, 3, 2, 1)

8.4 Reševanje problemov PTP s pomočjo Clarke – Wrightovega algoritma prihrankov

Clarke – Wrightov algoritem prihrankov je vsekakor eden najbolj poznanih algoritmov za projektiranje poti prevoznih sredstev [15]. Se pa s pridom lahko uporabi tudi za reševanje problemov trgovskega potnika.

Denimo imamo dano vozlišče B, iz katerega izhaja in se vanj vrača trgovski potnik (glej sliko 144). Na sliki 144 a) je prikazana prva možna strategija premikanja potnika. To pomeni, da potnik krene iz vozlišča B, obišče vozlišče i, se vrne nazaj do B, obišče še vozlišče j, ter se vrne ponovno nazaj v vozlišče B.



Slika 144.: Izračunavanje prihrankov

V tem primeru je skupnja razdalja, ki jo je prepotoval, enaka:

$$2 \cdot d(B,i) + 2 \cdot d(B,j) \tag{8.16.}$$

Na sliki 144 b) je prikazana izboljšana strategija premikanja trgovskega potnika. V tem primeru je skupna razdalja, ki bi jo prepotoval, enaka:

$$d(B,i) + d(i,j) + d(B,j) \tag{8.17.}$$

Očitno prihranek s(i,j) izboljšane strategije premikanja potnika glede na slabšo dobimo tako, da od izraza (8.16.) odštejemo izraz (8.17.). Pri tem dobimo:

$$\begin{aligned} s(i,j) &= 2 \cdot d(B,i) + 2 \cdot d(B,j) - \left[d(B,i) + d(i,j) + d(B,j) \right] = \\ &= d(B,i) + d(B,j) - d(i,j) \end{aligned} \tag{8.18.}$$

Razumljivo je, da želimo, da je prihranek (8.18.) čim večji. To z drugimi besedami pomeni, večji ko je prihranek (8.18.), bolj si želimo, da vozlišči i in j spojimo v eno pot v okviru poti trgovskega potnika.

Clarke-Wrightov algoritem prihrankov za konstrukcijo poti trgovskega potnika je sestavljen iz naslednjih korakov [15]:

KORAK 1: Izračunaj prihranke $s(i,j) = d(B,i) + d(B,j) - d(i,j)$ *za vsak par vozlišč, ki jih je potrebno obiskati, kjer je B začetno vozlišče.*

KORAK 2: Izvrši rangiranje vseh prihrankov in jih sortiraj po velikosti. Pri tem napravi tabelo prihrankov, ki se začne z največjim prihrankom.

KORAK 3: Pri opazovanju prihranka s(i, j) vključi pripadajočo povezavo (i,j) v delno pot TP, če velja ena od naslednjih možnosti:

a) Ne vozlišče i, ne vozlišče j še nista nikjer vključeni v sekvenco delne poti TP,

b) Sicer je eno izmed vozlišč i ali j že vključeno v sekvenco delne poti TP, hkrati pa je to vozlišče tudi sosednje vozlišču B v sekvenci poti TP.

c) Obe vozlišči i in j sta že vključeni v sekvenco delne poti TP, hkrati pa sta ti vozlišči tudi sosednji vozlišču B v sekvenci poti TP.

KORAK 4: Ko je tabela prihrankov v celoti obdelana (oz. vsa vozlišča zajeta v delni poti TP), končaj postopek.

Princip delovanja algoritma bomo osvetlili na naslednjem primeru.

Primer:

Za graf, katerega vozlišča in matrika sosednosti so prikazana na sliki 129, poišči optimalen cikel trgovskega potnika s pomočjo Clarke-Wrightovega algoritma. Kot izhodišče izberi vozlišče 1.

V tabeli 26 so prikazane vrednosti prihrankov, izračunane v skladu z izrazom (8.18.) za vsak par vozlišč, ki so sortirane od največjega do najmanjšega prihranka.

Povezava (i,j)	Prihranek	Povezava (i,j)	Prihranek	Povezava (i,j)	Prihranek
(5,6)	225	(4,6)	135	(2,6)	45
(4,5)	165	(2,3)	120	(4,7)	45
(3,4)	150	(5,7)	120	(2,7)	15
(6,7)	150	(2,5)	75	(3,6)	15
(2,4)	135	(3,5)	60	(3,7)	15

Tabela 26.: Rangirana tabela prihrankov

Pri konstrukciji delne poti TP začnemo s povezavo (5,6), ki ima največji prihranek. Slednjega dobimo s pomočjo izraza (8.18.) (glej tudi sliko 129):

$$s(i,j) = d(B,i) + d(B,j) - d(i,j) \tag{8.19.}$$
$$s(5,6) = d(1,5) + d(1,6) - d(5,6) = 135 + 180 - 90 = 225$$

Po vključitvi vozlišč 5 in 6 v delno pot TP se njena sekvenca glasi: (1, 5, 6, 1). Naslednja povezava po velikosti prihranka je povezava (4,5). Tudi njen prihranek izračunamo s pomočjo izraza (8.18.) (glej tudi sliko 129):

$$s(i,j) = d(B,i) + d(B,j) - d(i,j) \tag{8.20.}$$
$$s(4,5) = d(1,4) + d(1,5) - d(4,5) = 165 + 135 - 135 = 165$$

Sicer je vozlišče 5 že vključeno v sekvenco delne poti TP, vendar je v njej sosednje vozlišču 1. Zato lahko povezavo (4,5) vključimo v delno pot TP, njena nova sekvenca pa se glasi: (1, 4, 5, 6, 1).

Naslednja povezava po velikosti prihranka je povezava (3,4). Tudi njen prihranek izračunamo s pomočjo izraza (8.18.) in slike 129. Sicer je vozlišče 4 že vključeno v sekvenco delne poti TP, vendar je v njej sosednje vozlišču 1. Zato lahko povezavo (3,4) vključimo v delno pot TP, njena nova sekvenca pa se glasi: (1, 3, 4, 5, 6, 1).

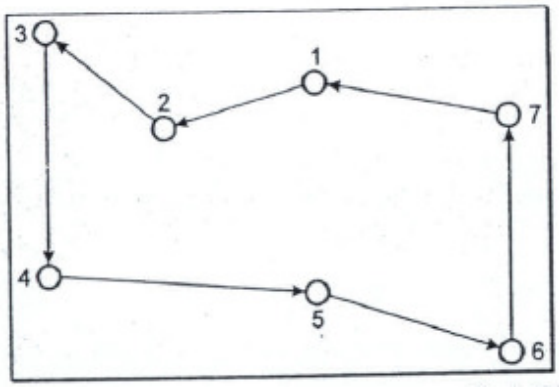
Naslednja povezava po velikosti prihranka je povezava (6,7). Tudi njen prihranek izračunamo s pomočjo izraza (8.18.) in slike 129. Sicer je vozlišče 6 že vključeno v sekvenco delne poti TP, vendar je v njej sosednje vozlišču 1. Zato lahko povezavo (6,7) vključimo v delno pot TP, njena nova sekvenca pa se glasi: (1, 3, 4, 5, 6, 7, 1).

Naslednja povezava po velikosti prihranka je povezava (2,4). Tudi njen prihranek izračunamo s pomočjo izraza (8.18.) in slike 129. Vozlišče 4 je že vključeno v sekvenco delne poti TP, hkrati pa v njej ni sosednje vozlišču 1 (ker je vmes vozlišče 3). Zato povezave (2,4) ne moremo vključimo v delno pot TP, njena sekvenca pa še vedno ostaja enaka.

Naslednja povezava po velikosti prihranka je povezava (4,6). Tudi njen prihranek izračunamo s pomočjo izraza (8.18.) in slike 129. Vozlišči 4 in 6 sta že vključeni v sekvenco delne poti TP, hkrati pa v njej nista sosednji vozlišču 1. Zato povezave (4,6) ne moremo vključimo v delno pot TP, kar je tudi logično, saj sta vozlišči 4 in 6 že vključeni v delno pot TP.

Naslednja povezava po velikosti prihranka je povezava (2,3). Tudi njen prihranek izračunamo s pomočjo izraza (8.18.) in slike 129. Sicer je vozlišče 3 že vključeno v sekvenco delne poti TP, vendar je v njej sosednje vozlišču 1. Zato lahko povezavo (2,3) vključimo v delno pot TP, njena nova sekvenca pa se glasi: (1, 2, 3, 4, 5, 6, 7, 1).

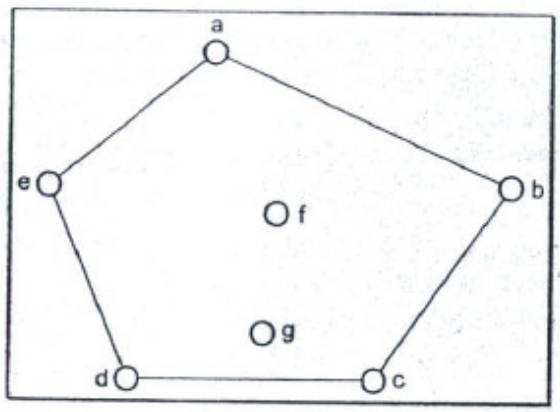
Glede na to, da smo že uspeli vključiti vsa vozlišča v pot trgovskega potnika, na tem mestu postopek končamo. Končni rezultat za pot TP se torej glasi: (1, 2, 3, 4, 5, 6, 7, 1) (glej sliko 145).



Slika 145.: Izračunana optimalna pot trgovskega potnika s pomočjo Clarke-Wrightovega algoritma prihrankov

8.5 Reševanje problemov PTP s pomočjo algoritma največjega kota

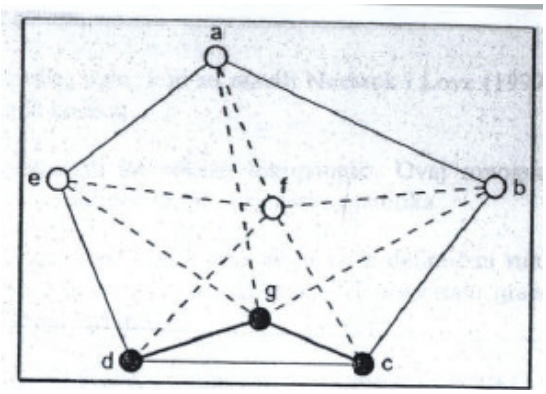
Algoritem sta razvila Norback in Love leta 1977 [15]. Osnovna ideja temelji na konstrukciji konveksnega mnogokotnika, katerega ogljišča so nekatera izmed vozlišč, ki morajo biti obiskana pri PTP problemu (glej sliko 146). Na takšen način dobimo delno pot TP, ki jo dokončamo tako, da vanjo vključimo še preostala neuporabljena vozlišča po principu tvorjenja največjega kota povezav med neuporabljenimi in že uporabljenimi vozlišči.



Slika 146.: Delna pot TP, dobljena na osnovi konstrukcije konveksnega mnogokotnika

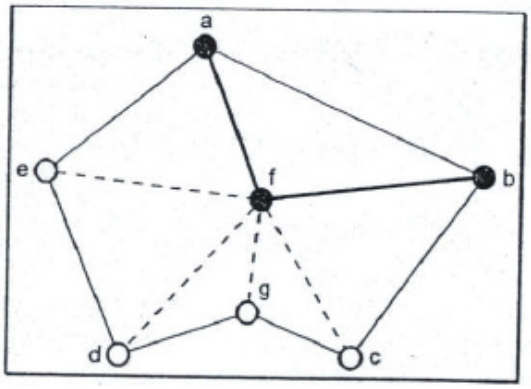
Na sliki 146 je prikazan konveksni mnogokotnik, ki ga tvorijo vozlišča a, b, c, d in e. Delna pot TP se torej glasi: (a, b, c, d, e, a). Kot je razvidno iz slike 146, v pot TP še nista vključeni vozlišči f in g. Slednji je tudi potrebno nekako spojiti z vozlišči, ki so že vključena v delno pot TP. To storimo tako, da vsako izmed teh ogljišč (f, g) povežemo z

vsemi preostalimi vozlišči mnogokotnika, ter opazujemo, katera povezava tvori največji kot (ki ga tvorita povezavi od sosednjih vozlišč mnogokotnika do neuporabljenega vozlišča!). Ko to ugotovimo, dotično vozlišče vključimo v delno pot TP. Ko na takšen način vključimo v delno pot TP obe vozlišči f in g, je pot TP v celoti zgrajena. Sliki 147 in 148 prikazujeta idejo, kako vključiti še preostali vozlišči f in g v delno pot TP po principu največjega kota.



Slika 147.: Določanje največjega kota za vozlišče g, ter njegova vključitev v delno pot TP

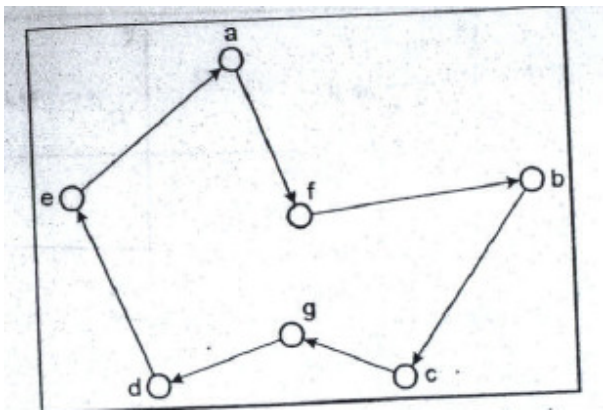
Iz slike 147 je razvidno, da je največji kot, ki ga tvorita povezavi od sosednjih vozlišč mnogokotnika do vozlišča g, kot $<$dgc (ki je hkrati tudi največji od vseh kotov do obeh neuporabljenih vozlišč). Zato vozlišče g vključimo v delno pot TP med vozlišči d in c, pri čemer dobimo nov mnogokotnik z vozlišči a, b, c, g, d in e (glej sliko 148).



Slika 148.: Nova delna pot TP, ko smo vanjo vključili tudi vozlišče g

Vozlišče f je zadnje vozlišče, ki ga je še potrebno vključiti v pot TP. Zopet pogledamo, kateri je največji kot, ki ga tvorita povezavi od sosednjih vozlišč mnogokotnika do vozlišča f (glej sliko 148). Očitno je to kot $<$afb. Zato vozlišče f vključimo v delno pot

TP med vozlišči a in b, pri čemer dobimo nov mnogokotnik z vozlišči a, f, b, c, g, d in e, ki hkrati predstavlja tudi do konca zgrajeno pot TP, ki se torej glasi: (a, f, b, c, g, d, e, a) (glej sliko 149).



Slika 149.: Optimalna pot TP, dobljena z algoritmom največjega kota

Algoritem za konstrukcijo konveksnega mnogokotnika se glasi [15]:

KORAK 1: Vozlišča, ki jih je potrebno obiskati, opazuj v kartezičnem koordinatnem sistemu. Izberi vozlišče, kateremu pripada najmanjša vrednost x koordinate. Označi ga s h_1 *in ga vključi v konveksni mnogokotnik.*

KORAK 2: Naredi povezave od vozlišča h_1 *do vseh ostalih vozlišč. Vozlišče, ki tvori največji kot, označi s* h_2 *in ga vključi v mnogokotnik.*

KORAK 3: Poveži vozlišči h_1 *in* h_2 *v daljico, ki predstavlja prvi krak. Poveži vozlišče* h_2 *s preostalimi vozlišči, kar bo dalo množico drugih krakov in s tem množico kotov. Izmed slednjih poišči največji kot med krakoma, ter pripadajočemu vozlišču daj oznako* h_3 *in ga vključi v mnogokotnik.*

KORAK 4: Ponavljaj korak 3, kar pomeni: Poveži vozlišči h_{i-1} *in* h_i *v daljico, ki predstavlja prvi krak. Poveži vozlišče* h_i *s preostalimi vozlišči, kar bo dalo množico drugih krakov in s tem množico kotov. Izmed slednjih poišči največji kot med krakoma, ter pripadajočemu vozlišču daj oznako* h_{i+1} *in ga vključi v mnogokotnik.*

KORAK 5: Končaj postopek, ko je mnogokotnik v celoti zgrajen in se vrnemo na vozlišče h_1.

Primer uporabe algoritma za konstrukcijo konveksnega mnogokotnika je prikazan na sliki 150 (zaporedje slik od 1) do 5)).

Algoritem vključitve še preostalih neuporabljenih vozlišč po principu tvorjenja največjega kota povezav med neuporabljenimi in že uporabljenimi vozlišči pa se glasi [15]:

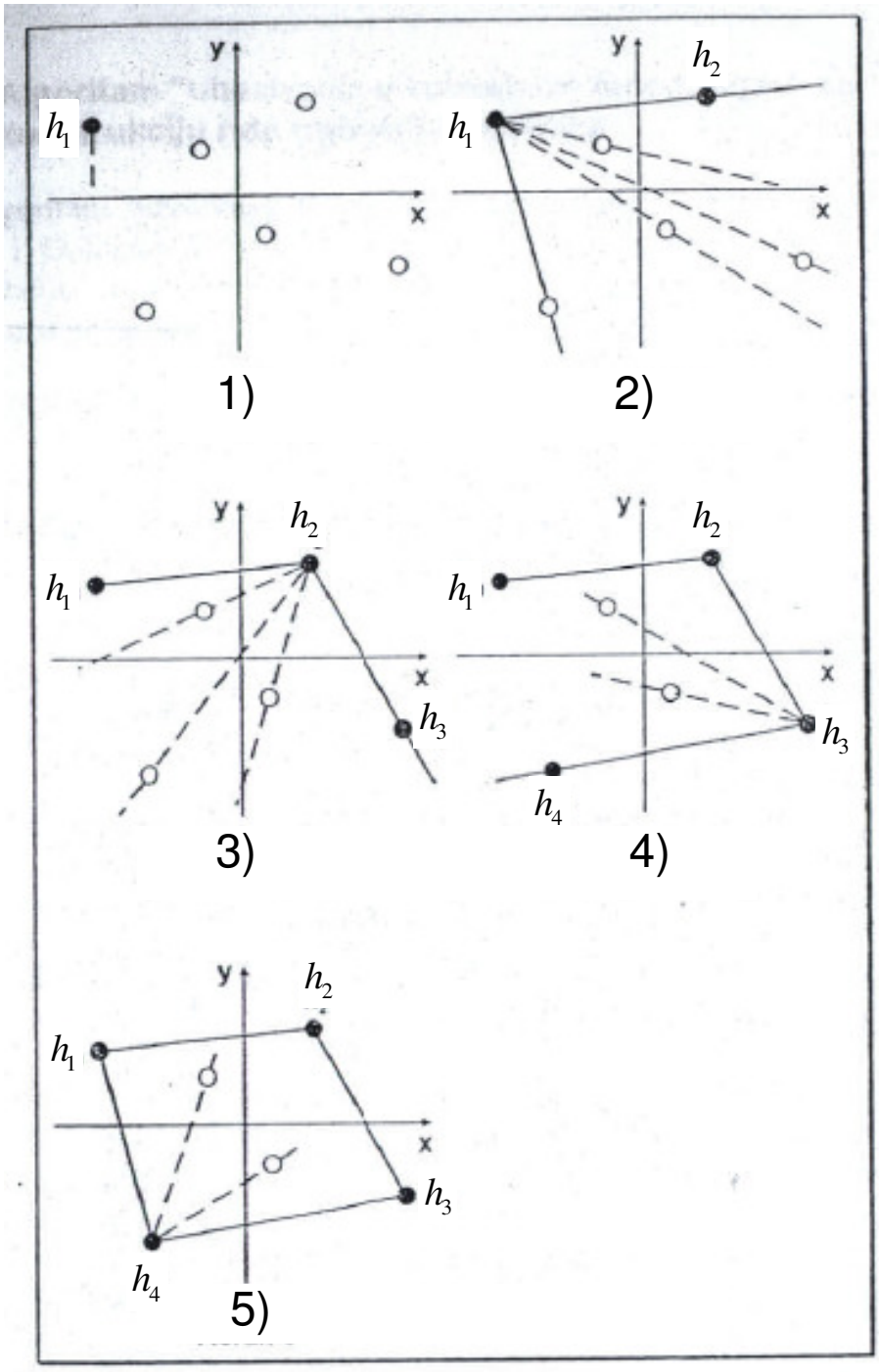
KORAK 1: Konstruiraj konveksni mnogokotnik, kot je opisano v prejšnjem algoritmu. Tako dobimo začetno delno pot TP.

KORAK 2: Določi vozlišče k, ki še ni vključeno v delno pot TP, ter povezavo (i,j), ki je že vključena v delno pot TP, na takšen način, da je kot, ki ga tvorita povezavi (i,k) in (k,j), največji možen kot.

KORAK 3: Vključi vozlišče k v delno pot med vozlišči i in j.

KORAK 4: Ponavljaj koraka 2 in 3 toliko časa, dokler pot TP ni v celoti zgrajena.

Algoritem največjega kota je računsko izredno učinkovit, kar še posebej velja za primere manjših dimenzij (20-30 vozlišč), ki se lahko rešijo zelo hitro celo brez uporabe računalnika [15].



Slika 150.: Primer konstrukcije konveksnega mnogokotnika

Princip delovanja tega algoritma bomo še dodatno osvetlili na naslednjem primeru.

Primer:

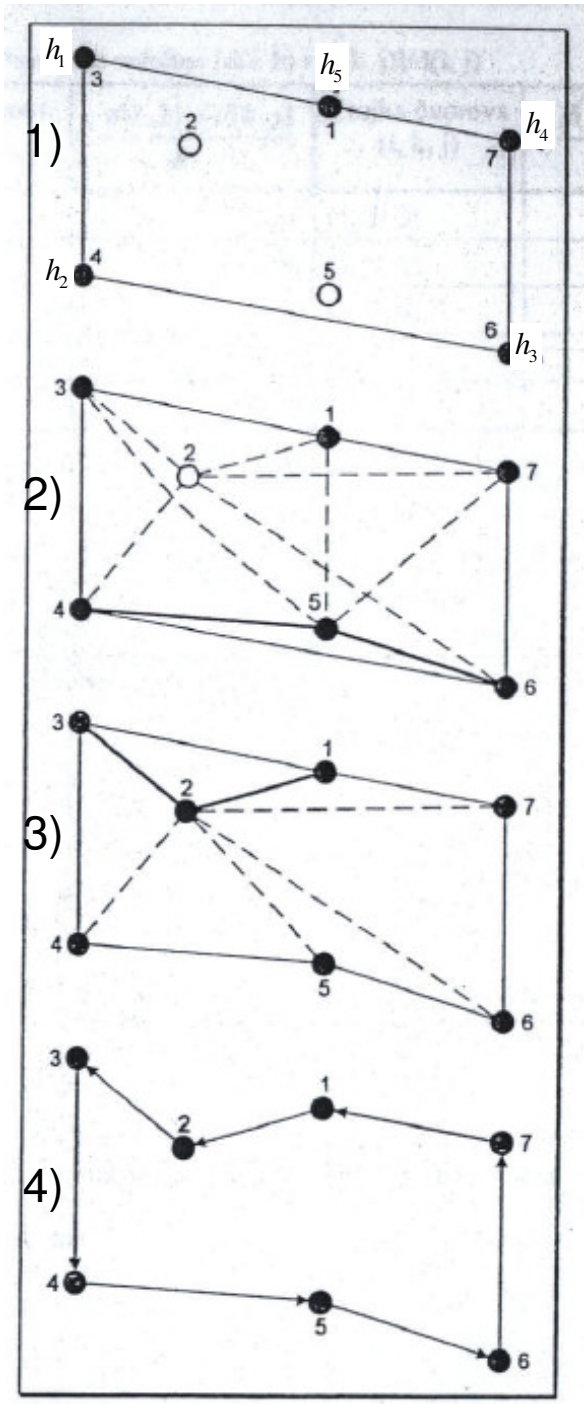
Za graf, katerega vozlišča in matrika sosednosti so prikazana na sliki 129, poišči optimalen cikel trgovskega potnika s pomočjo algoritma največjega kota. Kot izhodišče izberi vozlišče 1.

Rešitev PTP za dani primer s pomočjo algoritma največjega kota je prikazana na sliki 151. Kot je razvidno iz slike 151, najprej tvorimo konveksni mnogokotnik (1. slika), katerega ogljišča so vozlišča 3, 1, 7, 6 in 4 (dodali smo jim tudi oznake h_i). Ta mnogokotnik tvori začetno delno pot TP, vozlišči 2 in 5 pa sta še neuporabljeni.

Slednji je tudi potrebno nekako spojiti z vozlišči, ki so že vključena v delno pot TP. To storimo tako, da vsako izmed teh ogljišč (2, 5) povežemo z vsemi preostalimi vozlišči mnogokotnika (2. slika na sliki 151), ter opazujemo, katera povezava tvori največji kot (ki ga tvorita povezavi od sosednjih vozlišč mnogokotnika do neuporabljenega vozlišča!). Ko to ugotovimo, dotično vozlišče vključimo v delno pot TP. Ko na takšen način vključimo v delno pot TP obe vozlišči 2 in 5, je pot TP v celoti zgrajena.

Iz slike 151, 2. slika, je razvidno, da je največji kot, ki ga tvorita povezavi od sosednjih vozlišč mnogokotnika do vozlišča 5, kot <456 (ki je hkrati tudi največji od vseh kotov do obeh neuporabljenih vozlišč). Zato vozlišče 5 vključimo v delno pot TP med vozlišči 4 in 6, pri čemer dobimo nov mnogokotnik z vozlišči 3, 1, 7, 6, 5, 4 (3. slika na sliki 151).

Vozlišče 2 je zadnje vozlišče, ki ga je še potrebno vključiti v pot TP. Zopet pogledamo, kateri je največji kot, ki ga tvorita povezavi od sosednjih vozlišč mnogokotnika do vozlišča 2 (glej sliko 151, 3. slika). Očitno je to kot <321. Zato vozlišče 2 vključimo v delno pot TP med vozlišči 3 in 1, pri čemer dobimo nov mnogokotnik z vozlišči 3, 2, 1, 7, 6, 5 in 4, ki hkrati predstavlja tudi do konca zgrajeno pot TP, ki se glasi: (1, 2, 3, 4, 5, 6, 7, 1) (glej sliko 151, 4. slika).



Slika 151.: Konstrukcija poti TP s pomočjo algoritma največjega kota. Optimalna pot se glasi: (1, 2, 3, 4, 5, 6, 7, 1)

8.6 Reševanje problemov PTP s pomočjo algoritma vrivanja v konveksni mnogokotnik

Ta algoritem sta razvila Stewart in Golden leta 1981 in je dokaj podoben algoritmu največjega kota za konstrukcijo poti TP v poglavju 8.5. Algoritem sestavljajo naslednji koraki [15]:

KORAK 1: Konstruiraj konveksni mnogokotnik na način, kot je bilo to storjeno pri algoritmu največjega kota. Ta mnogokotnik predstavlja začetno delno pot TP.

KORAK 2: Določi vozlišče k, ki še ni vključeno v delno pot TP, pri čemer naj zanj in za neko povezavo (i,j), že vključeno v delno pot, velja, da je veličina:

$$\frac{d(i,k)+d(k,j)}{d(i,j)} \qquad (8.21.)$$

najmanjša možna.

KORAK 3: Vključi (vrini) vozlišče k med vozlišči i in j.

KORAK 4: Ponavljaj koraka 2 in 3 toliko časa, dokler pot TP ni v celoti zgrajena.

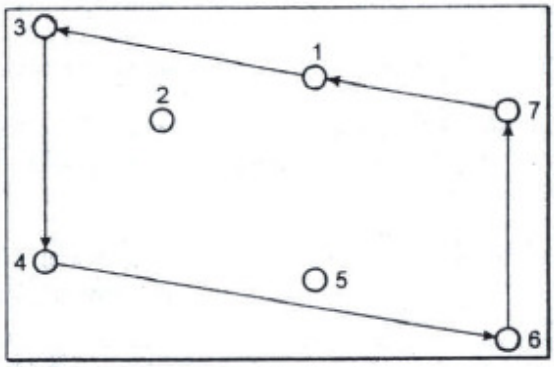
Namen koraka 2 je v prizadevanju, da se z vrinjenjem vozlišča k v delno pot TP med vozlišči i in j minimalno mogoče poveča dolžina delne poti TP.

Delovanje algoritam bomo osvetlili na naslednjem primeru.

Primer:

Za graf, katerega vozlišča in matrika sosednosti so prikazana na sliki 129, poišči optimalen cikel trgovskega potnika s pomočjo algoritma vrivanja v konveksni mnogokotnik. Kot izhodišče izberi vozlišče 1.

Kot je razvidno iz slike 152, najprej tvorimo konveksni mnogokotnik (glej poglavje 8.5.), katerega ogljišča so vozlišča 1, 3, 4, 6, 7. Ta mnogokotnik tvori začetno delno pot TP, vozlišči 2 in 5 pa sta še neuporabljeni. Da bi ju znali vključiti v delno pot TP, moramo uporabiti izračune na osnovi izraza (8.21.).



Slika 152: Konveksni mnogokotnik, ki predstavlja začetno delno pot TP (1, 3 , 4, 6, 7, 1).

Če izračunamo rezultate na osnovi izraza (8.21.) za vse potrebne trojice vozlišč, dobimo tabelo 27.

Trojica vozlišč (i, k, j)	$\frac{d(i,k)+d(k,j)}{d(i,j)}$	Trojica vozlišč (i, k, j)	$\frac{d(i,k)+d(k,j)}{d(i,j)}$
(1, 2, 3)	1.22	(1, 5, 3)	2.55
(3, 2, 4)	1.30	(3, 5, 4)	2.30
(4, 2, 6)	1.50	(4, 5, 6)	1.07
(6, 2, 7)	3.00	(6, 5, 7)	1.62
(7, 2, 1)	2.50	(7, 5, 1)	2.66

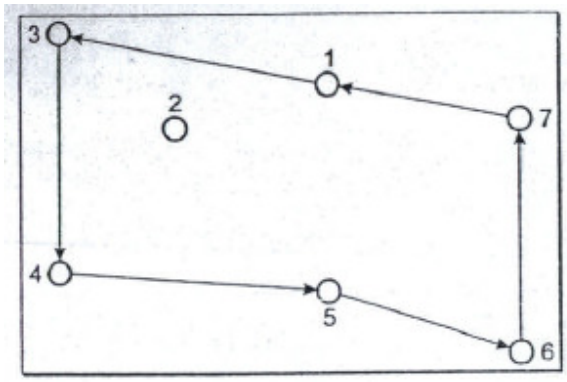
Tabela 27.: Izračuni izraza (8.21.) za neuporabljeni vozlišči 2 in 5

Kot je razvidno iz tabele 27, se v 1. oz. 3. stolpu generirajo trojice za vozlišči 2 oz. 5, ki sta vedno v sredini trojice, levo in desno v njej pa sta sosednji dve vozlišči, ki sta že uporabljeni v delni poti TP.

Kar se tiče izračunov na osnovi izraza (8.21.), moramo seveda pri tem opazovati tudi matriko sosednosti na sliki 129, tako npr. za trojico vozlišč (1, 2, 3) dobimo rezultat:

$$\frac{d(i,k)+d(k,j)}{d(i,j)}=\frac{d(1,2)+d(2,3)}{d(1,3)}=\frac{75+90}{135}=1.2222 \qquad (8.22.)$$

kar se ujema z vrednostjo v 1. vrstici in 2. stolpu tabele 27. Na podoben način izračunamo tudi vse ostale vrednosti v 2. oz. 4. stolpu tabele 27. V 2. in 4. stolpu te tabele nato opazujemo, kje se nahaja najmanjša vrednost. Pri vozlišču 2 je to vrednost 1.22 pri trojici vozlišč (1, 2, 3), pri vozlišču 5 pa vrednost 1.07 pri trojici vozlišč (4, 5, 6). Torej bomo očitno morali vozlišče 2 vriniti med vozlišči 1 in 3, vozlišče 5 pa med vozlišči 4 in 6. Če najprej vrinemo vozlišče 5 med vozlišči 4 in 6, dobimo delno pot TP, prikazano na sliki 153.



Slika 153.: Delna pot TP (1, 3 , 4, 5, 6, 7, 1).

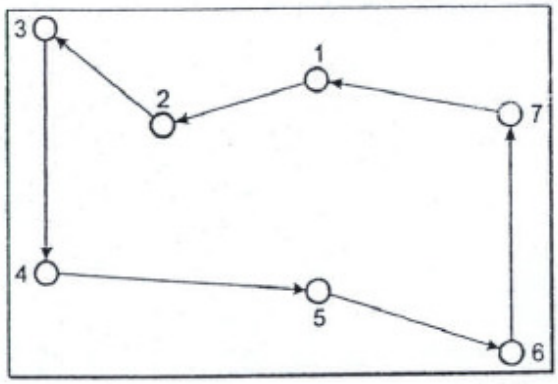
Da lahko vrinemo še vozlišče 2 med vozlišči 1 in 3, se lahko dodatno prepričamo tudi na osnovi tabele 28, ki jo tvorimo za trojice vozlišč pri spremenjeni delni poti TP na sliki 153.

Trojica vozlišč (i, k, j)	$\frac{d(i,k)+d(k,j)}{d(i,j)}$	Trojica vozlišč (i, k, j)	$\frac{d(i,k)+d(k,j)}{d(i,j)}$
(1, 2, 3)	1.22	(5, 2, 6)	3.83
(3, 2, 4)	1.30	(6, 2, 7)	3.00
(4, 2, 5)	1.77	(7, 2, 1)	2.50

Tabela 28.: Izračuni izraza (8.21.) za neuporabljeno vozlišče 2

Očitno je tudi tokrat najmanjša vrednost 1.22 izraza (8.21.) dobljena pri trojici vozlišč (1, 2, 3). Torej smo se dokončno prepričali, da lahko vozlišče 2 vrinemo med vozlišči 1 in 3.

Končna pot TP (1, 2, 3, 4, 5, 6, 7, 1) je po vključitvi še tega vozlišča prikazana na sliki 154.



Slika 154.: Končna pot TP (1, 2, 3, 4, 5, 6, 7, 1) na osnovi algoritma vrivanja v konveksni mnogokotnik.

8.7 Reševanje problemov PTP s pomočjo Orovega algoritma

Orov algoritem je dokaj podoben prej razloženima algoritmoma v poglavjih (8.5.) in (8.6.). Tudi tukaj je najprej potrebno konstruirati konveksni mnogokotnik, ki nam da začetno delno pot trgovskega potnika. Le da je v tem primeru nato potrebno določiti vozlišča k, ki še niso vključena v delno pot TP, na takšen način, da naj zanje in za povezave (i,j), že vključene v delno pot, velja, da je veličina:

$$\left[d(i,k)+d(k,j)-d(i,j)\right]\cdot\frac{d(i,k)+d(k,j)}{d(i,j)} \qquad (8.23.)$$

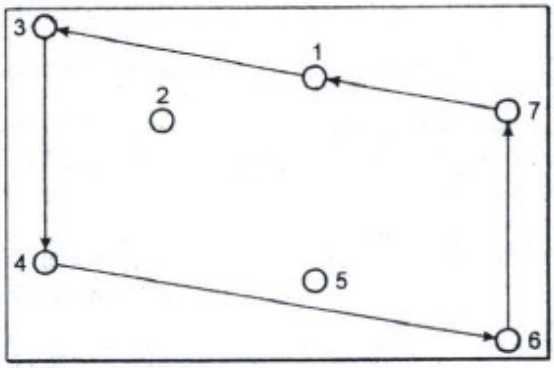
najmanjša možna [15].

Primer:

Za graf, katerega vozlišča in matrika sosednosti so prikazana na sliki 129, poišči optimalen cikel trgovskega potnika s pomočjo Orovega algoritma. Kot izhodišče izberi vozlišče 1.

Kot je razvidno iz slike 155, najprej tvorimo konveksni mnogokotnik (glej poglavje 8.5.), katerega ogljišča so vozlišča 1, 3, 4, 6, 7. Ta mnogokotnik tvori začetno delno pot TP,

vozlišči 2 in 5 pa sta še neuporabljeni. Da bi ju znali vključiti v delno pot TP, moramo uporabiti izračune na osnovi izraza (8.23.).



Slika 155: Konveksni mnogokotnik, ki predstavlja začetno delno pot TP (1, 3 , 4, 6, 7, 1).

Če izračunamo rezultate na osnovi izraza (8.23.) za vse potrebne trojice vozlišč, dobimo tabelo 29.

Trojica vozlišč (i, k, j)	$\left[d(i,k)+d(k,j)-d(i,j)\right]\cdot\left[\frac{d(i,k)+d(k,j)}{d(i,j)}\right]$
(1, 2, 3)	36.60
(3, 2, 4)	58.50
(4, 2, 6)	157.50
(6, 2, 7)	720.00
(7, 2, 1)	337.50
(1, 5, 3)	535.50
(3, 5, 4)	448.50
(4, 5, 6)	16.05
(6, 5, 7)	121.50
(7, 5, 1)	399.00

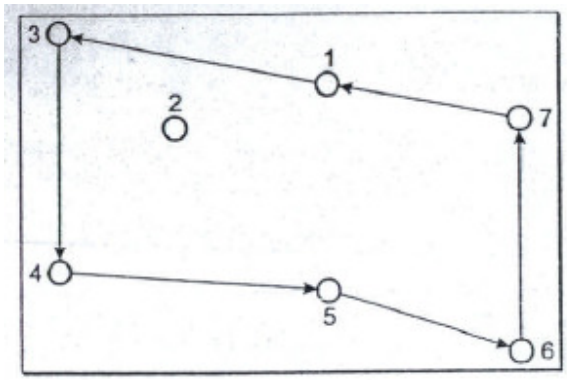
Tabela 29.: Izračuni izraza (8.23.) za neuporabljeni vozlišči 2 in 5

Kot je razvidno iz tabele 29, se v 1. stolpu generirajo trojice za vozlišči 2 oz. 5, ki sta vedno v sredini trojice, levo in desno v njej pa sta sosednji dve vozlišči, ki sta že uporabljeni v delni poti TP.

Kar se tiče izračunov na osnovi izraza (8.23.), moramo seveda pri tem opazovati tudi matriko sosednosti na sliki 129, tako npr. za trojico vozlišč (1, 2, 3) dobimo rezultat:

$$\begin{aligned}&\left[d(i,k)+d(k,j)-d(i,j)\right]\cdot\frac{d(i,k)+d(k,j)}{d(i,j)}=\\&=\left[d(1,2)+d(2,3)-d(1,3)\right]\cdot\frac{d(1,2)+d(2,3)}{d(1,3)}=\\&=\left[75+90-135\right]\cdot\frac{75+90}{135}=36,6\end{aligned} \qquad (8.24.)$$

kar se ujema z vrednostjo v 1. vrstici in 2. stolpu tabele 29. Na podoben način izračunamo tudi vse ostale vrednosti v 2. stolpu tabele 29. V 2. stolpu te tabele nato opazujemo, kje se nahaja najmanjša vrednost. Pri vozlišču 2 je to vrednost 36.6 pri trojici vozlišč (1, 2, 3), pri vozlišču 5 pa vrednost 16.05 pri trojici vozlišč (4, 5, 6). Torej bomo očitno vozlišče 2 morali vriniti med vozlišči 1 in 3, vozlišče 5 pa med vozlišči 4 in 6. Če najprej vrinemo vozlišče 5 med vozlišči 4 in 6, dobimo delno pot TP, prikazano na sliki 156.



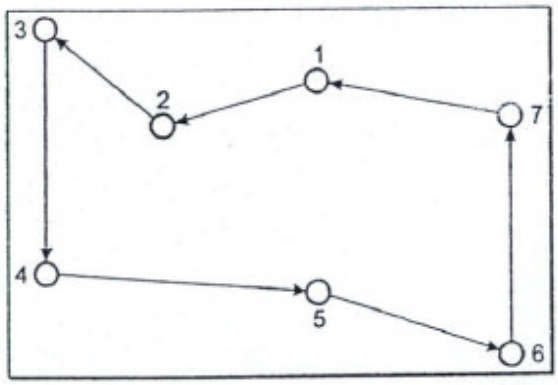
Slika 156.: Delna pot TP (1, 3, 4, 5, 6, 7, 1).

Da lahko vrinemo še vozlišče 2 med vozlišči 1 in 3, se lahko dodatno prepričamo tudi na osnovi tabele 30, ki jo tvorimo za trojice vozlišč pri spremenjeni delni poti TP na sliki 156.

Trojica vozlišč (i, k, j)	$\left[d(i,k)+d(k,j)-d(i,j)\right]\cdot\left[\frac{d(i,k)+d(k,j)}{d(i,j)}\right]$
(1, 2, 3)	36.60
(3, 2, 4)	58.50
(4, 2, 5)	185.85
(5, 2, 6)	976.65
(6, 2, 7)	720.00
(7, 2, 1)	337.50

Tabela 30.: Izračuni izraza (8.23.) za neuporabljeno vozlišče 2

Očitno je tudi tokrat najmanjša vrednost 36.6 izraza (8.23.) dobljena pri trojici vozlišč (1, 2, 3). Torej smo se dokončno prepričali, da lahko vozlišče 2 vrinemo med vozlišči 1 in 3. Končna pot TP (1, 2, 3, 4, 5, 6, 7, 1) je po vključitvi še tega vozlišča prikazana na sliki 157.



Slika 157.: Končna pot TP (1, 2, 3, 4, 5, 6, 7, 1) na osnovi Orovega algoritma.

8.8 Reševanje problemov PTP s Christofidesovim algoritmom

Algoritem je bi razvit leta 1976 in vsebuje naslednje ključne elemente [15]:

- Iskanje minimalno vpetega drevesa A, ki povezuje n vozlišč.
- Denimo je k on n vozlišč takšnih, ki imajo liho stopnjo vozlišč. Potrebno je izvesti spajanje po dve in dve vozlišči izmed teh k vozlišč na takšen način, da je skupna

dolžina povezav, ki spajajo ta vozlišča, najmanjša možna. Teh k vozlišč s pripadajočimi povezavami definira mrežo, ki jo označimo z B.

- Potrebno je načrtati mrežo C, ki predstavlja unijo mreže B in mreže A. Ta mreža ne ne vsebuje vozlišč lihe stopnje.
- Nato je potrebno poiskati Eulerjev obhod v mreži C, ki predstavlja približno rešitev za problem trgovskega potnika.
- Nazadnje je potrebno izboljšati dobljeno pot trgovskega potnika.

Podrobnosti glede tega algoritma lahko bralec zasledi v literaturi [15].

8.9 Problemi multiplega trgovskega potnika (PMTP) in problemi usmerjanja vozil (PUV)

Multipli trgovski potnik predstavlja generalizacijo klasičnega problema trgovskega potnika, pri čemer je v optimalni rešitvi upoštevanih več trgovskih potnikov [15]. Problemi multiplih trgovskih potnikov (PMTP) se pojavljajo v številnih realnih aplikacijah, da pa se jih tudi razširiti v probleme usmerjanja vozil (Vehicle Routing problem - VRP) z upoštevanjem določenih dodatnih omejitev.

Pri problemih multiplega trgovskega potnika je potrebno določiti set optimalnih poti za m trgovskih potnikov, ki vsi začnejo in končajo svoj obhod v isti izhodiščni lokaciji (single depot). Pri tem mora biti vsako vmesno vozlišče obiskano natanko enkrat, ter morajo biti skupni stroški obiskovanja vseh vozlišč najmanjši možni. Obstajajo tudi modificirane različice klasičnega PMTP problema, če imamo npr. opravka z več izhodiščnimi lokacijami (multiple depots), ali imamo opravka s časovnimi okni (vozlišča morajo biti obiskana v predpisanem časovnem intervalu), in podobno [15].

Nekateri najbolj značilni problemi multiplega trgovskega potnika so naslednji [15]:

- Razporejanje in razvrščanje posadk,
- Razporejanje in razvrščanje vozil pri obisku določenih lokacij,

- Problem optimalnega razvrščanja in usmerjanja šolskih avtobusov,
- Optimalno razporejanje delovne sile, transportnih sredstev in proizvodnje,
- Problemi pobiranja in dostavljanja (Pickup and delivery problems),
- Itn.

Poleg eksaktnih algoritmov za reševanje obstaja cela množica hevrističnih algoritmov in metod za reševanje PMTP problemov, kot npr.: Paralelno procesiranje na osnovi evolucijskega programiranja, Pristopi z nevronskimi mrežami, Pristop z genetskimi algoritmi, pristop s Tabu Search tehnikami, itn [15].

V praksi se pogosto srečujemo tudi s problemi usmerjanja vozil. To pomeni projektiranje optimalnih poti, po katerih naj se premikajo prevozna sredstva. Slednja morajo velikokrat obiskati točno določeno število vozlišč v transportnem grafu, ali pa prevoziti točno določene povezave v tem grafu. Značilni primeri PUV problemov so [15]:

- Zbiranje smeti,
- Zbiranje ali razvažanje pošte,
- Čiščenje in pranje ulic,
- Razvoz časopisov, mleka, kruha, itn,
- Zbiranje učencev za transport v njihove šole,
- Razporejanje avionov ali avtobusov,
- Razporejanje posadk avionov ali voznikov avtobusov na določena opravila, itn.

Za reševanje različnih variant problemov rotacije prevoznih sredstev ali problemov optimalnega planiranja razporejanja posadk se uporabljajo različne tehnike, kot npr. dinamično porgramiranje ali metoda razvejitev in omejitev [15]. Torej je veliko število tovrstnih problemov reševano na osnovi najrazličnejših hevrističnih algoritmov. V mnogo primerih namreč klasične tehnike matematičnega programiranja potrošijo preveč časa za računanje optimalnih rešitev, še posebej, ko imamo opravka z velikim številom vozlišč na trasportni mreži. Torej je edini način za reševanje kompleksnejših kombinatornih problemov na transportnih grafih takšen, da se uporabijo različne hevristične metode. Pri

tem pa velja, da uporaba najprimernejše hevristične procedure, dinamičnega programiranja, kombinatornega programiranja, ali neke druge skupine metod, zavisi predvsem od vrste problema, ki ga rešujemo [15].

Problemi usmerjanja vozil, problemi določanja optimalnega položaja baz prevoznih sredstev na transportni mreži, problemi planiranja posadk, itn, spadajo v razred kombinatornih problemov [15]. Te lahko delimo v probleme nizanja, probleme razporejanja ali razvrščanja, probleme izbora, ali katerokoli kombinacijo teh problemov.

V vseh transportnih poteh se v odvisnosti od konkretnega problema pojavljajo različne možne kombinacije problemov usmerjanja vozil in njihovih voznih redov na transportni mreži. Pri tem velja, da dobro načrtovano usmerjanje vozil in vozni red voženj lahko ključno doprineseta tako k zmanjšanju stroškov transporta, kot tudi dvigu kvalitete transportne storitve.

Več podrobnosti o reševanju problemov multiplega trgovskega potnika in usmerjanja vozil, ter vsemu instrumentariju v ta namen uporabljene metodologije, lahko bralec zasledi v literaturi [15].

9 LOKACIJSKI PROBLEMI

Od položaja določenih objektov na transportni mreži lahko ključno zavisi tako kvaliteta prometnih storitev, kot tudi skupni stroški transportnega sistema. Položaj objekta na mreži, v katerem se vrši določena storitev, zavisi tudi od same vrste storitve [15]. Tako moramo na primer neko letališče zaradi ekoloških razlogov locirati kar najdlje od centra mesta. Vendar pa po drugi strani letališče ne smemo locirati predaleč od centra mesta, saj s tem opazno znižamo kvaliteto storitev v letalskem prometu. Lociranje postaj javnega mestnega prometa pa je poleg ostalih kriterijev pogojeno tudi z zahtevo, da se kar najbolj minimizira dolžina pešačenja uporabnikov javnega prevoza. Pri lociranju gasilskih in policijskih postaj pa moramo upoštevati zahtevo, da je razdalja do najbolj oddaljenega uporabnika tovrstnih storitev kar najmanjša možna [5, 15].

Teorija lokacijskih problemov poskuša odgovoriti na naslednja ključna vprašanja [15]:

- Kolikšno mora biti skupno število objektov na mreži, v katerih se bo opravljala storitev (kandidati za ponudnike storitev),
- Kam locirati te objekte,
- Na kakšen način izvršiti razporeditev klientov (uporabnikov oz. potrošnikov storitev), ki zahtevajo storitev po posameznih objektih, oz. kako jih razporediti po objektih,
- Itn.

Obstaja mnogo delitev lokacijskih problemov glede na različne kriterije oz. zorne kote obravnave tovrstne problematike [5, 15]. Tako lahko opazujemo te probleme na celi ravnini opazovanega območja, tedaj jih običajno imenujemo planarni lokacijski problemi. V veliko primerih se tovrstne probleme opazuje le na določeni mreži s svojo topologijo, tedaj jih imenujemo mrežni lokacijski problemi. V okviru tega se v nekaterih primerih lahko ponudniki storitev pojavijo kjerkoli mreži, torej tako na povezavah, kot tudi v vozliščih. V nasprotju s tem pa je v nekaterih drugi primerih možno locirati ponudnike storitev le v vnaprej določenih točkah mreže, npr. le v vozliščih mreže. Poleg naštetih

delitev lahko lokacijske probleme delimo tudi glede na metodologijo, s katero jih rešujemo, in podobno [5].

Pogostokrat pa se zadovoljimo kar z delitvijo na "zvezne" in "diskretne" lokacijske probleme. Če je mogoče objekte oz. ponudnike storitev locirati v katerikoli točki opazovanega območja, tedaj takšne probleme običajno imenujemo zvezni lokacijski problemi. Drugi razred lokacijskih problemov pa predstavljajo diskretni problemi, kjer je možno lociranje objektov izvršiti samo v točno določenih, vnaprej definiranih točkah [15]. To npr. velja za transportne mreže, ko je večinoma možno locirati prometne terminale zaradi geografskih, urbanističnih, pravnih, ekonomskih in organizacijskih omejitev le v določenem številu vozlišč na mreži [15].

Pri lokacijskih problemih je pomembna tudi delitev glede na tip problema, ki ga obravnavamo, v smislu vrste lociranih objektov. Tako npr. poznamo probleme pokritja "setov", probleme maksimalnega pokritja, probleme median, probleme centrov, itn [5]. Pri problemih pokritja setov moramo minimizirati število zmogljivosti, potrebnih za pokritje vseh zahtev pri danem standardu. Pri problemih maksimalnega pokritja moramo maksimizirati število zahtev, ki jih lahko servisiramo pri danem standardu. Pri problemih median je potrebno locirati en ali več objektov na takšen način, da se minimizira povprečna razdalja med objekti in uporabniki storitev. Pri problemih centrov pa je potrebno locirati en ali več objektov na mreži na takšen način, da se minimizira razdalja do najbolj oddaljenega uporabnika storitev [15].

Kot se izkaže, so modeli za reševanje lokacijskih problemov večinoma zasnovani na osnovi matematičnega programiranja ali teorije grafov. V nadaljevanju si poglejmo osnovno klasifikacijo tovrstnih problemov za primer transportnih mrež [15].

9.1 Klasifikacija lokacijskih problemov

Klasifikacijo lokacijskih problemov lahko podamo na naslednji način, če jih gledamo s stališča transportne mreže [15]:

A. Število objektov na mreži

- Na mreži je potrebno locirati samo en objekt,
- Na mreži je potrebno locirati več objektov.

B. Dovoljena mesta za lociranje objektov

- Objekte je možno locirati v katerikoli točki opazovanega področja (zvezni lokacijski problem),
- Objekte je možno locirati samo v nekaterih, vnaprej definiranih točkah opazovanega področja (diskretni lokacijski problem),

C. Vrsta objektov na mreži

- Mediane, kjer je potrebno locirati en ali več objektov na mreži na takšen način, da se minimizira povprečna razdalja med objekti in uporabniki storitev,
- Centri, kjer je potrebno locirati en ali več objektov na mreži na takšen način, da se minimizira razdalja do najbolj oddaljenega uporabnika storitev,
- Objekti s predhodno definiranimi karakteristikami sistema (kot npr. prepotovane razdalje, časi potovanj, čakalni časi na storitev, itn). Tu gre za takoimenovane probleme zahtevanja.

D. Tip algoritmov za reševanje lokacijskih problemov

- Eksaktni algoritmi,
- Hevristični algoritmi.

E. Število kriterijski funkcij, na osnovi katerih se določa lokacija objektov

- Opravka imamo z eno kriterijsko funkcijo,
- Opravka imamo z več kriterijskimi funkcijami.

9.2 Praktični primeri lokacijskih problemov

Naštejmo nekaj najbolj značilnih primerov lokacijskih problemov [5]:

- Lociranje tovarn, prodajaln in luk,
- Lociranje skladišč in terminalov,
- Lociranje v družbenem sektorju (gasilske postaje, policijske postaje, bolnišnice, pošte, zbiralniki odpadkov, šole, itn),
- Lociranje strojev, delovnih prostorov, itn, v tovarnah,
- Lociranje vodnjakov v poslovnih poslopjih, itn.

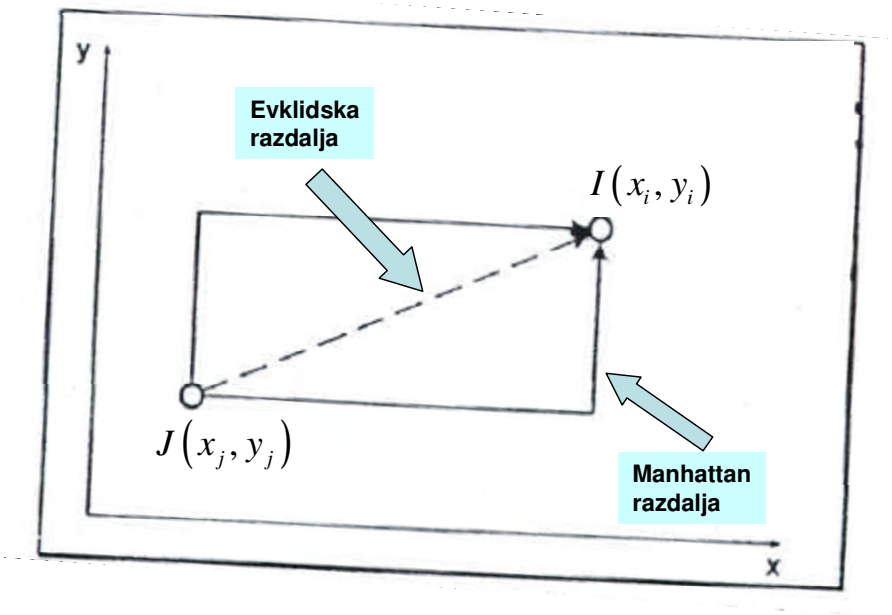
V javnem sektorju najdemo probleme razmeščanja ponudnikov tudi pri načrtovanju okolja, razmeščanju ambulant, postavljanju telekomunikacijskih in energetskih postaj, itn. V zasebnem sektorju pa pride tovrstna problematika v poštev tudi pri razmeščanju trgovinskih in distribucijskih centrov, pri razmeščanju pisarn v poslovnih poslopjih, itn.

Pri javnem sektorju lahko slabe lokacije povečajo verjetnost škode na javnih poslopjih ali celo huje, vodijo v večjo verjetnost izgube življenj. Pri zasebnem sektorju pa slabe odločitve o lokacijah vodijo do povečanih stroškov poslovanja in zmanjšane konkurenčnosti. Če sklenemo, tako v javnem kot zasebnem sektorju lahko uspeh ali neuspeh delovanja ustreznih zmogljivosti oz. objektov servisiranja lahko delno zavisi tudi od lokacij, kamor so te zmogljivosti oz. objekti postavljeni.

Nazadnje še poudarimo, da odločitve o reševanju lokacijskih problemov v veliki meri temeljijo na ekonomiki in dohodkih, še posebej pri zasebnem sektorju. Tako so pri tovrstnih odločitvah zlasti pomembni transportni stroški, stroški prenašanja zalog, pretočnost prometa, odzivni čas, dostopnost, itn [5].

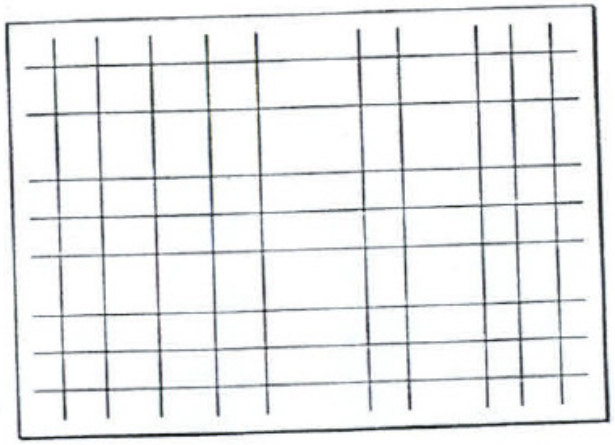
9.3 Mere oddaljenosti in distančna metrika pri lokacijskih problemih

Osnovne vhodne podatke za reševanje kateregakoli lokacijskega problema predstavljajo razdalje med klienti, ki potrebujejo storitev, ter točkami, ki predstavljajo kandidate za servisiranje storitev [5, 15]. V primeru, če imamo npr. opravka s planarnimi oz. zveznimi lokacijskimi problemi, lahko uporabimo metriko z Evklidskimi ali takoimenovanimi "Manhattan" pravokotnimi razdaljami. Oboje predstavljajo poseben primer takoimenovanih l_p razdalj in ju prikazuje slika 158 [15].



Slika 158.: Evklidska in Manhattan razdalja med dvema točkama I in J

Manhattan razdalje med dvema točkama se najpogosteje pojavljajo v mestih z ulično mrežo v obliki rešetke. Tipičen primer takšne mreže je področje New Yorka Manhattan, kjer se ulice sekajo z avenijami pod kotom 90 stopinj (slika 159.) [15].



Slika 159.: Mreža v obliki rešetke, kjer se ulice sekajo pod kotom 90 st.

Manhattan razdalja med dvema točkama $I(x_i, y_i)$ in $J(x_j, y_j)$ je enaka [5, 15]:

$$m(I,J) = |x_i - x_j| + |y_i - y_j| \tag{9.1.}$$

Evklidska razdalja med dvema točkama pa predstavlja dolžino, merjeno po zračni poti. Evklidska razdalja med dvema točkama $I(x_i, y_i)$ in $J(x_j, y_j)$ je potemtakem enaka [15]:

$$e(I,J) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{9.2.}$$

Obe razdalji v izrazih (9.1.) in (9.2.) sta poseben primer l_p razdalj, ki so definirane kot:

$$l_p(I,J) = \left\{ |x_i - x_j|^p + |y_i - y_j|^p \right\}^{\frac{1}{p}} \tag{9.3.}$$

Očitno je, da s postavitvijo p = 1 iz izraza (9.3.) dobimo izraz (9.1.), s postavitvijo p = 2 pa iz izraza (9.3.) dobimo izraz (9.2.).

Kot pokažejo v praksi merjenja izvenmestnih poti, so potovalne razdalje med posameznimi mesti za 10 do 30 % večje od Evklidskih razdalj (npr. v ZDA so večje za 18% pri opazovanju razdalj med velikimi mesti) [15].

V odvisnosti od konteksta lokacijskega problema, ki se rešuje, je torej potrebno na pravilen način definirati metriko za merjenje razdalj. Tako se na primer v večini lokacijskih problemov v letalskem prometu uporablja Evklidska razdalja. Pri kopenskem prometu, ko so lokacijski problemi odvisni od določene potovalne mreže, pa je običajno potrebno Evklidske razdalje povečati za določen procent [15].

Kar se tiče urbanih lokacijskih problemov, je nujno potrebno izračunati tip mreže in nato v odvisnosti od tega tipa uporabiti Manhattan, Evklidsko, ali korigirano Evklidsko metriko merjenja razdalj [15].

V primeru, da se lokacijske probleme rešuje s pomočjo teorije grafov, je seveda potrebno dano transportno mrežo pretvoriti v ekvivalentni graf, pri čemer je potrebno tudi izračunati povprečne razdalje med posameznimi vozlišči (ki so povezane s proporcionalnimi transportnimi stroški). Tedaj je običajno možno lociranje objektov izvršiti samo v točno določenih, vnaprej definiranih točkah (diskretni lokacijski problemi), pri računanju razdalj med posameznimi pari vozlišč pa uporabiti katero izmed tehnik za računanje najkrajših poti (npr. s pomočjo Dijkstrinega algoritma) [5, 15].

V nadaljevanju se bomo omejili le na lokacijske probleme, ki se jih rešuje v okviru teorije grafov, pri čemer se bomo osredotočili zgolj na reševanje problemov Median [5, 15].

9.4 Reševanje lokacijskih problemov tipa Mediane in uporaba teorije grafov

Pri problemih mediane je potrebno locirati eden ali več objektov na mreži, pri čemer se mora minimizirati povprečna oddaljenost (povprečen čas potovanja, povprečni transportni stroški) med objekti (ponudniki storitev) in uporabniki storitev [5, 15]. Problem mediane je še posebej značilen za transportno dejavnost, saj se tovrsten razred problemov pogostokrat pojavlja pri projektiranju različnih distribucijskih sistemov.

Problem mediane se pogostokrat pojavlja tudi v družbenem sektorju [5]. Denimo želi mestna oblast določiti optimalno lokacijo knjižnjice na takšen način, da bo meščanom v povprečju potrebno prehoditi čim krajšo pot od njihovih domov do locirane knjižnjice. Pri reševanju tovrstnega problema je seveda tudi potrebno upoštevati, kako je populacija razpršena po domovih, koliko je npr. v primeru blokovskih naselij ljudi naseljeno v posameznih blokih, in podobno. V takšnem primeru pravimo, da poskušamo minimizirati skupno, s številom uporabnikov storitev uteženo razdaljo, ki jo bodo morali v povprečju uporabniki premagati od svojih domov do locirane knjižnjice.

Pri obravnavi problemov median običajno predpostavimo, da so stroški, potrebni za prepotovanje določenih razdalj, linearno odvisni od teh razdalj. V nadaljevanju si poglejmo matematično formulacijo problema median, ki jo je prvi podal Hakimi leta 1964 [5, 15].

9.4.1 Formulacija problema p median

Denimo imamo opravka z neusmerjeno mrežo, ki ima n vozlišč. Označimo z a_i število zahtev po servisiranju iz vozlišča i. Označimo tudi z d_{ij} razdaljo med vozliščem i in vozliščem j, s p pa število objektov (ponudnikov storitev), ki jih je potrebno locirati na mreži in so lahko locirani v kateremkoli od n vozlišč. Vpeljimo tudi binarne odločitvene spremenljivke x_{ij} in y_j, ki jih definiramo na naslednji način [15]:

$$x_{ij} = \begin{cases} 1, & \text{če je vozlišče } i \text{ postreženo s strani vozlišča } j \\ 0, & \text{če vozlišče } i \text{ ni postreženo s strani vozlišča } j \end{cases} \qquad (9.4.)$$

$$y_j = \begin{cases} 1, & \text{če je locirano vozlišče } j \\ 0, & \text{če ni locirano vozlišče } j \end{cases}$$

Ker težimo k minimizicji skupne potovalne razdalje med objekti in koristniki pri lociranju p objektov, je problem p median možno formulirati na naslednji način [15]:

$$
\begin{aligned}
&\min \textit{imiziraj} \\
&F = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i \cdot d_{ij} \cdot x_{ij} \\
&\textit{pri} \quad \textit{omejitvah}: \\
&\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,2,\ldots,n \\
&\sum_{j=1}^{n} y_j = p \\
&y_j \geq x_{ij}, \quad i, j = 1,2,\ldots,n, i \neq j
\end{aligned}
\tag{9.5.}
$$

Kot je razvidno iz izraza (9.5.), želimo minimizirati skupno (z zahtevami uteženo) potovalno razdaljo med objekti in koristniki storitev. Pri tem prva omejitev pomeni, da je vsak klient postrežen s strani samo enega objekta. Druga omejitev pomeni, da moramo locirati natanko p objektov. Tretja omejitev pa poveže oba tipa odločitvenih spremenljivk in pomeni, da je klient v vozlišču i lahko dodeljen objektu v vozlišču j le v primeru, če je objekt lociran v vozlišču j [5, 15].

Hakimi je tudi pokazal, da obstaja vsaj en set p-median v vozliščih dane mreže, kar pomeni, da se mora p optimalnih lokacij objektov v mreži nahajati izključno v vozliščih te mreže. To dejstvo pa lahko znatno olajša problem iskanja p-median, saj je potrebno preveriti le vse lokacije, ki se nahajajo v vozliščih mreže [5, 15].

Za reševanje problemov p-median je bilo razvitih več algoritmov, ki jih lahko strnemo v naslednje kategorije [5, 15]:

- Algoritem za generiranje množice dopustnih rešitev,
- Algoritmi, zasnovani na osnovi teorije grafov,
- Hevristični algoritmi, ter
- Algoritmi, zasnovani na osnovi matematičnega porgramiranja.

Algoritem za generiranje množice dopustnih rešitev obravnava iskanje vseh možnih rešitev lokacij p-median, izračunavanje pripadajočih vrednosti kriterijske funkcije F v

izrazu (9.5.), ter določitev optimalne rešitve. Jasno je, da je tovrsten algoritem uporaben le v primeru mrež z manjšim številom vozlišč, na katerih je potrebno locirati manjše število objektov. Število vseh možnih razporeditev p objektov na mreži z n vozlišči je namreč enako številu kombinacij brez ponavljanja pri n elementih p-tega razreda, kar pomeni število $\binom{n}{p}$. Slednje pa ni preveliko le v primeru manjših mrež [5, 15].

Pokazati se da, da je problem p-mediane rešljiv v polinomskem času le na mreži v obliki drevesa [5]. Pri generalnih grafih pa se izkaže, da takšni problemi spadajo v kategorijo NP-težkih problemov. V takšnih primerih je morda bolje uporabiti hevristične algoritme, kot npr. [5]: miopični algoritem, algoritem izmenjalne hevristike, ali algoritem preiskovanja soseščine, ki spadajo v enega izmed dveh razredov hevrističnih algoritmov: razred konstrukcijskih algoritmov, ali razred izboljševalnih algoritmov [5].

V nadaljevanju se bomo osredotočili le na obravnavo takoimenovanega Hakimijevega algoritma za neusmerjene mreže, ki generira množico dopustnih rešitev. Pri tem bomo posvetili pozornost zgolj določanju lokacije ene mediane. Obravnavo za več median ter princip delovanja drugih algoritmov za določanje optimalnih median v mreži pa si lahko bralec pogleda v literaturi [5, 10, 15].

9.4.2 Hakimijev algoritem za reševanje problemov 1-mediane

Hakimijev algoritem je dokaj enostaven algoritem, ki generira množico dopustnih rešitev ter določi lokacijo ene mediane v primeru neusmerjene mreže. Algoritem sestavljajo naslednji koraki [5, 15]:

KORAK 1: Izračunaj dolžine najkrajših poti d_{ij} med vsemi pari vozlišč (i, j) mreže G in prikaži jih v matriki najkrajših razdalj D. Pri tem naj vozlišča i predstavljajo možne lokacije za mediano, vozlišča j pa predstavljajo lokacije klientov, ki zahtevajo storitev.

KORAK 2: Pomnoži j-ti stolp matrike najkrajših poti D s številom zahtev za postrežbo a_j *v vozlišču j. Element* $a_j \cdot d_{ij}$ *matrike* $D' = \left[a_j \cdot d_{ij} \right]$ *predstavlja uteženo razdaljo, ki jo morajo premagati uporabniki iz vozlišča j, da dospejo do potencialnega klienta v vozlišču i.*

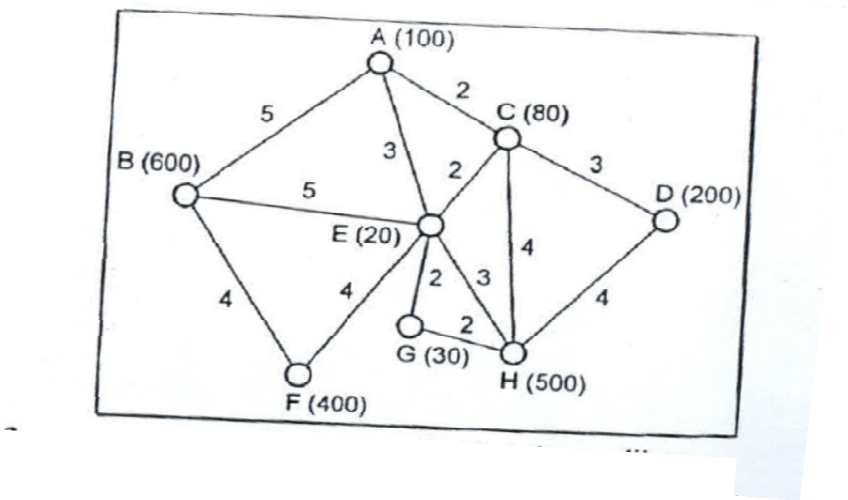
KORAK 3: Izvedi seštevanje preko vsake vrstice i matrike $D' = \left[a_j \cdot d_{ij} \right]$*. Pri tem izraz* $\sum_{j=1}^{n} a_j \cdot d_{ij}$ *predstavlja skupno uteženo razdaljo, ki jo morajo premagati uporabniki v primeru, če je objekt lociran v vozlišču i.*

KORAK 4: Vozlišče, katerega vsota $\sum_{j=1}^{n} a_j \cdot d_{ij}$ *je najmanjša, naj bo optimalna lokacija za mediano.*

Princip delovanja Hakimijevega algoritma za lociranje ene mediane bomo dodatno osvetlili na naslednjem primeru [15].

Primer:

Za dano mrežo na sliki 160 rešite lokacijski problem ene mediane.



Slika 160.: Mreža, kjer rešujemo lokacijski problem ene mediane.

Vozlišča transportne mreže A, B, …, H so na sliki 160 ustrezno označena. Za vsakega izmed njih je v oklepaju tudi podano število zahtev uporabnikov storitev na dotičnem vozlišču. Na sliki 160 so tudi označene uteži (dolžine) vseh povezav med vozlišči. Problem lahko formuliramo na naslednji način. Kje naj se locira objekt, ki bo nudil storitve, da bo skupna utežena razdalja, ki jo bodo morali uporabniki storitev premagati iz ostalih vozlišč do tega objekta, najmanjša možna.

Očitno je v danem primeru 8 mest oz. lokacij kandidatov, kamor bi lahko locirali naš objekt (saj je 8 vozlišč, optimalno lokacijo pa seveda iščemo v vozliščih).

Če izračunamo matriko D najkrajših poti d_{ij} med vsemi pari vozlišč (i, j) mreže na sliki 160 (npr. z uporabo Dijkstrinega algoritma), dobimo naslednji rezultat:

$$D = \left[d_{ij} \right] = \left[d_{i1} \quad d_{i2} \quad d_{i3} \quad d_{i4} \quad d_{i5} \quad d_{i6} \quad d_{i7} \quad d_{i8} \right]$$

$$D = \left[d_{ij} \right] = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} A & B & C & D & E & F & G & H \end{array} \\ \left[\begin{array}{cccccccc} 0 & 5 & 2 & 5 & 3 & 7 & 5 & 6 \\ 5 & 0 & 7 & 10 & 5 & 4 & 7 & 8 \\ 2 & 7 & 0 & 3 & 2 & 6 & 4 & 4 \\ 5 & 10 & 3 & 0 & 5 & 9 & 6 & 4 \\ 3 & 5 & 2 & 5 & 0 & 4 & 2 & 3 \\ 7 & 4 & 6 & 9 & 4 & 0 & 6 & 7 \\ 5 & 7 & 4 & 6 & 2 & 6 & 0 & 2 \\ 6 & 8 & 4 & 4 & 3 & 7 & 2 & 0 \end{array} \right] \end{array} \qquad (9.6.)$$

V nadaljevanju tvorimo vektor A s številom zahtev za postrežbo a_j v vozlišču j (glej sliko 160, izraze v oklepaju):

$$A = \left[a_j \right]^T = \left[a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8 \right] \qquad (9.7.)$$

$$A = \left[a_j \right]^T = \begin{array}{cccccccc} A & B & C & D & E & F & G & H \end{array}$$

$$\left[100 \quad 600 \quad 80 \quad 200 \quad 20 \quad 400 \quad 30 \quad 500 \right]^T$$

Nato tvorimo matriko $D' = \left[a_j \cdot d_{ij} \right]$ na naslednji način [15]:

$$D' = \left[a_j \cdot d_{ij} \right] = \\ = \left[a_1 \cdot d_{i1} \quad a_2 \cdot d_{i2} \quad a_3 \cdot d_{i3} \quad a_4 \cdot d_{i4} \quad a_5 \cdot d_{i5} \quad a_6 \cdot d_{i6} \quad a_7 \cdot d_{i7} \quad a_8 \cdot d_{i8} \right] \tag{9.8.}$$

pri čemer dobimo:

$$D' = \begin{bmatrix} 100 \cdot 0 & 600 \cdot 5 & 80 \cdot 2 & 200 \cdot 5 & 20 \cdot 3 & 400 \cdot 7 & 30 \cdot 5 & 500 \cdot 6 \\ 100 \cdot 5 & 600 \cdot 0 & 80 \cdot 7 & 200 \cdot 10 & 20 \cdot 5 & 400 \cdot 4 & 30 \cdot 7 & 500 \cdot 8 \\ 100 \cdot 2 & 600 \cdot 7 & 80 \cdot 0 & 200 \cdot 3 & 20 \cdot 2 & 400 \cdot 6 & 30 \cdot 4 & 500 \cdot 4 \\ 100 \cdot 5 & 600 \cdot 10 & 80 \cdot 3 & 200 \cdot 0 & 20 \cdot 5 & 400 \cdot 9 & 30 \cdot 6 & 500 \cdot 4 \\ 100 \cdot 3 & 600 \cdot 5 & 80 \cdot 2 & 200 \cdot 5 & 20 \cdot 0 & 400 \cdot 4 & 30 \cdot 2 & 500 \cdot 3 \\ 100 \cdot 7 & 600 \cdot 4 & 80 \cdot 6 & 200 \cdot 9 & 20 \cdot 4 & 400 \cdot 0 & 30 \cdot 6 & 500 \cdot 7 \\ 100 \cdot 5 & 600 \cdot 7 & 80 \cdot 4 & 200 \cdot 6 & 20 \cdot 2 & 400 \cdot 6 & 30 \cdot 0 & 500 \cdot 2 \\ 100 \cdot 6 & 600 \cdot 8 & 80 \cdot 4 & 200 \cdot 4 & 20 \cdot 3 & 400 \cdot 7 & 30 \cdot 2 & 500 \cdot 0 \end{bmatrix} \tag{9.9.}$$

kar nam da rezultat:

$$D' = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccccccc} A & B & C & D & E & F & G & H \end{array} \\ \begin{bmatrix} 0 & 3000 & 160 & 1000 & 60 & 2800 & 150 & 3000 \\ 500 & 0 & 560 & 2000 & 100 & 1600 & 210 & 4000 \\ 200 & 4200 & 0 & 600 & 40 & 2400 & 120 & 2000 \\ 500 & 6000 & 240 & 0 & 100 & 3600 & 180 & 2000 \\ 300 & 3000 & 160 & 1000 & 0 & 1600 & 60 & 1500 \\ 700 & 2400 & 480 & 1800 & 80 & 0 & 180 & 3500 \\ 500 & 4200 & 320 & 1200 & 40 & 2400 & 0 & 1000 \\ 600 & 4800 & 320 & 800 & 60 & 2800 & 60 & 0 \end{bmatrix} \end{array} \tag{9.10.}$$

V nadaljevanju še tvorimo vsote elementov vrstičnih vektorjev v izrazu (9.10.), kar nam da skupne utežene razdalje, katere bi morali premagati uporabniki storitev, če bi bil objekt lociran v enem izmed vozlišč, ki pripadajo dotičnim vrsticam. Tako dobimo naslednje izraze [15]:

$$
\begin{aligned}
vsota(A) &= 0+3000+160+1000+60+2800+150+3000=10170\\
vsota(B) &= 500+0+560+2000+100+1600+210+4000=8970\\
vsota(C) &= 200+4200+0+600+40+2400+120+2000=9560\\
vsota(D) &= 500+6000+240+0+100+3600+180+2000=12620\\
vsota(E) &= 300+3000+160+1000+0+1600+60+1500=7620\\
vsota(F) &= 700+2400+480+1800+80+0+180+3500=9140\\
vsota(G) &= 500+4200+320+1200+40+2400+0+1000=9660\\
vsota(H) &= 600+4800+320+800+60+2800+60+0=9440
\end{aligned}
\qquad (9.11.)
$$

V tabeli 31 so na osnovi izrazov (9.11.) podani skupni uteženi potovalni kilometri za primere, ko je objekt lociran v enem izmed danih vozlišč transportne mreže na sliki 160.

Lokacija objekta je v vozlišču	Število uteženih potovalnih km	Lokacija objekta je v vozlišču	Število uteženih potovalnih km
A	10170	E	7620
B	8970	F	9140
C	9560	G	9660
D	12620	H	9440

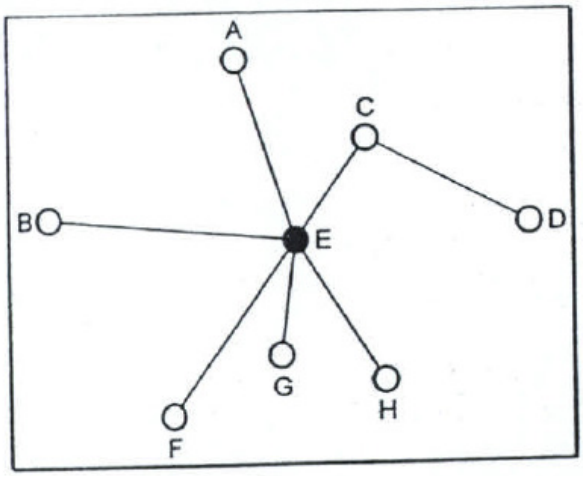
Tabela 31: Skupni uteženi potovalni kilometri za primere, ko je objekt lociran v enem izmed danih vozlišč transportne mreže na sliki 160.

Očitno je, da je najmanjše število skupnih uteženih potovalnih kilometrov doseženo tedaj, če objekt (mediano) lociramo v vozlišču E (7620 km). Na sliki 161 je prikazan končni rezultat obravnavanega problema, ko objekt (mediano) lociramo v vozlišču E.

Potek računalniškega programa v programskem orodju Matlab, s katerim lahko opravimo izračune (9.8.) do (9.11.) ter izračunamo lokacijo optimalne mediane (v vozlišču E), je podan v prilogi 10.1.

Prikazani algoritem je mogoče uporabiti tudi v primerih, ko imamo opravka z usmerjenimi mrežami. Tedaj je potrebno razlikovati situacije, ko uporabniki gredo v objekt, da bi bili postreženi, od situacij, ko transportna sredstva gredo iz objekta in nato v

vozlišča, ki zahtevajo servisiranje. V prvem primeru imamo opravka z vhodno mediano, v drugem primeru pa opravka z izhodno mediano. Podrobnosti o uporabi obravnavanega algoritma pri usmerjenih mrežah lahko bralec zasledi v literaturi [5, 10, 15].

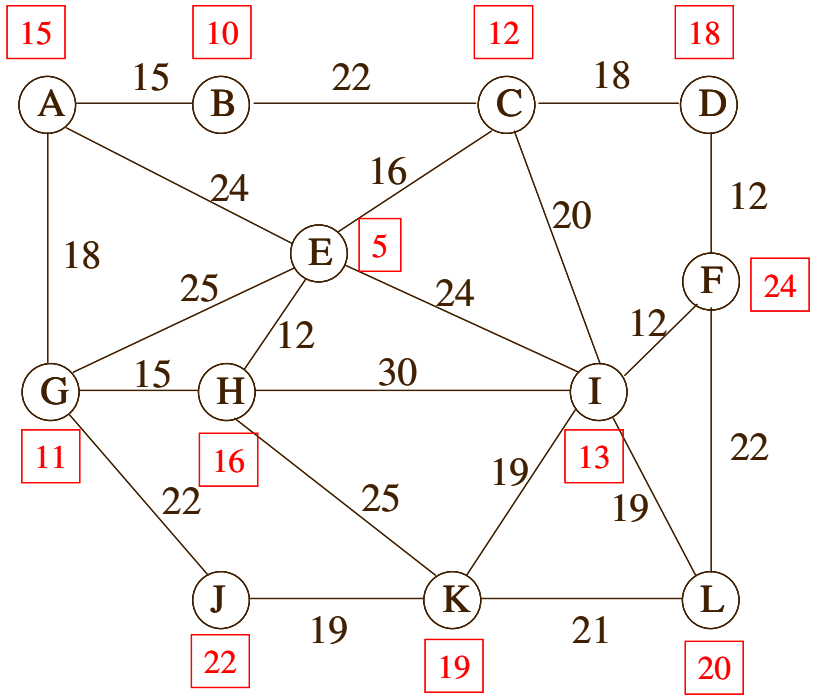


Slika 161: Mediana, ki smo jo locirali v vozlišču E.

V nadaljevanju utrdimo naše znanje še na enem primeru.

Primer:

Za dano mrežo na sliki 162 rešite lokacijski problem ene mediane.



Slika 162: Mreža, kjer rešujemo lokacijski problem ene mediane.

Vozlišča transportne mreže A, B, …, L so na sliki 162 ustrezno označena. Za vsakega izmed njih je v kvadratnem prostorčku tudi podano število zahtev uporabnikov storitev na dotičnem vozlišču. Na sliki 162 so tudi označene uteži (dolžine) vseh povezav med vozlišči.

Očitno je v danem primeru 12 mest oz. lokacij kandidatov, kamor bi lahko locirali naš objekt (saj je 12 vozlišč, optimalno lokacijo pa seveda iščemo v vozliščih).

Če izračunamo matriko D najkrajših poti d_{ij} med vsemi pari vozlišč (i, j) mreže na sliki 162 (npr. z uporabo Dijkstrinega algoritma), dobimo naslednji rezultat:

$$D=\left[d_{ij}\right]=\left[d_{i1}\ \ d_{i2}\ \ d_{i3}\ \ d_{i4}\ \ d_{i5}\ \ d_{i6}\ \ d_{i7}\ \ d_{i8}\ \ d_{i9}\ \ d_{i10}\ \ d_{i11}\ \ d_{i12}\right]$$

$$D=\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \\ K \\ L \end{array}\begin{array}{c} \begin{array}{cccccccccccc} A & B & C & D & E & F & G & H & I & J & K & L \end{array} \\ \left[\begin{array}{cccccccccccc} 0 & 15 & 37 & 55 & 24 & 60 & 18 & 33 & 48 & 40 & 58 & 67 \\ 15 & 0 & 22 & 40 & 38 & 52 & 33 & 48 & 42 & 55 & 61 & 61 \\ 37 & 22 & 0 & 18 & 16 & 30 & 41 & 28 & 20 & 58 & 39 & 39 \\ 55 & 40 & 18 & 0 & 34 & 12 & 59 & 46 & 24 & 62 & 43 & 34 \\ 24 & 38 & 16 & 34 & 0 & 36 & 25 & 12 & 24 & 47 & 37 & 43 \\ 60 & 52 & 30 & 12 & 36 & 0 & 57 & 42 & 12 & 50 & 31 & 22 \\ 18 & 33 & 41 & 59 & 25 & 57 & 0 & 15 & 45 & 22 & 40 & 61 \\ 33 & 48 & 28 & 46 & 12 & 42 & 15 & 0 & 30 & 37 & 25 & 46 \\ 48 & 42 & 20 & 24 & 24 & 12 & 45 & 30 & 0 & 38 & 19 & 19 \\ 40 & 55 & 58 & 62 & 47 & 50 & 22 & 37 & 38 & 0 & 19 & 40 \\ 58 & 61 & 39 & 43 & 37 & 31 & 40 & 25 & 19 & 19 & 0 & 21 \\ 67 & 61 & 39 & 34 & 43 & 22 & 61 & 46 & 19 & 40 & 21 & 0 \end{array}\right] \end{array} \qquad (9.12.)$$

V nadaljevanju tvorimo vektor A s številom zahtev za postrežbo a_j v vozlišču j (glej sliko 162, izraze v oklepaju):

$$A = \left[a_j \right]^T = \left[a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8 \quad a_9 \quad a_{10} \quad a_{11} \quad a_{12} \right] \tag{9.13.}$$

$$\begin{array}{cccccccccccc} A & B & C & D & E & F & G & H & I & J & K & L \end{array}$$

$$A = \left[a_j \right]^T = \left[15 \quad 10 \quad 12 \quad 18 \quad 5 \quad 24 \quad 11 \quad 16 \quad 13 \quad 22 \quad 19 \quad 20 \right]^T$$

Nato tvorimo matriko $D' = \left[a_j \cdot d_{ij} \right]$ na naslednji način:

$$D' = \left[a_j \cdot d_{ij} \right] = \tag{9.14.}$$

$$= \left[a_1 \cdot d_{i1} \quad a_2 \cdot d_{i2} \quad a_3 \cdot d_{i3} \quad a_4 \cdot d_{i4} \quad a_5 \cdot d_{i5} \quad a_6 \cdot d_{i6} \quad a_7 \cdot d_{i7} \quad a_8 \cdot d_{i8} \quad a_9 \cdot d_{i9} \quad a_{10} \cdot d_{i10} \quad a_{11} \cdot d_{i11} \quad a_{12} \cdot d_{i12} \right]$$

pri čemer dobimo:

$$D' = \begin{bmatrix} 15{\cdot}0 & 10{\cdot}15 & 12{\cdot}37 & 18{\cdot}55 & 5{\cdot}24 & 24{\cdot}60 & 11{\cdot}18 & 16{\cdot}33 & 13{\cdot}48 & 22{\cdot}40 & 19{\cdot}58 & 20{\cdot}67 \\ 15{\cdot}15 & 10{\cdot}0 & 12{\cdot}22 & 18{\cdot}40 & 5{\cdot}38 & 24{\cdot}52 & 11{\cdot}33 & 16{\cdot}48 & 13{\cdot}42 & 22{\cdot}55 & 19{\cdot}61 & 20{\cdot}61 \\ 15{\cdot}37 & 10{\cdot}22 & 12{\cdot}0 & 18{\cdot}18 & 5{\cdot}16 & 24{\cdot}30 & 11{\cdot}41 & 16{\cdot}28 & 13{\cdot}20 & 22{\cdot}58 & 19{\cdot}39 & 20{\cdot}39 \\ 15{\cdot}55 & 10{\cdot}40 & 12{\cdot}18 & 18{\cdot}0 & 5{\cdot}34 & 24{\cdot}12 & 11{\cdot}59 & 16{\cdot}46 & 13{\cdot}24 & 22{\cdot}62 & 19{\cdot}43 & 20{\cdot}34 \\ 15{\cdot}24 & 10{\cdot}38 & 12{\cdot}16 & 18{\cdot}34 & 5{\cdot}0 & 24{\cdot}36 & 11{\cdot}25 & 16{\cdot}12 & 13{\cdot}24 & 22{\cdot}47 & 19{\cdot}37 & 20{\cdot}43 \\ 15{\cdot}60 & 10{\cdot}52 & 12{\cdot}30 & 18{\cdot}12 & 5{\cdot}36 & 24{\cdot}0 & 11{\cdot}57 & 16{\cdot}42 & 13{\cdot}12 & 22{\cdot}50 & 19{\cdot}31 & 20{\cdot}22 \\ 15{\cdot}18 & 10{\cdot}33 & 12{\cdot}41 & 18{\cdot}59 & 5{\cdot}25 & 24{\cdot}57 & 11{\cdot}0 & 16{\cdot}15 & 13{\cdot}45 & 22{\cdot}22 & 19{\cdot}40 & 20{\cdot}61 \\ 15{\cdot}33 & 10{\cdot}48 & 12{\cdot}28 & 18{\cdot}46 & 5{\cdot}12 & 24{\cdot}42 & 11{\cdot}15 & 16{\cdot}0 & 13{\cdot}30 & 22{\cdot}37 & 19{\cdot}25 & 20{\cdot}46 \\ 15{\cdot}48 & 10{\cdot}42 & 12{\cdot}20 & 18{\cdot}24 & 5{\cdot}24 & 24{\cdot}12 & 11{\cdot}45 & 16{\cdot}30 & 13{\cdot}0 & 22{\cdot}38 & 19{\cdot}19 & 20{\cdot}19 \\ 15{\cdot}40 & 10{\cdot}55 & 12{\cdot}58 & 18{\cdot}62 & 5{\cdot}47 & 24{\cdot}50 & 11{\cdot}22 & 16{\cdot}37 & 13{\cdot}38 & 22{\cdot}0 & 19{\cdot}19 & 20{\cdot}40 \\ 15{\cdot}58 & 10{\cdot}61 & 12{\cdot}39 & 18{\cdot}43 & 5{\cdot}37 & 24{\cdot}31 & 11{\cdot}40 & 16{\cdot}25 & 13{\cdot}19 & 22{\cdot}19 & 19{\cdot}0 & 20{\cdot}21 \\ 15{\cdot}67 & 10{\cdot}61 & 12{\cdot}39 & 18{\cdot}34 & 5{\cdot}43 & 24{\cdot}22 & 11{\cdot}61 & 16{\cdot}46 & 13{\cdot}19 & 22{\cdot}40 & 19{\cdot}21 & 20{\cdot}0 \end{bmatrix} \tag{9.15.}$$

kar nam da rezultat:

$$D' = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \\ K \\ L \end{array} \begin{array}{c} \begin{array}{cccccccccccc} A & B & C & D & E & F & G & H & I & J & K & L \end{array} \\ \begin{bmatrix} 0 & 150 & 444 & 990 & 120 & 1440 & 198 & 528 & 624 & 880 & 1102 & 1340 \\ 225 & 0 & 264 & 720 & 190 & 1248 & 363 & 768 & 546 & 1210 & 1159 & 1220 \\ 555 & 220 & 0 & 324 & 80 & 720 & 451 & 448 & 260 & 1276 & 741 & 780 \\ 825 & 400 & 216 & 0 & 170 & 288 & 649 & 736 & 312 & 1364 & 817 & 680 \\ 360 & 380 & 192 & 612 & 0 & 864 & 275 & 192 & 312 & 1034 & 703 & 860 \\ 900 & 520 & 360 & 216 & 180 & 0 & 627 & 672 & 156 & 1100 & 589 & 440 \\ 270 & 330 & 492 & 1062 & 125 & 1368 & 0 & 240 & 585 & 484 & 760 & 1220 \\ 495 & 480 & 336 & 828 & 60 & 1008 & 165 & 0 & 390 & 814 & 475 & 920 \\ 720 & 420 & 240 & 432 & 120 & 288 & 495 & 480 & 0 & 836 & 361 & 380 \\ 600 & 550 & 696 & 1116 & 235 & 1200 & 242 & 592 & 494 & 0 & 361 & 800 \\ 870 & 610 & 468 & 774 & 185 & 744 & 440 & 400 & 247 & 418 & 0 & 420 \\ 1005 & 610 & 468 & 612 & 215 & 528 & 671 & 736 & 247 & 880 & 399 & 0 \end{bmatrix} \end{array} \tag{9.16.}$$

V nadaljevanju še tvorimo vsote elementov vrstičnih vektorjev v izrazu (9.16.), kar nam da skupne uteženе razdalje, katere bi morali premagati uporabniki storitev, če bi bil objekt lociran v enem izmed vozlišč, ki pripadajo dotičnim vrsticam. Tako dobimo naslednje izraze:

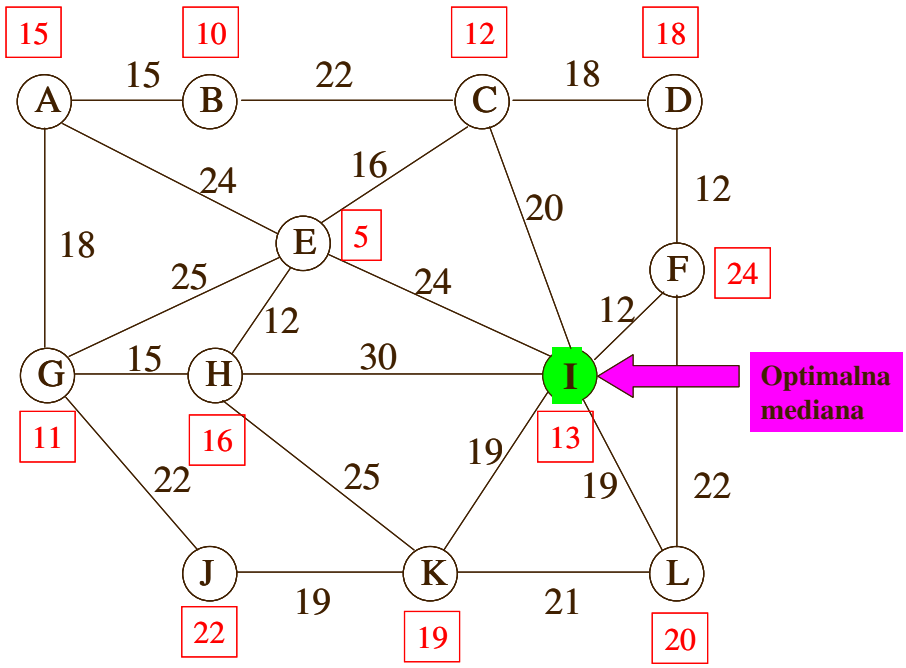
$$\begin{aligned}
&vsota\,(A) = 0 + 150 + 444 + 990 + 120 + 1440 + 198 + 528 + 624 + 880 + 1102 + 1340 = 7816 \\
&vsota(B)=225+0+264+720+190+1248+363+768+546+1210+1159+1220=7913 \\
&vsota(C)=...=5855 \\
&vsota(D)=...=6457 \\
&vsota(E)=...=5784 \\
&vsota(F)=...=5760 \\
&vsota(G)=...=6936 \\
&vsota(H)=...=5971 \\
&vsota(I)=...=4772 \\
&vsota(J)=...=6886 \\
&vsota(K)=...=870+610+468+774+185+744+440+400+247+418+0+420= 5576 \\
&vsota\,(L) = ...1005 + 610 + 468 + 612 + 215 + 528 + 671 + 736 + 247 + 880 + 399 + 0 = 6371
\end{aligned} \tag{9.17.}$$

V tabeli 32 so na osnovi izrazov (9.17.) podani skupni uteženi potovalni kilometri za primere, ko je objekt lociran v enem izmed danih vozlišč transportne mreže na sliki 162.

LOKACIJA OBJEKTA JE V VOZLIŠČU	ŠTEVILO UTEŽENIH POTOVALNIH km
A	7816
B	7913
C	5855
D	6457
E	5784
F	5760
G	6936
H	5971
I	**4772**
J	6886
K	5576
L	6371

Tabela 32: Skupni uteženi potovalni kilometri za primere, ko je objekt lociran v enem izmed danih vozlišč transportne mreže na sliki 162.

Očitno je, da je najmanjše število skupnih uteženih potovalnih kilometrov doseženo tedaj, če objekt (mediano) lociramo v vozlišču I (4772 km). Na sliki 163 je prikazan končni rezultat obravnavanega problema, ko objekt (mediano) lociramo v vozlišču I.



Slika 163: Optimalna mediana, ki smo jo locirali v vozlišču I.

Potek računalniškega programa v programskem orodju Matlab, s katerim lahko opravimo izračune (9.14.) do (9.17.) ter izračunamo lokacijo optimalne mediane (v vozlišču I), je podan v prilogi 10.2.

9.5 Sklep obravnave lokacijskih problemov

Glavni namen 9. poglavja je bil zgolj seznaniti bralca s sicer zelo obširnim področjem obravnave lokacijskih problemov in veliko uporabno vrednostjo reševanja teh problemov. Zato se v tehnične podrobnosti delovanja posameznih algoritmov za reševanje tovrstnih problemov nismo spuščali, razen prikaza delovanja Hakimijevega algoritma za reševanje problemov 1-median. Želeli smo namreč le ponazoriti, kako se izredno dobro obnese teorija grafov tudi pri reševanju lokacijskih problemov. Kar se tiče podrobnosti številnih metod in tehnik, ki so bile razvite za vse štiri glavne tipe lokacijskih problemov, to je problema pokritja "setov", problema maksimalnega pokritja, problema median in problema centrov, pa se bralec lahko z detajli seznani v literaturi [5, 10, 15].

10 PRILOGE

10.1 Potek računalniškega programa v Matlabu, s katerim izračunamo lokacijo optimalne 1-mediane (1. primer)

```
% program za 1-mediano (1. primer):

clear
clc

vozlisca = ['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H'];

d1 = [0 5 2 5 3 7 5 6]'
d2 = [5 0 7 10 5 4 7 8]'
d3 = [2 7 0 3 2 6 4 4]'
d4 = [5 10 3 0 5 9 6 4]'
d5 = [3 5 2 5 0 4 2 3]'
d6 = [7 4 6 9 4 0 6 7]'
d7 = [5 7 4 6 2 6 0 2]'
d8 = [6 8 4 4 3 7 2 0]'

D = [d1 d2 d3 d4 d5 d6 d7 d8]       % matrika najkrajših poti

A = [100 600 80 200 20 400 30 500]' % št.zahtev za postrežbo
                                    % na vozliščih
D1 = [];

for i = 1:size(A,1)
    D1 = [D1 [A(i)*D(:,i)]];
end

D1

vsote = [];
for i = 1:size(A,1)
    vsote(i) = sum(D1(i,:));
end

vsote

disp('optimalna mediana:')
[minsuma,Imin] = min(vsote)
disp('je v vozliscu:')
vozlisca(Imin)
```

10.2 Potek računalniškega programa v Matlabu, s katerim izračunamo lokacijo optimalne 1-mediane (2. primer)

```
% program za 1-mediano (2. primer):

clear
clc

vozlisca = ['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L'];

d1 = [0 15 37 55 24 60 18 33 48 40 58 67]'
d2 = [15 0 22 40 38 52 33 48 42 55 61 61]'
d3 = [37 22 0 18 16 30 41 28 20 58 39 39]'
d4 = [55 40 18 0 34 12 59 46 24 62 43 34]'
d5 = [24 38 16 34 0 36 25 12 24 47 37 43]'
d6 = [60 52 30 12 36 0 57 42 12 50 31 22]'
d7 = [18 33 41 59 25 57 0 15 45 22 40 61]'
d8 = [33 48 28 46 12 42 15 0 30 37 25 46]'
d9 = [48 42 20 24 24 12 45 30 0 38 19 19]'
d10= [40 55 58 62 47 50 22 37 38 0 19 40]'
d11 =[58 61 39 43 37 31 40 25 19 19 0 21]'
d12 =[67 61 39 34 43 22 61 46 19 40 21 0]'

D = [d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12] % matrika najkrajših poti

A = [15 10 12 18 5 24 11 16 13 22 19 20]'    % št. zahtev za postrežbo
                                             % na vozliščih
D1 = [];

for i = 1:size(A,1)
    D1 = [D1 [A(i)*D(:,i)]];
end

D1

vsote = [];
for i = 1:size(A,1)
    vsote(i) = sum(D1(i,:));
end

vsote

disp('optimalna mediana:')
[minsuma,Imin] = min(vsote)
disp('je v vozliscu:')
vozlisca(Imin)
```

LITERATURA

[1] Ahuja, R. K.: Network flows: Theory, algorithms, and Applications, Prentice Hall, 1993.

[2] Aldous J. M., Wilson R. J.: Graphs and Applications: An Introductory Approach (with CD-ROM), Springer, 2000.

[3] Balakrishnan V.; Network Optimization (Chapman Hall/CRC Mathematics Series), Chapman and Hall/CRC, 1 edition, 1995.

[4] Čižman A.: Operacijske raziskave, Teorija in uporaba v organizaciji, Moderna organizacija Kranj, 2004.

[5] Daskin M. S.: Network and Discrete Location: Models, Algorithms, and Applications, Wiley-Interscience, Har/Dskt edition, 1995.

[6] Dragan D.: Upravljanje logističnih sistemov, učbenik, Fakulteta za logistiko, Celje, 2009.

[7] Eiselt H.A., Sandblom C. L.: Integer programming and Network Models, Springer, 2010.

[8] Fogiel M., The Operations Research Problem Solver, Research And Education Association, New Jersey, 1989.

[9] Hillier, F.S.: Introduction to Operations Research, McGraw-Hill, 2001.

[10] Mirchandani P. B., Francis R.L.: Discrete Location Theory, Wiley-Interscience, 1990.

[11] Salkin, M. S. : Integer Programming, Addison-Wesley, 1974.

[12] Sierksma G., Ghosh D.: Networks in Action: Text and Computer Exercises in Network Optimization (International Series in Operations Research & Management Science), Springer, 2009.

[13] Syslo M. M.: Discrete Optimization Algorithms: with Pascal Programs (Dover Books on Mathematics), Dover Publications, 2006.

[14] Taha H.: Operations Research, An Introduction, Sixth Edition, Prentice-Hall, 1997.

[15] Teodorovič D.: Transportne mreže: algoritamski pristup, Univerzitet u Beogradu, 1996.

[16] Wilson R. J., Watkins J. J.: Uvod v teorijo grafov, Društvo matematikov, fizikov in astronomov Slovenije, 1997.

[17] Winston, W., L. : Operations Research, Applications and Algorithms, Duxbury press, International Thomson Publishing, 1994, ISBN 0-534-20971-8.

[18] Zadnik Stirn L.: Metode operacijskih raziskav za poslovno odločanje, Novo Mesto, 2001.

[19] Žerovnik J., Vesel A.: Osnove teorije grafov in diskretne optimizacije, Univerza v Mariboru, Fakulteta za strojništvo, Maribor, 2003.