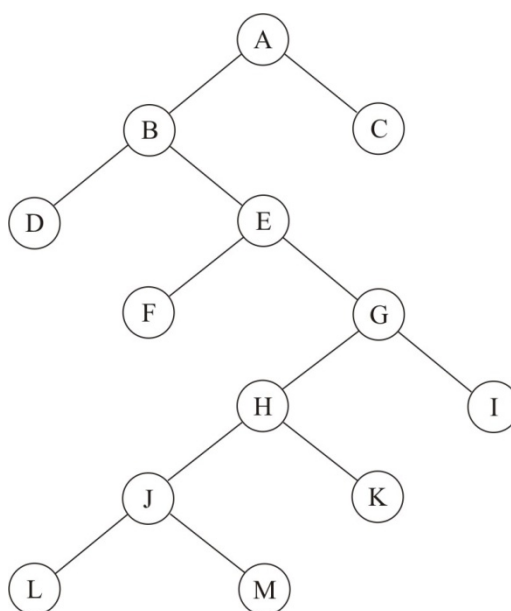


Iskanje v globino

1. Splošna predstavitev problema

Pri pripravi na potovanja se soočamo s problemom, kako priti iz izhodiščnega mesta A v ciljno mesto B. Pri reševanju tega problema si pomagamo z zemljevidom, na katerem poiščemo lokacijo obeh mest in cestne povezave med njima. Ker ceste med seboj povezujejo mesta, je naša pot določena s seznamom krajev, ki ležijo med našim izhodiščnim mestom A in ciljnim mestom B. Ta postopek je časovno precej zahteven in v kolikor moramo to aktivnost velikokrat ponavljati, postane potreba po avtomatskem iskanju toliko bolj izrazita. Ker je problem iskanja poti med posameznimi mesti zelo dolgo prisoten, so se tudi tukaj že zgodaj pojavile metode, s katerimi ta problem uspešno rešujemo. Tudi v tem primeru nam rešitev ponuja teorija grafov, ki nam mesta pretvori v vozlišča V , poti med njimi pa v povezave E . Grafe zapišemo kot dvojico vozlišč in povezav $G=(V, E)$. Ker pa se problemi iskanja pojavljajo tudi v računalniških igrah, kjer recimo pomeni A trenutno stanje, B pa stanje po recimo petih potezah uporabnika, ne preseneča, da so algoritmi iskanja eni temeljnih algoritmov v računalništvu. V našem primeru si bomo pogledali najenostavnejše med iskanji, to je iskanje v globino. Denimo, da imamo graf G na sliki 1 in nas zanima, kako priti iz vozlišča A do vozlišča M.



Slika 1: Primer izhodiščnega grafa

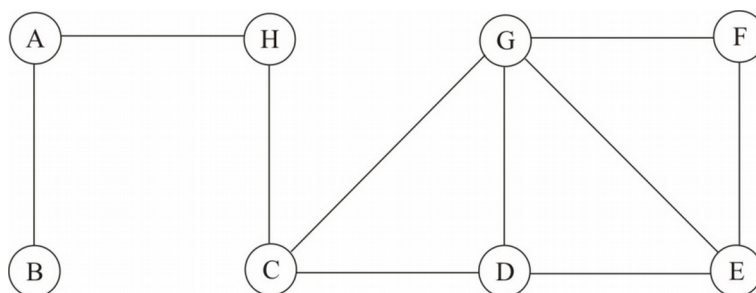
Iz slike 1 je takoj razvidno, da lahko pridemo iz vozlišča A do M po naslednji poti: A–B–E–G–H–J–M. Iskanje je bilo enostavno, saj je graf na sliki 1 dvojiško drevo. V primeru poljubnega grafa je iskanje težje, kljub temu pa to za človeka ni težka naloga, saj rešujemo tovrstne probleme vizualno. Računalniki te možnosti nimajo, zato se poslužujejo algoritmičnega reševanja problemov.

Pri iskanju v globino bomo za vsako vozlišče v grafu uvedli tri stanja, ki jih bomo opisali s številsko vrednostjo od 0 do 2 in sicer: *vozlišče je nepregledano* (0), *vozlišče je v stanju pregledovanja* (1) in *vozlišče je razvito* (2). Pojem vozlišče je razvito, se

pojavlja tudi v drugih algoritmi in v splošnem pomeni to, da je vozlišče obdelano ter da poznamo vse njegove sinove.

Poleg uvedbe stanj pa bo potrebno voditi še sklad za hranjenje vozlišč za pregled, ki ga označimo s črko S . Sklad je podatkovna struktura, pri kateri iz seznama ali polja jemljemo podatke na istem mestu, kot jih v njega vstavljamo, to je pri vrhu. Če v sklad vstavimo vrednosti 1, 2, in 3, pri branju dobimo vrednosti: 3, 2, 1.

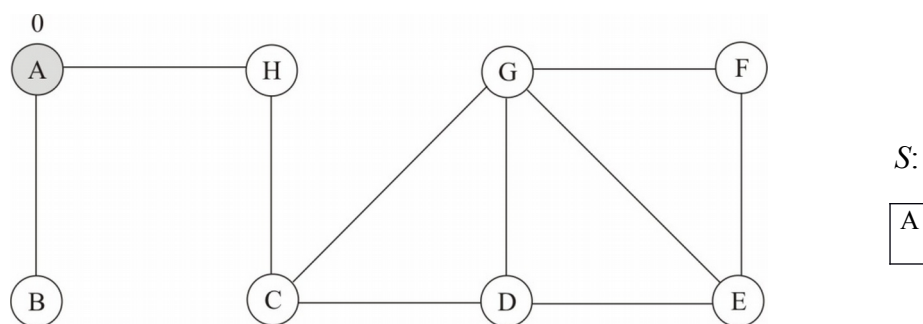
Opremljeni s stanji in skladom vozlišč za razvijanje, S , pogledjmo nekoliko kompleksnejši primer, kot je tisti na sliki 1.



Slika 2: Kompleksnejši izhodiščni graf

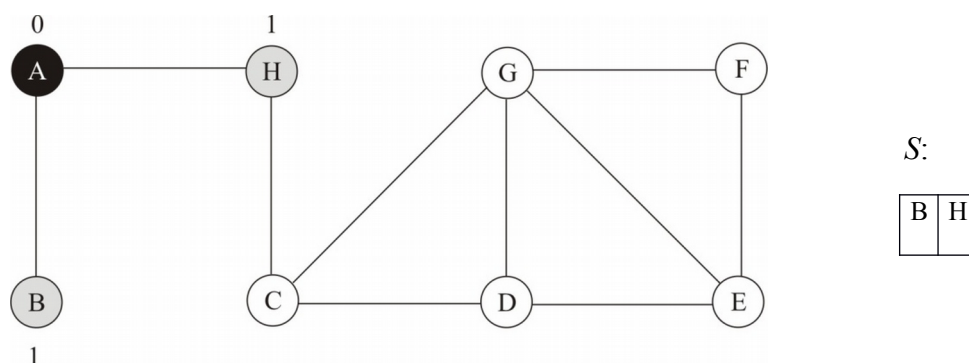
Na sliki 2 je prikazan nov izhodiščni graf G' . Izhodiščno vozlišče je tudi v tem primeru vozlišče A, ciljno vozlišče pa je vozlišče E. Oddaljenost nekega vozlišča od vozlišča A bomo merili s številom povezav v poti med njima. Dolžina poti do izhodiščnega vozlišča bo eden od parametrov, ki jih bomo hranili za vsako vozlišče, prav tako pa bomo hranili ime starša, to je vozlišča iz, katerega smo v dano vozlišče prišli.

Z iskanjem poti pričnemo v vozlišču A, vozlišče, ki ga iščemo pa je vozlišče E. Ker gre za izhodiščno vozlišče, je dolžina poti enaka 0, oče pa ne obstaja. A nato damo v sklad vozlišč za razvijanje, S . Dobimo situacijo, ki jo prikazuje slika 3.



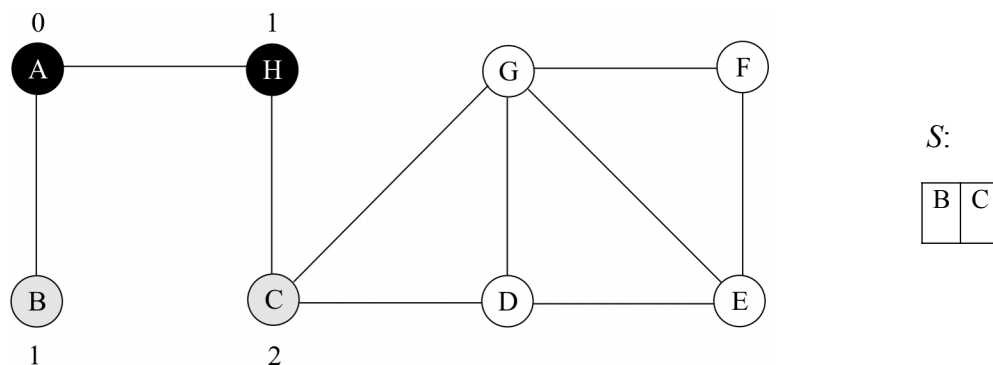
Slika 3: Začetek iskanja poti

Sedaj vzamemo vozlišče A iz sklada S in začnemo njegov razvoj, tako da poiščemo njegove potomce in jih vpišemo v sklad S . Vozlišče A ima dva potomca, B in H. Preden jih dodamo v sklad, je potrebno izračunati oddaljenost od izhodiščnega vozlišča. Le – to dobimo tako, da oddaljenosti njihovega starša prištejemo vrednost 1. Tako je oddaljenost vozlišč B in H od vozlišča A enaka 1 ($0+1$), vozlišče A pa vpišemo kot starša v obe vozlišči. Po izračunu razdalj, vozlišči B in H vstavimo v sklad S , vozlišče A pa označimo kot razvito, saj smo pregledali vse njegove potomce. To, da je vozlišče razvito, bomo označili s črno barvo vozlišča, v programu pa status vozlišča označimo z ustrezno konstanto. Situacijo po razvoju vozlišča A prikazuje slika 4.



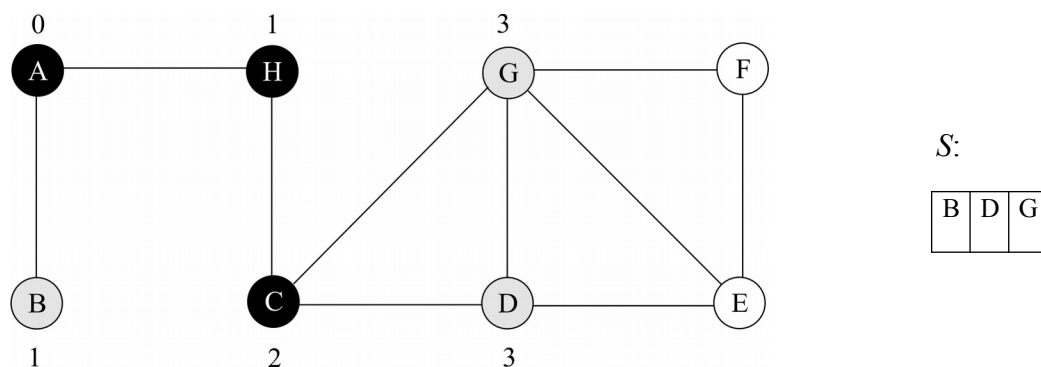
Slika 4: Stanje v grafu po razvoju vozlišča A

Ker še nismo izčrpali vseh vozlišč grafa v skladu ali pa našli iskanega vozlišča, z iskanjem nadaljujemo in sicer vzamemo iz sklada vozlišče H. Iz grafa G' razberemo, da ima vozlišče H enega samega potomca, to je vozlišče C. Preden ga vstavimo v sklad S , je potrebno izračunati njegovo oddaljenost od vozlišča A. Spet je ta razdalja določena z razdaljo predhodnika, to je vozlišča H, ki jo povečamo za 1. Razdalja med C in A tako znaša 2 ($1 + 1$). V vozlišče C shranimo tudi informacijo o njegovem staršu, to je vozlišču, ki ga trenutno razvijamo, v našem primeru torej H. Po tem izračunu tudi vozlišče H označimo kot razvito, vozlišče C pa vstavimo v seznam S . Nastala situacija je prikazana na sliki 5.



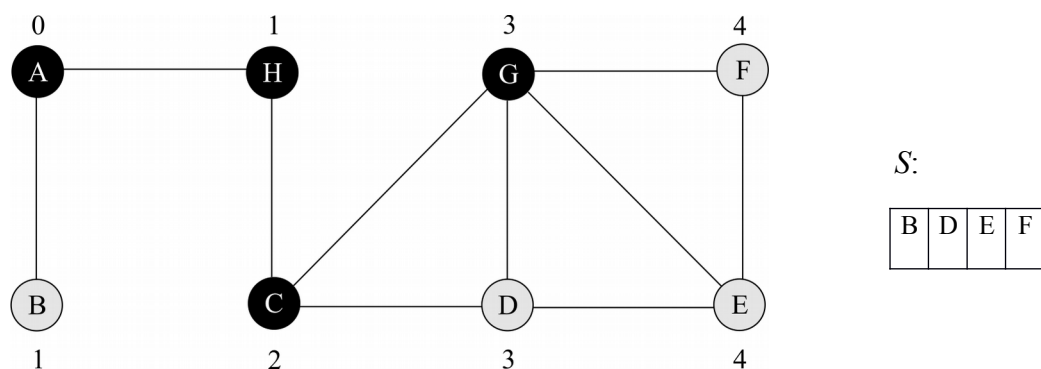
Slika 5: Stanje po razvoju vozlišča H

Ker sklad S ni prazen, nadaljujemo z iskanjem. Tako iz sklada S ponovno vzamemo vozlišče C in poiščemo njegove potomce. To sta vozlišči G in D. Za obe vozlišči izračunamo oddaljenost od vozlišča A ($2+1$) in vozlišče C vpišemo kot očeta pri obeh vozliščih. Po opravljenem vozlišče C označimo kot pregledano, vozlišči G in D pa vstavimo v sklad S . Dobimo situacijo, ki jo prikazuje slika 6.



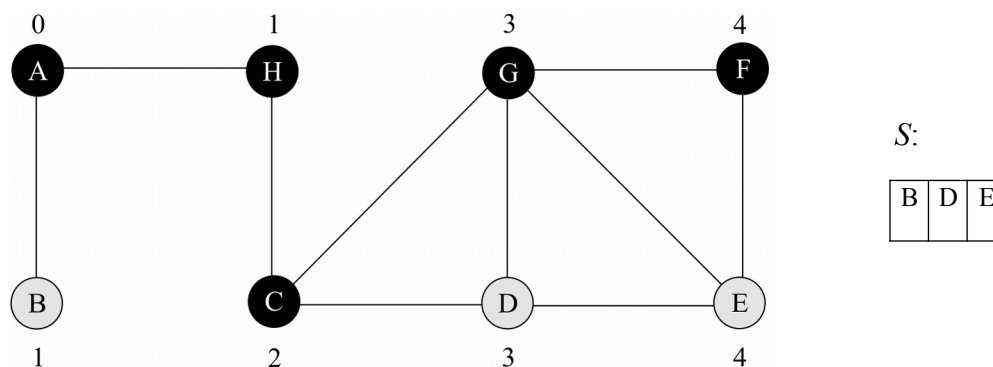
Slika 6: stanje po razvoju vozlišča C

Ker še vedno nismo pregledali vseh vozlišč v grafu, oziroma nismo našli vozlišča, ki ga iščemo, z iskanjem nadaljujemo in tako vzamemo iz sklada S vozlišče G. Vozlišče G ima štiri sosedbe: C, D, E in F. Pri tem je vozlišče C že razvito, vozlišče D čaka na pregled, nepregledani pa sta le vozlišči E in F, ki sta tako potomca vozlišča G. Preden vstavimo vozlišči E in F v sklad S , ju je potrebno opremiti s podatkom o očetu, se pravi, da v vsakega od vozlišč vpišemo, da je njegov oče vozlišče G. Prav tako je potrebno izračunati oddaljenost vozlišč od izhodišnega vozlišča A, za kar uporabimo oddaljenost njunega očeta, ki jo povečamo za ena. Tako dobimo, da je razdalja vozlišč E in F do A enaka 4. V kolikor bi želeli iskanje skrajšati, bi bilo potrebno preverjati, ali je kateri od sinov že iskano vozlišče in nato razviti tega in s tem končati iskanje, vendar se v tem primeru število preverjanj precej poveča. Zato obe vozlišči brez preverjanja vstavimo v sklad in ju razvijemo enako kot vsa ostala vozlišča. Nastalo situacijo prikazuje slika 7.



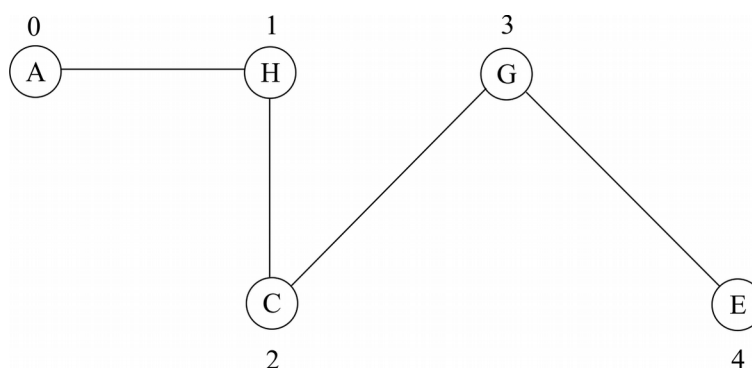
Slika 7: nastala situacija po razvoju vozlišča D

Naslednje vozlišče, ki ga vzamemo iz sklada S je vozlišče F, ki nima več nepregledanih sosedov, zato ga označimo kot razvitega. Dobljeno situacijo prikazuje slika 8.



Slika 8: Nastala situacija, po razvoju vozlišča G

Ker sklad še ni prazen, vzamemo ven naslednje vozlišče, to pa je vozlišče E. Vozlišče E je tisto vozlišče, ki ga iščemo, zato z iskanjem prenehamo, nepregledana vozlišča pa zavržemo. Iskana pot je tako A–H–C–G–E, njena dolžina pa je štiri. To pot nam kaže slika 9.



Slika 9: Najdena pot med vozliščema A in E

Ker smo v obravnavanem primeru našli rešitev, smo lahko z iskanjem končali, preden smo pregledali celoten graf in smo nepregledana vozlišča zavrgli. V kolikor do iskanega vozlišča pot ne bi obstajala, bi z iskanjem končali šele, ko bi bil sklad vozlišč prazen. Pri implementaciji algoritma bomo v vsakem primeru z iskanjem končali, ko bo sklad razen. Na takšen način bomo tvorili iskalno drevo – seznam vozlišč z izračunanimi predhodniki in dolžinami poti, iz katerega bomo lahko razbrali pot iz začetnega vozlišča do vseh ostalih.

3. Pomoč pri implementaciji

Iz predhodno opisanega sledi, da je v vozlišču potrebno hraniti štiri podatke: njegovo lokacijo oziroma indeks znotraj seznama vseh vozlišč (V), dolžino poti od izhodišnega vozlišča, njegovega starša, kot indeks znotraj seznama vseh vozlišč, in status vozlišča. Dodatno se lahko odločimo in dodamo tudi ime vozlišča. V tem primeru bi imela podatkovna struktura v programskem jeziku C++ naslednjo obliko:

```
struct Vozlisce {  
    int predhodnik;  
    int dolzina;  
    int status;  
    int indeks;  
    string ime;  
};
```

Vozlišča bomo hranili v polju (V) in sicer tako, da je vozlišče A prvo, B drugo itd. Vozlišča, ki bodo čakala na razvoj bomo hranili tudi v skladu S , ki bo prav tako implementiran s poljem primerne dolžine. Pseudokod algoritma za iskanje v globino je prikazan v izpisu 1.

```
function ISKANJE_V_GLOBINO( $G, s$ )  
  begin  
    for each  $v \in V - \{s\}$  do  
      begin  
         $v.status := NEPREGLEDANO$ ;  
         $v.dolzina := \infty$ ;  
         $v.predhodnik := -1$ ;  
      end  
       $s.status := V\_OBDELAVI$ ;  
       $s.dolzina := 0$ ;  
       $s.predhodnik := -1$ ;  
       $PUSH(s)$ ; // vstavi na vrh sklada  
  
      while not SKLAD_PRAZEN() do  
        begin  
           $v = POP()$ ; //vzemi vozlišče iz vrha sklada  
  
          for each vozlišče  $u \in \text{sosedi}\{v\}$  do  
            if  $u.status = NEPREGLEDANO$  then  
              begin  
                 $u.status := V\_OBDELAVI$ ;  
                 $u.dolzina := v.dolzina + 1$ ;  
                 $u.predhodnik := v.indeks$ ;  
                 $PUSH(u)$ ; // vstavi na vrh sklada  
              end  
             $v.status := RAZVITO$ ;  
          end  
        end  
      end
```

Izpis 1: Pseudokod algoritma ISKANJE_V_GLOBINO

V izpisu 1 vidimo, da smo vrednosti dolžin poti do izbranega izhodišča s inicializirali na vrednost ∞ , kar pomeni neko zelo veliko vrednost. Vhodni parametri v proceduro ISKANJE_V_GLOBINO so podatki o grafu, G , ter začetno vozlišče, s , in iskano končno vozlišče, d . Povedati pa je treba tudi na kak način določimo sosede nekega vozlišča. Najenostavnejša metoda je uporaba matrike sosednosti. Recimo, da iščemo

sosede vozlišča A. Vozlišču A pripada prva vrstica v matriki sosednosti. Sosedni vozlišča A so tista vozlišča, kjer so stolpci različni od 0. Grafu G' s slike 2 pripada naslednja matrika sosednosti:

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Iz matrike C torej vidimo, da imamo v drugem in osmem stolpcu prve vrstice neničelne vrednosti, torej sta soseda vozlišča A vozlišči B in H. Matriko sosednosti C tvorimo iz seznama povezav tako, da na križišču vrstice p in stolpca q ter vrstice q in stolpca p vpišemo vrednost 1, pri tem pa so začetne vrednosti v matriki C enake 0. Matriko sosednosti lahko implementiramo z navadnimi dvodimenzionalnim poljem. Dodatno pa lahko nalogo poenostavimo tako, da namesto črk za imena vozlišč uporabljamo kar številke, tako da velja, da ime vozlišča pomeni tudi indeks, s katerim do vozlišča dostopamo. Iskana pot, če le-ta obstaja, je shranjena v razvitih voziščih, zato jo je od tam najprej potrebno rekonstruirati. To opravimo z naslednjo proceduro, prikazano v izpisu 2.

```
function IZPIS_POTI(G, s, v)
begin
  if v = s then
    begin
      izpiši "Pot je: " + v.ime + " "
    end
  else
    begin
      if v nima predhodnika then
        izpiši "Med " + s.ime + " in " + v.ime + " ni poti!"
      else
        begin
          IZPIS_POTI(G, s, V[v.predhodnik])
          izpiši v.ime + " "
        end
      end
    end
  end
end
```

Izpis 2: Pseudokod algoritma IZPIS_POTI

Pri tem je s izhodiščno vozlišče, v je vozlišče v katerem se trenutno nahajamo, c pa je cena poti, ki jo preberemo iz ciljnega vozlišča.

4. Zahteve naloge

Implementirati je potrebno algoritem iskanja v globino, ki bo omogočal iskanje najkrajše poti med izhodiščnim vozliščem s in ciljnim vozliščem d , če le-ta obstaja. Ob zagonu se naj izpiše meni, prikazan na sliki 10.

```
Iskanje v globino – izbira:
1) Preberi graf iz datoteke
2) Poženi iskanje iz vozlišča s
3) Izpis seznama vozlišč in njihovih podatkov
4) Izpis poti med vozliščema s in d
5) Konec
Vaša izbira:
```

Slika 10: Meni aplikacije za iskanje v globino

Ob izbiri menijske postavke *Preberi graf iz datoteke* poženite proceduro, za branje podatkov iz datoteke, ki naj prebere datoteko z imenom *graf.txt* ter tvori matriko sosednosti. Vhodni graf bo tekstovna datoteka, v kateri bomo imeli na začetku podano število vozlišč v grafu (do vključno 2000), nato pa bo sledil seznam vseh povezav v grafu, predstavljen s trojico podatkov, imenoma krajiščnih vozlišč, ter ceno povezave, ki bo v našem primeru vedno enaka 1. Primer takega grafa je objavljen na sistemu za vaje.

Ob izbiri menijske postavke *Poženi iskanje iz vozlišča s*, uporabnik vnese izhodiščno vozlišče nato pa se zažene algoritem za iskanje v globino, ki tvori iskalno drevo – seznam vozlišč z izračunanimi predhodniki in dolžinami poti. Po tvorbi iskalnega drevesa ob izbiri menijske postavke *Izpis seznama vozlišč* izpišite vsa vozlišča iz seznama vozlišč. Pri vsakem vozlišču izpišite vse podatke iz strukture *Vozlišče*. Ob izbiri menijske postavke *Izpiši poti med vozliščema s in d* kličite funkcijo *IZPIS_POTI*, ki vrne pot med izhodiščnim in ciljnim vozliščem. V kolikor iskalno drevo še ni generirano, uporabnika o tem obvestite, pot pa se ne izpiše. Enako se uporabnika obvesti, če se izhodiščno vozlišče v iskani poti razlikuje od tistega, uporabljenega pri generiranju iskalnega drevesa. Program naj omogoča izpis poti do različnih ciljnih vozlišč. Program se konča, ko uporabnik izbere menijsko postavko *Konec*. V nalogi lahko črke v imenu vozlišč zamenjate s številkami, kar bo dodatno poenostavilo implementacijo.