Trgovski potnik, dinamično programiranje

1. Splošna predstavitev problema

Problem trgovskega potnika (TSP) spada v področje kombinatorične optimizacije, ki je še vedno aktualen v operacijskih raziskavah in teoretičnem računalništvu. Problem trgovskega potnika je naslednji: Dana je množica mest, s povezavami med njimi, ki pa so usmerjene, kar pomeni, da pot iz mesta A v mesto B ni nujno enako dolga kot pot iz mesta B v mesto A. Naša naloga je poiskati najkrajši možni obhod vseh mest, tako da vsako mesto obiščemo natanko enkrat.

Problem se je prvič pojavil v matematiki leta 1930 in je eden najbolj proučevanih problemov v optimizaciji. Uporablja se kot merilo za mnoge optimizacijske metode. Čeprav gre za računsko težak problem, je znanih veliko hevrističnih in eksaktnih rešitev, ki lahko rešijo tudi nekatere primere z več deset tisoč mesti.

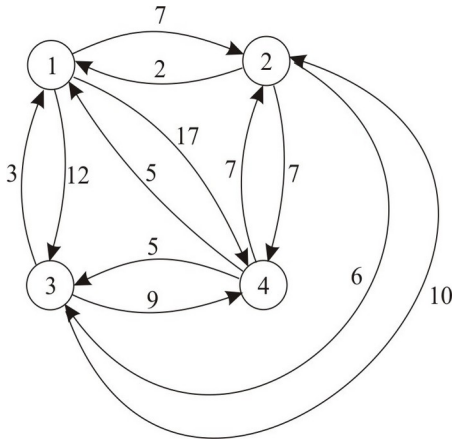
Problem trgovskega potnika ima veliko aplikacij celo v svoji osnovni obliki, kot na primer v logistiki in načrtovanju in celo pri izdelavi mikročipov. V nekoliko spremenjeni obliki pa ga najdemo celo na področju genetike. Velikokrat se osnovnemu problemu dodajo še omejitve resursov, ki jih imamo na voljo in omejitve v času, v katerem je treba nalogo opraviti, kar naredi osnovni problem še težji.

Problem TSP lahko opišemo s teorijo grafov, pri čemer mesta predstavimo z vozlišči, poti med njimi pa s povezavami. Ker so poti enosmerne, problem predstavimo z usmerjenim uteženim grafom.

Kot smo že navajeni tudi ta graf predstavimo z matriko sosednosti, ki je kvadratna matrika, ki pa v primeru usmerjenega grafa ni več zgornje ali spodnje trikotna. Za primer predpostavimo, da imamo poln graf na štirih vozliščih. Matrika sosednosti **C**, ki ta graf definira, je naslednja:

$$C=\begin{bmatrix} 0 & 7 & 12 & 17 \\ 2 & 0 & 6 & 7 \\ 3 & 10 & 0 & 9 \\ 5 & 7 & 5 & 0 \end{bmatrix}$$

Graf, ki je z matriko **C** določen, je prikazan na sliki 1. Naša naloga je, da v danem grafu poiščemo najcenejši obhod, v katerem se bodo pojavila vsa vozlišča natanko enkrat. Ker iščemo cikel je popolnoma vseeno, katero od vozlišč vzamemo kot izhodišče našega iskanja, zato bomo vzeli kar prvo vozlišče, to je vozlišče 1. Vozlišča hranimo v množici V. Ob upoštevanju našega problema, lahko gremo iz vozlišča 1 v katerokoli vozlišče k iz množice V-{1}. Krožna pot bo torej sestavljena iz poti iz $1 \rightarrow k$ in iz poti od k preko vseh ostalih vozlišč spet v vozlišče 1 tako, da bo vsako vozlišče obiskano le enkrat.



Slika 1: Graf, določen z matriko sosednosti **C**

Če je pot iz *k* v 1 optimalna, bo skupna krožna pot optimalna le tedaj, če je vozlišče *k* izbrano tako, da je pot iz 1→ *k* tudi najkrajša. Če to sklepanje posplošimo, imejmo najkrajšo pot, ki se začne v vozlišču *i* in gre skozi vsa vozlišča iz množice S (S ⊂ V), tako da je vsako vozlišče obiskano samo enkrat in da se pot zaključi v vozlišču 1. Dolžino te najkrajše poti označimo z:

$g(i, \mathrm{S})$,

pri čemer velja, da je *i* vozlišče v katerem se najkrajša pot začne, S ⊂ V pa je množica vseh vozlišč, vključenih v to pot. Ta množica je lahko sestavljena iz katerega koli od vozlišč iz množice V, le vozlišča 1 ne sme vsebovati.
Ko je množica S enaka V - {1} to pomeni, da naša pot poteka že skozi vsa vozlišča in je potrebna samo še ena povezava, da krožno pot sklenemo. Dolžina najkrajše poti je tedaj:

$$g(1, \mathrm{V}-\{1\}) = \min_{2 \le k \le n} \left\{ \mathrm{C}[1,k] + g(k, \mathrm{V}-\{1,k\}) \right\} \quad .$$

Če enačbo posplošimo za primer, ko še nismo obiskali vseh vozlišč, dobimo Bellmanovo enačbo:

$$g(i, \mathrm{S}) = \min_{j \in \mathrm{S}} \left\{ \mathrm{C}[i,j] + g(j, \mathrm{S}-\{j\}) \right\} \quad .$$

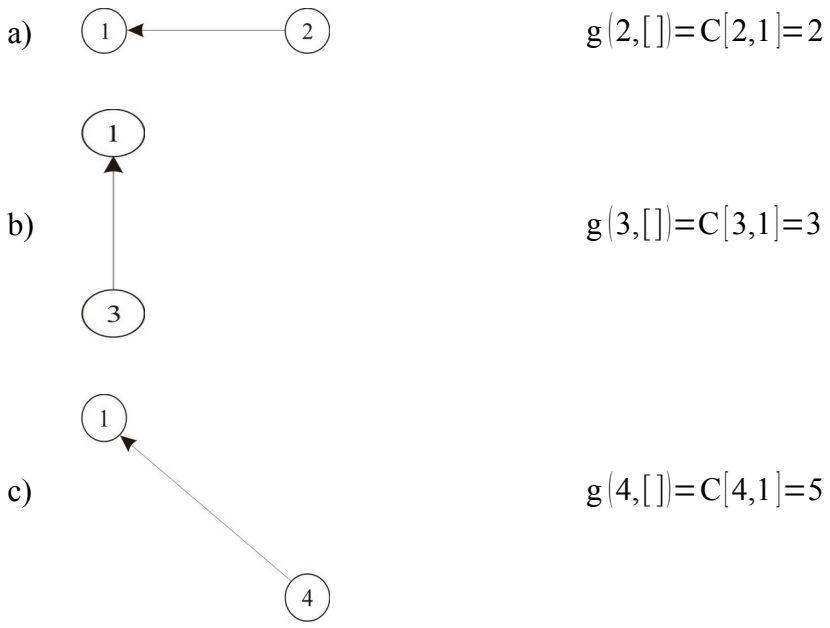
Bellmanova enačba je osnova strategije dinamičnega programiranja. Njeno uporabo bomo prikazali na primeru, ki ga bomo podrobneje obdelali v naslednjem poglavju.

2. Pomoč pri implementaciji

Za pomoč pri implementaciji rešimo primer s slike 1. Za začetno vozlišče bomo izbrali vozlišče 1. Bellmanovo enačbo bomo pri tem uporabljali postopoma, tako, da bomo najprej poiskali vse poti, ki direktno vodijo v vozlišče 1 in imajo eno samo povezavo, nato pa bomo dodajali vedno nove povezave, dokler krožna pot ne bo popolna. Pri tem bomo z Bellmanovo enačbo ugotavljali katero pot se splača razvijati naprej in katera ne bo vodila do rešitve.

1. Iskanje poti z eno vključeno povezavo

Začnimo tako s potjo, ki vsebuje eno samo povezavo in se konča v vozlišču 1. Glede na graf s slike 1, imamo tri take poti, prikazane na sliki 2.



Slika 2: Pot v vozlišče 1 z eno samo povezavo

Kot lahko vidimo s slike 1, imamo pri poteh z eno samo povezavo eno samo možnost, kako priti iz vozlišča $k \in V - \{1\}$ v vozlišče 1, kar pomeni, da ne moremo izbirati med alternativami, zaradi česar se Bellmanova enačba ustrezno poenostavi.

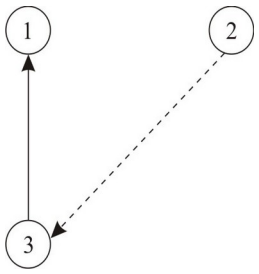
2. Iskanje poti z dvema vključenima povezavama

Z iskanjem poti nadaljujemo tako, da v pot vključimo še eno vozlišče. Naša pot bo tako vključevala tri vozlišča in dve povezavi.

a) Iskanje začnimo z vozliščem 2. Iz vozlišča 2 lahko pridemo v vozlišče 1 ob pogoju, da bo v poti vključeno eno dodatno vozlišče. To dosežemo na dva

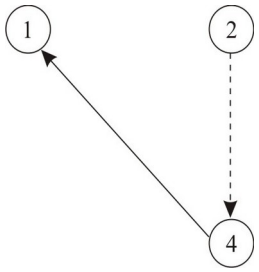
načina in sicer preko vozlišča 3 ali pa vozlišča 4. Obe poti lahko vidimo na sliki 3.

a)



$$g(2,[3])=C[2,3]+g(3,[\,])=6+3=9$$

b)



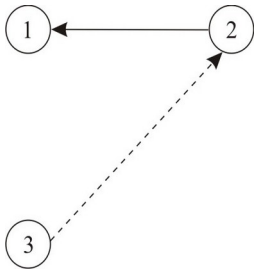
$$g(2,[4])=C[2,4]+g(4,[\,])=7+5=12$$

Slika 3: Pot v vozlišče 1 iz vozlišča 2 preko enega vmesnega vozlišča

Vidimo lahko, da imamo tudi v tem primeru eno samo možnost, kako pridemo iz vozlišča 2 v vozlišče 1 preko vozlišča 3 oziroma 4, kar se prav tako kaže v Bellmanovi enačbi.

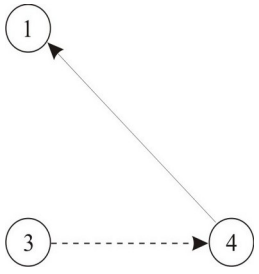
b) Naslednje vozlišče je vozlišče 3. Iz vozlišča 3 lahko pridemo v vozlišče 1 preko enega vmesnega vozlišča na dva načina, preko vozlišč 2 ali 4 (glej sliko 4).

a)



$$g(3,[2])=C[3,2]+g(2,[\,])=10+2=12$$

b)

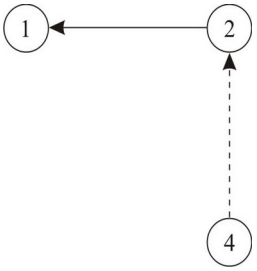


$$g(3,[4])=C[3,4]+g(4,[\,])=9+5=14$$

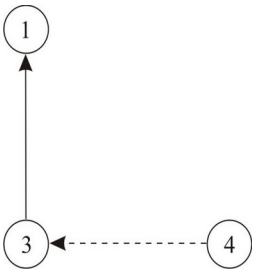
Slika 4: Pot v vozlišče 1 iz vozlišča 3 preko enega vmesnega vozlišča

Enako kot pri vozlišču 2 lahko tudi tukaj pridemo vozlišče 1 preko vozlišča 2 oziroma 4 na samo en način, tako da nimamo nobene alternative.

c) Zadnje nam je ostalo vozlišče 4. Iz vozlišča 4 lahko pridemo v vozlišče 1 ob pogoju, da bo v poti vključeno eno dodatno vozlišče preko vozlišča 2 ali pa vozlišča 3 (glej sliko 5).



a) $g(4,[2])=C[4,2]+g(2,[\,])=7+2=9$



b) $g(4,[3])=C[4,3]+g(3,[\,])=5+3=8$

Slika 5: Pot v vozlišče 1 iz vozlišča 4 preko enega vmesnega vozlišča

S pregledom poti, ki se začnejo v četrtem vozlišču smo končali z iskanjem poti z dvema povezavama. Kot smo lahko videli v zgornjih treh primerih, si pri računanju Bellmanove enačbe poti z dvema povezavama pomagamo z Bellmanovo enačbo poti z eno samo povezavo. Že izračunane vrednosti so bile na slikah ponazorjene s polno črto v povezavi. Enako oznako bomo uporabljali tudi v nadaljevanju.

3. Iskanje poti s tremi vključenimi povezavami

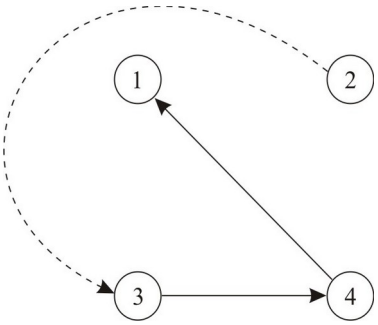
Naš naslednji korak je, da v krožno pot vključimo še eno povezavo. To pomeni, da bomo iz vozlišča *k* šli v vozlišče 1 preko dveh vmesnih vozlišč.

a) Če tudi tokrat začnemo v vozlišču 2, lahko glede na graf s slike 1, v vozlišče 1 pridemo preko vozlišč 3 in 4 pri čemer imamo dve možnosti. Prva možnost je, da gremo iz vozlišča 2 najprej v vozlišče 3, nato pa v 4 in odtod do 1, druga možnost pa je, da gremo iz vozlišča 2 najprej v vozlišče 4 in nato v vozlišči 3 in 1. Opraviti imamo torej s potema:

$2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ in
$2 \rightarrow 4 \rightarrow 3 \rightarrow 1$.

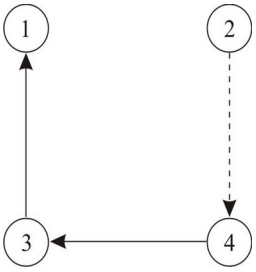
Grafa obeh poti sta skupaj z Bellmanovima enačbama prikazana na sliki 6. Za razliko od prejšnjih primerov, sta sedaj v obeh primerih v množici S enaki vozlišči, to pa pomeni, da sedaj lahko izbiramo med dvema alternativama in izberemo tisto pot, ki je cenejša, dražjo pa zavržemo, glej sliko 6a.

a)



$$g(2,[3,4]) = C[2,3] + g(3,[4]) = 6 + 14 = 20$$

b)



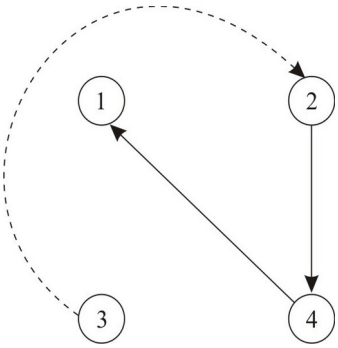
$$g(2,[3,4]) = C[2,4] + g(4,[3]) = 7 + 8 = 15$$

Slika 6: Pot v vozlišče 1 iz vozlišča 2 preko dveh vmesnih vozlišč

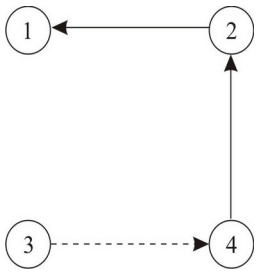
Kot cenejša se je izkazala pot 6b s ceno 15, ki jo uporabimo vedno, ko pridemo v situacijo, da gremo iz vozlišča 2 v vozlišče 1 preko dveh vmesnih vozlišč.

b) Z iskanjem nadaljujemo v vozlišču 3, iz katerega poiščemo pot v vozlišče 1 preko dveh vmesnih vozlišč. Podobno kot pri vozlišču 2, imamo tudi tukaj dve možnosti, ki jih vidimo na sliki 7.

a)



$$g(3,[2,4]) = C[3,2] + g(2,[4]) = 10 + 12 = 22$$

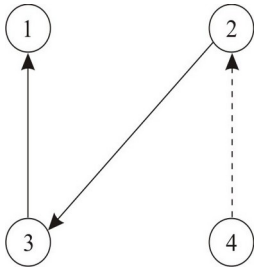
b) 

$$g(3,[2,4])=C[3,4]+g(4,[2])=9+9=18$$

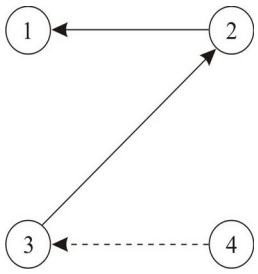
Slika 7: Pot v vozlišče 1 iz vozlišča 3 preko dveh vmesnih vozlišč

Tudi v tem primeru se kot cenejša izkaže pot na sliki 7b, ki si jo zapomnimo, medtem ko alternativo zavržemo, saj ne more voditi k najcenejši krožni poti. Vedno, ko bomo prišli v situacijo, da je potrebno iti z vozlišča 3 do vozlišča 1 preko dveh vmesnih vozlišč bomo uporabili pot s slike 7b.

c) Kot zadnje nam je ostalo vozlišče 4. Naša naloga je enaka kot pri predhodnih dveh vozliščih, to je najti pot do vozlišča 1, ki se prične v vozlišču 4 in vsebuje dve vključeni vozlišči, ki sta v našem primeru vozlišči 2 in 3. Tudi v tem primeru imamo dve možnosti, ki sta prikazani na sliki 8 skupaj s pripadajočima Bellmanovima enačbama.

a) 

$$g(4,[2,3])=C[4,2]+g(2,[3])=7+9=16$$

b) 

$$g(4,[2,3])=C[4,3]+g(3,[2])=5+12=17$$

Slika 8: Pot v vozlišče 1 iz vozlišča 4 preko dveh vmesnih vozlišč

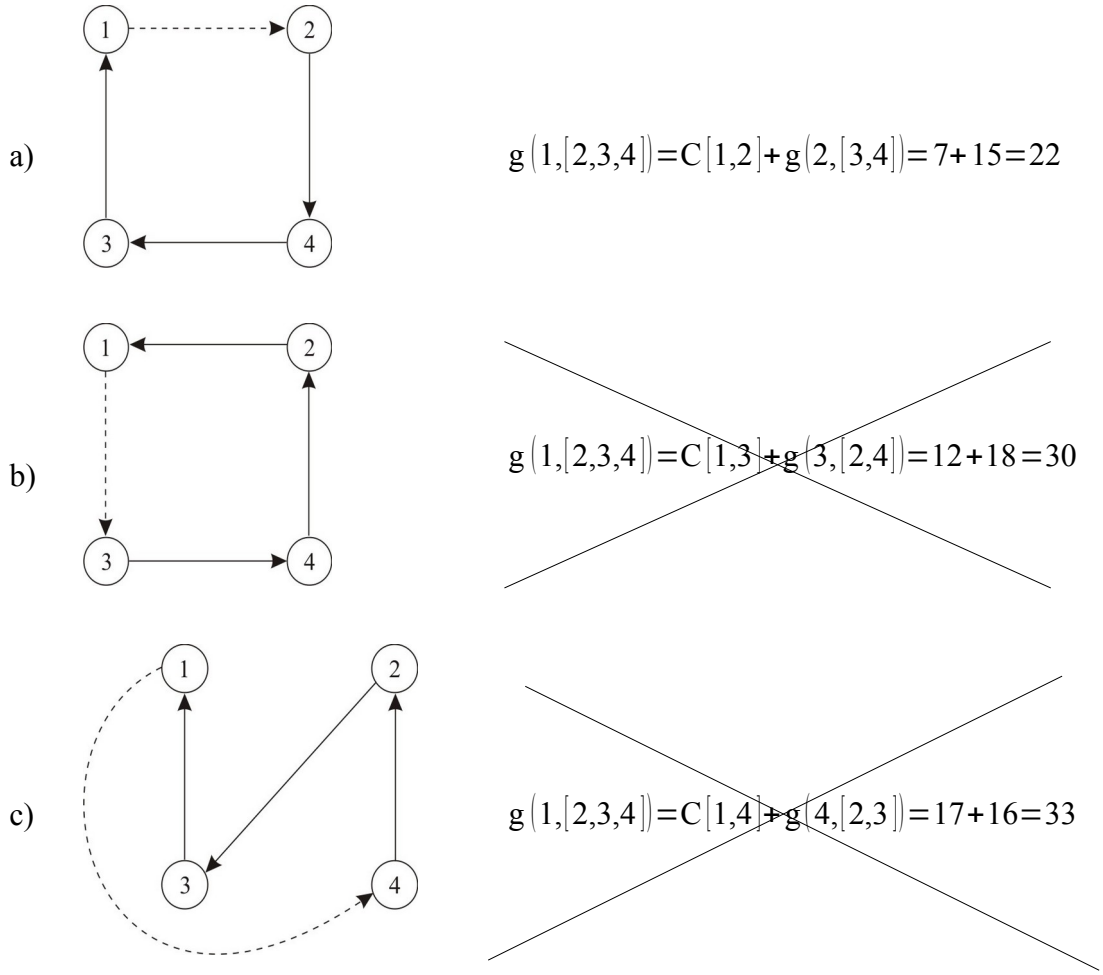
Iz Bellmanovih enačb s slike 8 vidimo, da je pot na sliki 8a cenejša, zato jo ohranimo, medtem, ko pot 8a zavržemo.

4. Zaključevanje krožne poti

S potmi, ki vključujejo tri povezave smo prišli do točke, kjer nam manjka samo še ena povezava, da krožno pot zaključimo. Iščemo torej še zadnjo povezavo, $1 \rightarrow k$ kjer vozlišče k pripada množici V-{1}. Iz vozlišča 1 lahko gremo v tri različna vozlišča 2, 3 in 4. Ko enkrat pridemo v katerokoli od teh vozlišč, je naša nadaljnja pot določena s predhodno izračunanimi Bellmanovimi enačbami. Tako imamo opraviti samo s tremi krožnimi potmi in sicer:

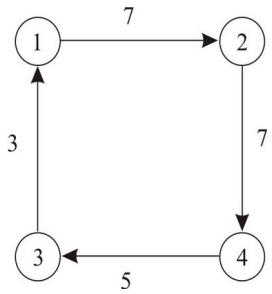
$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1,$
$1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ in
$1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1.$

Njihovo ceno spet izračunamo s pomočjo Bellmanove enačbe. Enačbe, skupaj s tremi krožnimi potmi, so prikazane na sliki 9. Krožna pot, ki bo najcenejša je tudi rešitev našega problema.



Slika 9: Zaključena krožna pot, ki se začne in konča v vozlišču 1

Iz slike 9 lahko vidimo, da je najcenejša krožna pot na sliki 9a. Bellmanova enačba pove, da je cena te krožne poti enaka 22, kar lahko preverimo na sliki 10.



Slika 10: Najcenejša krožna pot za graf na sliki 1

Iz celotnega postopka lahko vidimo, da smo z uporabo Bellmanove enačbe zmanjšali število možnih krožnih poti s šestih na samo tri, s tem pa močno zmanjšali časovno zahtevnost algoritma, ob tem da smo pregledali vse možne kombinacije, kar pomeni da je ta način reševanja eksaktna metoda.

Iz samega postopka vidimo, da poteka iskanje krožne poti postopoma. Najprej zgradimo vse poti z eno samo povezavo, ki se končajo v vozlišču 1, nato pa postopoma dodajamo povezavo za povezavo v našo pot, dokler na koncu z zadnjo povezavo poti ne zaključimo v izhodiščnem vozlišču.

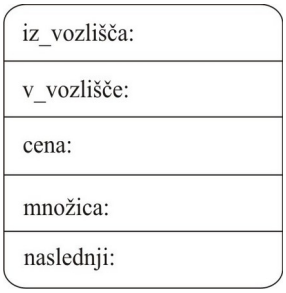
Na vsakem nivoju, če jih lahko tako imenujemo, potrebujemo izračune Bellmanove enačbe predhodnega nivoja. Na ta način se posamezni izračuni po nepotrebnem ne ponavljajo, kar je bistveno za pohitritev celotnega postopka iskanja.
Algoritem za reševanja problema trgovskega potnika s pomočjo dinamičnega programiranja je prikazan v izpisu 1.

```
procedure POTNIK(C, N)
 begin
   sklad={} // sklad nivojev
   nivo=VSTAVI_PRVA_VOZLIŠČA(C,N);
   NOV_ELEMENT_SKLADA(sklad, seznam_poti);
   for st_nivoja := 2 to N - 1 do
     begin
       nivo={}
       for vozlišče := 2 to N do
         begin
           NAPRAVI_SEZNAM(C,N, vozlišče, sklad, nivo);
         end;
       NOV_ELEMENT_SKLADA(sklad, nivo);
     end;
   nivo=KONČAJ_SEZNAM(C, N, sklad);
   NOV_ELEMENT_SKLADA(sklad, nivo);
   return sklad;
 end
```

Izpis 1: Psevdokod algoritma POTNIK

Vhodni podatki za algoritem v izpisu 1 so matrika sosednosti *C*, Število vozlišč v grafu *N*, izhodna podatka pa sta *R*, to je vektor rešitve, kamor vpišemo najdeno krožno pot in parameter *d_poti*, kamor vpišemo dolžino najdene poti. V algoritmu kličemo tudi več funkcij. Funkcije *VSTAVI_PRVA_VOZLIŠČA*, *NAPRAVI_SEZNAM* in *KONČAJ_SEZNAM* so v bistvu klici Bellmanove enačbe. Funkcija *VSTAVI_PRVA_VOZLIŠČA* je klic enačb nivoja 1, funkcija *NAPRAVI_SEZNAM* pa je klic Bellmanovih enačb v točkah 2 in 3. Funkcija *KONČAJ_SEZNAM* pa krožno pot zaključi, tako da ji doda zadnjo povezavo $1 \rightarrow k$, se pravi predstavlja 4. nivo našega primera.

Algoritem pa za svoje delovanje potrebuje tudi posebno podatkovno strukturo, ki bo v našem primeru sklad seznamov. V posameznem seznamu hranimo posamezen nivo, to so poti z določenim številom sprejetih povezav, nivoji pa se bodo shranjevali na sklad. Posamezni element seznama bo tako optimalna pot iz danega vozlišča v vozlišče 1, katere ceno dobimo s pomočjo Bellmanove enačbe. Struktura elementa seznama je prikazana na sliki 11.



Slika 11: Element seznama v katerem hranimo posamezno pot do izhodiščnega vozlišča

Kot je razvidno iz slike 11, je pot opisana z zadnjo povezavo v poti, njeno ceno in množico vključenih vmesnih vozlišč do vozlišča 1. Kot rečeno v seznamu hranimo samo najugodnejše poti, kar določimo z Bellmanovo enačbo. Element na sliki 11 lahko v C-ju opišemo z naslednjo strukturo:

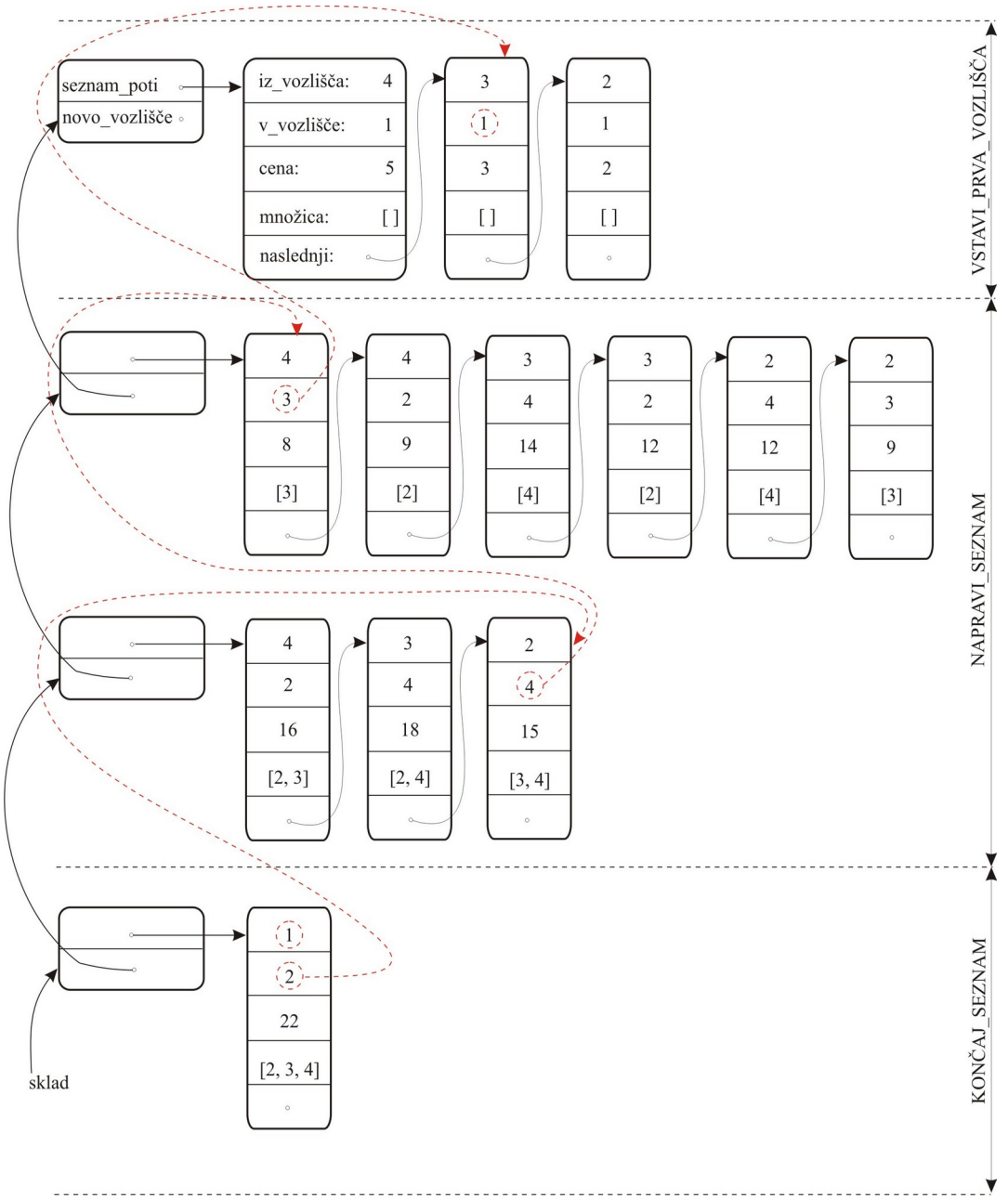
```
struct pot {
      int iz_vozlisca;
      int v_vozlisce;
      int cena;
      bool mnozica[ST_VOZLISC];
      pot *naslednji;
};
```

Zaradi hitrejšega iskanja bomo množico S predstavili kar s poljem boolovih vrednosti, kjer vrednost *ST_VOZLISC*, predstavlja število vozlišč.

Povedali smo že, da posamezne nivoje shranjujemo na sklad, ki pozneje služi tudi za rekonstrukcijo rešitve. Sklad predstavimo v C-ju z naslednjo strukturo:

```
struct nivo {
      pot  *seznam_poti;
      nivo *novo_vozlisce;
};
```

Za lažjo predstavo poglejmo opisano podatkovno strukturo, ki bi jo dobili z reševanjem problema trgovskega potnika z dinamičnim programiranjem s pomočjo dinamičnega programiranja za naš testni graf s slike 1. Podatkoma struktura je prikazana na sliki 12, kjer so posebej označeni nivoji, ki jih naredijo funkcije *VSTAVI_PRVA_VOZLIŠČA*, *NAPRAVI_SEZNAM* in *KONČAJ_SEZNAM*.



Slika 12: Podatkovna struktura pri reševanju problema TSP za graf s slike 1

Na sliki 12 pa poleg same podatkovne strukture vidimo še nekaj več, z rdečimi puščicami imamo namreč označeno rekonstrukcijo rešitve krožne poti. Naše iskanje začnemo na zadnjem nivoju, z zadnjo sprejeto povezavo. To je v našem primeru povezava 1 → 2. Naša krožna pot se torej začne s to povezavo in vsebuje vozlišča 2, 3 in 4, kar razberemo z množice S. Iz te množice izvzamemo vozlišče 2, ki je vključeno v prvi povezavi in se premaknemo en nivo više, kjer poiščemo pot, ki se začne v vozlišču 2, v njej pa so vključena vozlišča 3 in 4. Po preiskovanju seznama na enem nivoju više vidimo, da zadnji element seznama izpolnjuje te pogoje, našo pot tako razširimo s povezavo 2 → 4. Iz množice S izvzamemo vozlišče 4, tako da v njej

ostane samo vozlišče 3. Z iskanjem nato nadaljujemo na enem nivoju više in sicer iščemo pot, ki se prične v vozlišču 4 in vsebuje vozlišče 3. Iz slike 12 vidimo, da je to takoj prvi element seznama, naši poti pa pridružimo vozlišče 3. Naša pot je tako 1 → 2 → 4 → 3. Iz množice S odstranimo vozlišče 3, tako da postane množica S prazna, mi pa se pomaknemo en nivo više in poiščemo pot, ki se prične v vozlišču 3. Iz slike 12 vidimo, da je na tem zadnjem nivoju ena sama taka pot, ki je shranjena v drugem elementu seznama. Našo krožno pot tako zaključimo z zadnjo povezavo, to je povezavo 3 → 1. Naša rekonstruirana krožna pot se tako glasi:

$$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1.$$

Celoten algoritem za rekonstrukcijo poti je prikazan v izpisu 2.

```
procedure REKONSTRUCKIJA_POTI(sklad)
 begin
   trenutni_nivo=POP(sklad)
   pot_od=PRVA_POT(trenutni_nivo)
   print pot_od.iz_vozlisca

   while not SKLAD_PRAZEN(sklad)
     trenutni_nivo=POP(sklad)
     pot_do=NAJDI_PRAVO_POT(pot_od, trenutni_nivo)
       // NAJDI_PRAVO_POT naj vrne pot_do iz trenutni_nivo, kjer je:
       //   pot_od.v_vozlisce=pot_do.iz_vozlisca in
       //   pot_od.S - pot_od.v_vozlisce = pot_do.S
       //

     print pot_do.iz_vozlisca

     pot_od=pot_do
   end
 end
```

Izpis 2: Psevdokod algoritma POTNIK

Ceno krožne poti, ki je v našem primeru 22, preprosto preberemo iz končnega nivoja sklada, kjer z rekonstrukcijo poti tudi začnemo.