Fakultet elektrotehnike i računarstva
Zavod za primijenjeno računarstvo

Napredni algoritmi i strukture podataka

3. laboratorijska vježba

Josip Milić 0036456339

Zagreb, 23.01.2015.

1. Zadatak

Zadaci za 11 (17) bodova

2) Modelirati grafom dio nekog naselja i programski odrediti najkraći put između dva mjesta (dva vrha). Tko želi, može modelirati i nešto drugo, gdje bi bilo čak i bridova negativnih težina. – za kolokviranje vježbe važno je pregledno i jasno opisati model te organizaciju podataka u programu – nije potrebno graditi komplicirane modele, dovoljni su grafovi s 10...20 vrhova. Naravno, kompliciraniji modeli će vjerojatno biti i izazovniji te kao takvi zanimljivija i „plodonosnija“ laboratorijska vježba. – program mora riješiti pohranu grafa u kompjutoru, pronalaženje najkraćeg puta i ispis (iscrtavanje) rješenja – iscrtavanje grafa i najkraćeg puta nije obavezno, nego samo poželjno, ali prikladan ispis najkraćeg puta je obavezan – iscrtavanje se brzo i relativno lako može postići prepuštanjem tog posla slobodnom (open source) programu Graphviz koji možete preuzeti sa stranice http://www.graphviz.org/, gdje su i podrobne upute za njegovo korištenje. Dovoljno je iskoristiti samo njegovu osnovnu funkcionalnost, bez posebnog dotjerivanja rješenja, a ni njegovo pozivanje ne mora biti automatsko, nego je dovoljno programski pripremiti podatke za Graphviz, a pozivati ga možete i „ručno“ iz komandne linije

2. Rješenje zadatka

2.1. Teorijski uvod

2.1.1. Graf

Vrh (*vertex*) **v** je točka grafa. Skup vrhova označava se s V.
Brid (*edge*) **e** je svaki dvočlani podskup skupa V, poveznica dva vrha. Skup bridova označava se s E.
Težina brida (*edge weight*) **w** je neka brojčana vrijednost pridijeljena bridu (dužina, trajanje).

Graf (*graph*) **G** je uređeni par nepraznog konačnog skupa vrhova V i skupa (moguće praznog) bridova E.
G=(V,E)
Graf se može shvatiti kao stablo (šuma; *forest*) u kojem nema hijerarhije među čvorovima.
Putanja (*trail*) je obilazak u kojem su svi bridovi različiti (znači svakim bridom se prolazi samo jednom, ali vrhovi se mogu ponavljati).
Put ili **staza** (*path*) grafa G je putanja u kojoj su svi vrhovi različiti (ne mogu se ponavljati).

2.1.2. Dijkstrin algoritam najkraćeg puta

Cilj algoritma najkraćeg puta je pronaći najkraći put u grafu na temelju težina i povezanosti njegovih bridova [1]. Oni su temeljni algoritmi za brojne primjene (npr. transport, komunikacije, distribucijske energetske mreže, projektiranje integriranih elektroničkih sklopova). Oni osim težina bridova koriste i sljedeće pojmove:
Međuvrh je vrh na putu između neka dva vrha u grafu, dakle ni polazni ni završni.
Labela (*label*) je udaljenost vrha od nekog referentnog vrha (točke).
Odabir algoritma za traženje najkraćeg puta ovisi o težinama bridova u grafu; dvije su osnovne skupine tih algoritama:

- **label-setting** metoda: jednom upisana labela više se ne mijenja - za grafove s težinama bridova ≥0; primjer je Dijkstrin algoritam najkraćeg puta
- **label-correcting** metoda: sve labele mogu se mijenjati sve do završetka cijelog postupka
  - za grafove s bilo kakvim težinama bridova; primjer Bellman-Fordov algoritam najkraćeg puta

Temelj Dijkstrinog algoritma najkraćeg put

Pretpostavimo da već znamo najkraći put od v0 do vn i označimo sve vrhove na tom putu s vi, i=0,1,2,...,n.
Također, označimo udaljenost pojedinog vrha vi od polaznog vrha s d(v0,vi).
Tada je najkraći put (najmanja udaljenost) od v0 do vn sigurno jednak

- dmin(v0,vn) = dmin(v0,vn–1) + dmin(vn–1,vn) .

Općenito, za svaki vrh vi na najkraćem putu vrijedi

- dmin(v0,vn) = dmin(v0,vi) + dmin(vi,vn) . (vidi Cormen...)

Dakle, gledajući unatrag, ako svaki vrh na najkraćem putu "zna" (barem) svojeg neposrednog prethodnika, može se rekonstruirati cijeli put do polaznog vrha.
– računati d(v0,vj) = d(v0,vj–1) + d(vj–1,vj), smanjujući j do j=n do j=1

Dijkstrin algoritam najkraćeg puta:

1. Svim vrhovima, osim polaznom, pridijelimo udaljenost ∞ i označimo ih kao neobrađene.

2. Promatramo sve neobrađene susjede vi vrha u kojem se trenutačno nalazimo i izračunavamo im udaljenosti od polaznog vrha kada do njih dolazimo iz onog u kojem jesmo. Označimo tu udaljenost s d*(v0,vi). Ako je d*(v0,vi) manja od udaljenosti koja je do tada bila upisana u labelu vrha vi, našli smo kraći put do vrha vi i upisujemo d*(v0,vi) u labelu, a trenutačni vrh postaje prethodnik vrha vi. Ako d*(v0,vi) nije kraća, samo nastavljamo s obilaskom susjeda.

3. Nakon osvježavanja labela neobrađenih vrhova (susjeda), za sljedeći vrh odabiremo onaj među neobrađenima (svima, ne samo susjedima prethodnog) koji je najbliži polaznom vrhu (najmanja udaljenost upisana u labelu) i nastavljamo ponavljajući postupak od točke 2 sve dok kao vrh najbliži polaznom ne bude izabran upravo odredišni vrh (cilj).

1.1. Implementacija

Implementacija rješenja problema najkraćeg puta je izvršena pomoću C#-a koristeći standardne biblioteke. Unutar *solutiona* nalazi se projekt `ShortestPathGUI` koji sadrži razrede `WeightedGraph` i `ShortestPathCalculator`.

2.2.1. Opis rješenja, složenost $O(V^2)$ – nije korišteno u implementaciji

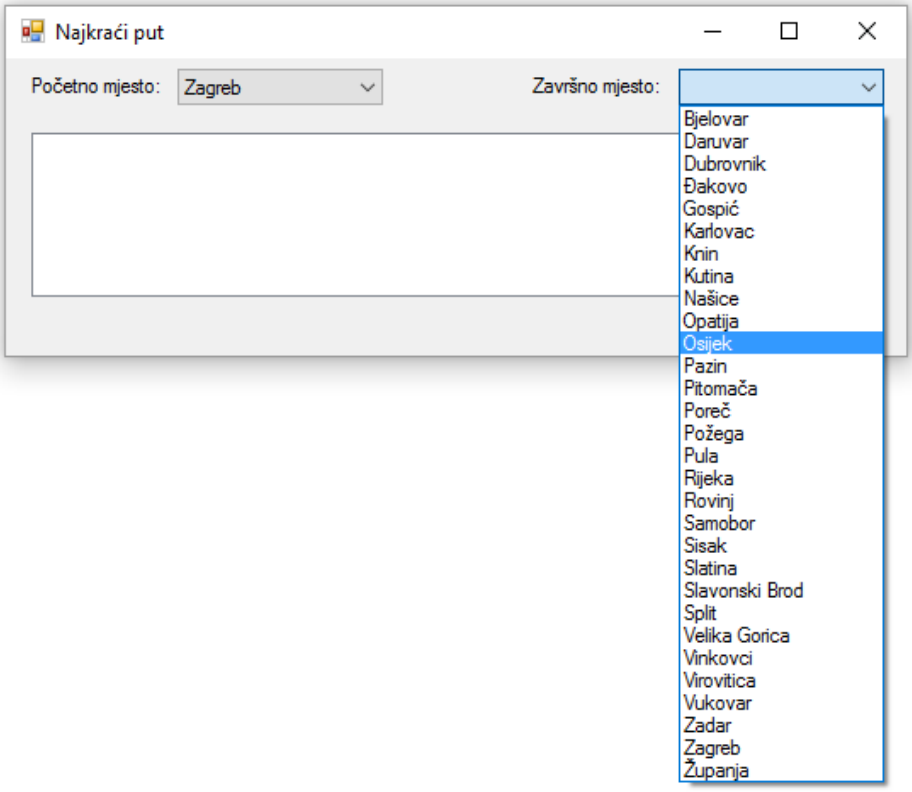
```
Dijkstra (source, dest)

for all vertices
     d(v)= inf
d(source) = 0;
ToBeCh = all vertices;
while there are vertices in ToBeCh
     v = a vertex in ToBeCh with the least d(v);
     if v == dest
           return ... ;
     remove v from ToBeCh;
     for all vertices u adjacent to v and in ToBeCh
           dnew(u)=d(v)+edge(v,u);
           if dnew(u) < d(u)
                 d(u) = dnew(u);
           predecessor(u) = v;
```

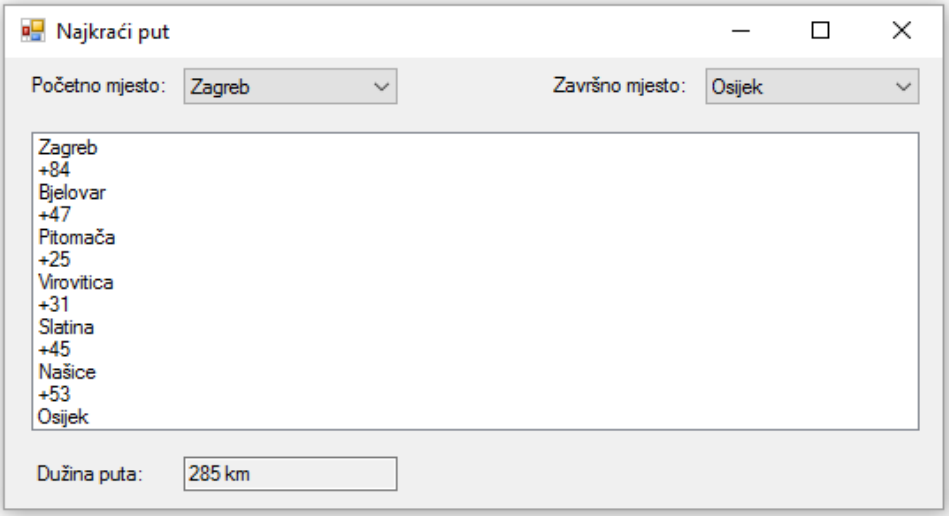
2.2.2. Opis rješenja, složenost $O(E+V \cdot \log_2 V)$ – korišteno u implementaciji [2]

```
Dijkstra (source, dest)

 For each u in V
     d[u] = ∞
   d[s] = 0;
   pred[s] = NULL;
   Q = (queue with all vertices)

   while (non-empty Q)
     u = extract-min(Q);
     for each v in Adj[u]
           if d[u] + w(u,v) < d[v]
                 d[v] = d[u] + w(u,v);
                 pred[v] = u;
```

1.2. Grafičko sučelje programa

Izgled grafičkog sučelja (GUI) programa:



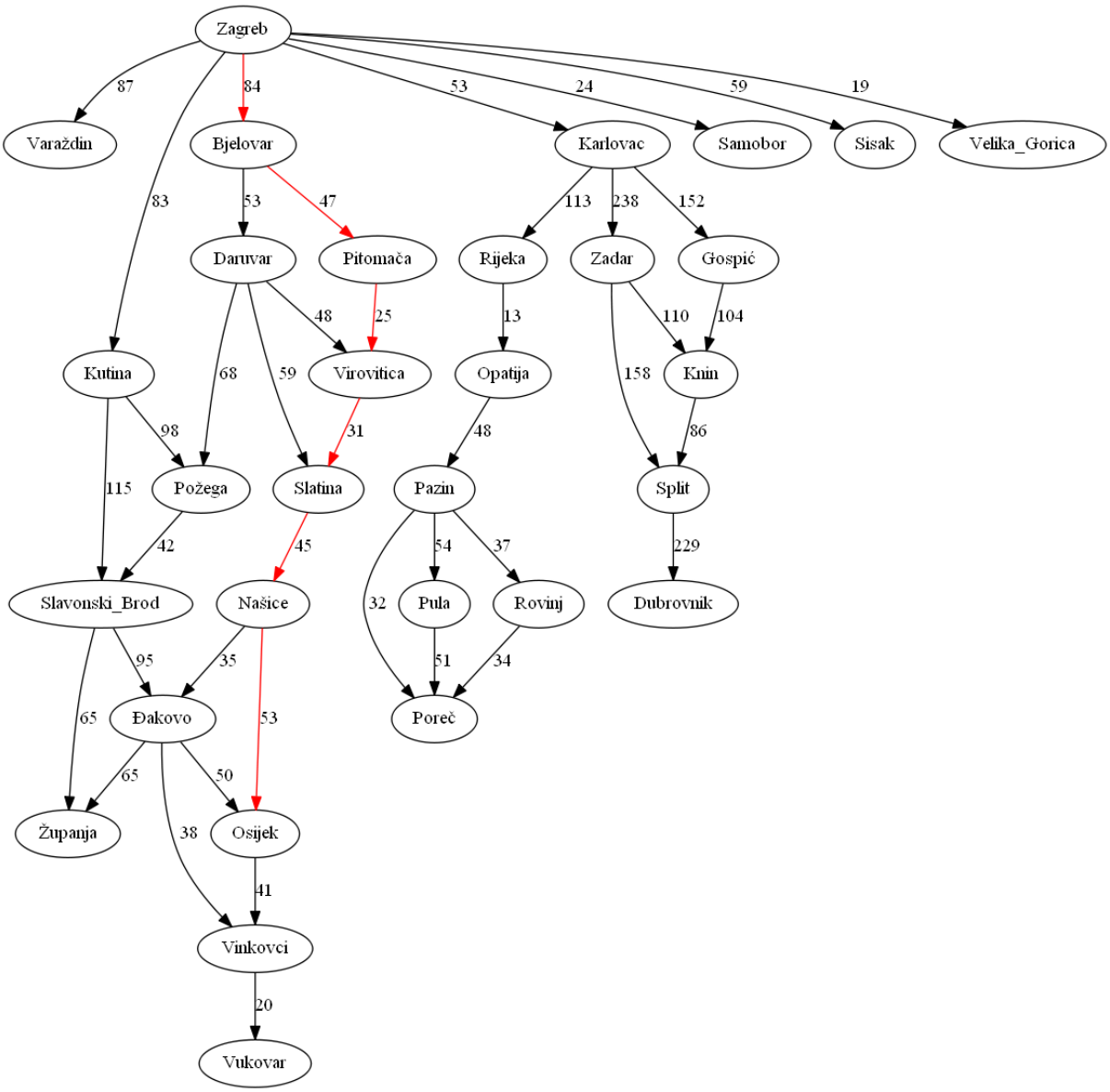


Na slikama je prikazan izgled programa za učitan primjer grafa *gradovi.txt*

Prikazano je grafičko sučelje za graf sa stvarnim mjestima (gradovima) između koji su izračunate udaljenosti u kilometrima pomoću *Google Maps* karata.
Nakon odabiranja početnog/završnog grada automatski se prikazuje najkraći put između ta dva grada i udaljenosti između gradova na tom putu. Prikazuje se i ukupna udaljenost najkraćeg puta između dva odabrana grada.

1.3. Korišten graf

Slika prikazuje korišten graf. Prikaz je napravljen pomoću *GraphViza*.
Crvenom bojom je prikazan primjer za put od Zagreba do Osijeka.



3. Zaključak

Dijkstrin algoritam najkraćeg puta je jednostavan algoritam za shvatiti i programski implementirati. Njegove primjene su očito korisne. Jedna od njih je prikazana u primjeru traženja najkraćeg puta između gradova.

4. Literatura

[1] *Nikica Hlupić, Damir Kalpić, Odabrani algoritmi nad grafovima (prezentacija predavanja), FER, Zagreb, 2009.*
[2] *Dekai WU*, *Dijkstra's Shortest Path Algorithm* (*Lecture 10*), *The Hong Kong University of Science & Technology, Hong Kong, 2005.*