Specifikacija i implementacija mrežnih modela prostornih podataka

Dražen Odobašić

Sažetak

U diplomskom radu opisuje se specifikacija i implementacija mrežnih modela prostornih podataka. U specifikaciji su teoretski obrađeni osnovni pojmovi prostornih podataka s naglaskom na mrežne strukture prostornih podataka. Objašnjeno je i modeliranje mrežnih podataka pomoću grafova i objektno orijentiranih modela.

U drugom dijelu objašnjena je implementacija mrežnog modela prostornih podataka kroz algoritme za traženje optimalnog puta između dvije lokacije. Objašnjeni su i slijedeći algoritmi: Dijkstra, Bellman-Ford, A*, D*, Floyd-Warshall i Johnson. Spomenut je i problem trgovačkog putnika koji je zasebno objašnjen.

Praktični primjer je realiziran upotrebom PostGIS i pgRouting prostornih proširenja za PostgreSQL objektno relacijsku bazu podataka koja se koristi za skladištenje prostornih podataka. Vizualizacija prostornih podataka napravljena je upotrebom Mapnik biblioteke za iscrtavanje i TileCache softverskog paketa koji priprema dijelove rasterskog mozaika. Korisnici kroz OpenLayers softverski paket pristupaju i koriste servis na poslužitelju.

Diplomski rad se temelji na slobodnim prostornim podacima. Digitalni model terena predstavljen je SRTM prostornim podacima koje nadopunjava CORINE baza podataka pokrova. Openstreetmap podaci o cestama se koriste za traženje optimalne putanje i za vizualizaciju iste.

Realizacija diplomskog rada izvedena je upotrebom slobodnog softvera i slobodnih podataka čime se želi skrenuti pozornost na skrivenu vrijednost koju oni posjeduju.

Ključne riječi: mrežni model, modeliranje, slobodni softver, slobodni prostorni podaci

Abstract

This thesis describes the work of the specification and implementation of spatial network data models. The specification theoretically treats basic concepts of spatial data with emphasis on network structure of spatial data. Modeling and spatial network data are discussed using graphs and object-oriented data models.

Second part deals with the implementation of the spatial network data model through the algorithms that have objective to find optimal path between the two locations. Following algorithms are explained: Dijkstra, Bellman-Ford, A*, D*, Floyd-Warshall and Johnson. Traveling salesman problem is also mentioned and explained separately.

Practical example is realized by using PostGIS and pgRouting spatial extensions for PostgreSQL object-relational database witch is used for the storage of spatial data. Visualization of spatial data is made by using Mapnik library for rendering and TileCache software package for tilling. Users use OpenLayers software to access and use the service on the server.

Thesis is based on a free spatial data. Digital terrain model is represented by SRTM spatial data that is fulfilled by the CORINE cover database. Openstreetmap information about roads is used to find optimal route and for visualization.

This thesis was created by using free software and free data and it draws attention to hidden value which they possess.

Keywords: network data, modeling, free software, free spatial data

Sadržaj

Popis slika

Poglavlje 1

Uvod

Gotovo je nemoguće zamisliti entitet i opisati ga atributima bez upotrebe prostorne komponente. Entitet je nešto što ima jasnu odvojivu postojanost, a može se nazvati i objektom, i predmet je proučavanja ontologije. Ontologija [Wikipedia, 2009b] je dio metafizike, i u filozofskom smislu je znanost koja se bavi proučavanjem prirode bitka, postojanja i stvarnosti. Ontologija, kao dio računalne znanosti, proučava općenitu klasifikaciju i odnose među stvarima koje postoje, te je u bliskoj vezi s modeliranjem podataka. Ključna razlika između ontologije i modeliranja podataka je u tome što ontologija pokušava razviti općenitu klasifikaciju onoga što postoji, a modeliranje podataka pokušava razviti klasifikaciju unutar određene domene.

Općenito o prostoru i modeliranju prostornih podataka može se više pronaći u drugom poglavlju. U skladu s temom diplomskog rada, više pozornosti se posvećuje mrežnim prostornim podacima, i njihovim reprezentacijama. Modeliranje mrežnih podataka se svodi na modeliranje pomoću grafova ili upotrebom objektno orijentiranog modeliranja kako je prikazano u poglavlju 2.3.

Nakon modeliranja, treće poglavlje obuhvaća detaljniji prikaz implementacije mrežnih modela prostornih podataka. Odnosno, u poglavlju 3.1 opisane su osnovne metode pretraživanja grafova. Pretraživanje grafova koristi se za određivanje putanje do traženog čvora. U poglavlje 3.2 se opisuju najčešće korišteni algoritmi koji se koriste u svrhu određivanja optimalne putanje između dva čvora. Optimalna putanja često predstavlja i najkraću putanju. Svaki algoritam je ukratko opisan, uključujući i problem trgovačkog putnika koji je obrađen u posebnom odjeljku. Naime, problem trgovačkog putnika je tzv. NP-potpuni problem čije se rješenje traži od 1930. godine, a više ovoj temi može se pročitati u odjeljku 3.3.

Četvrto poglavlje sadrži pregled softverskih paketa koji se koriste u izradi praktičnog dijela diplomskog rada u kojem se koristi isključivo slobodni softver.

O slobodnom softveru i zajednici koja se okupila oko njega, njegovim prednostima i manama može se pronaći u odjeljku 4.1. U zadnjem dijelu četvrtog poglavlja opisuje se svaka softverska komponenta koja se koristi u izradi praktičnog dijela diplomskog rada. Ukratko, operacijski sustav je Ubuntu 8.04.2, a za bazu podataka se koristi PostgreSQL 8.3 koji je proširen s PostGIS 1.3.3 i pgRouting 1.0.3 prostornim nadogradnjama. Korisnici pristupaju servisu koristeći internet preglednik (novije generacije) preko OpenLayers 2.7 sučelja iza kojeg se nalazi Tilecache 2.10 i Mapnik 0.6.0.

Osim slobodnog softvera korišteni su i slobodni prostorni podaci. Slobodnim prostornim podacima se smatraju oni podaci koji se mogu slobodno umnožavati, mijenjati i distribuirati u nekomercijalne svrhe. U diplomskom radu korišten je SRTM elevacijski model, CORINE baza podataka pokrova i Openstreetmap podaci o cestama.

Peto poglavlje opisuje korake koji se poduzimaju kako bi se preuzeli i pripremili prostorni podaci. Vizualizacija prostornih podataka podrazumijeva njihovo uključivanje u XML datoteku koja definira pravila iscrtavanja. Iscrtavanje dijelova rasterskog mozaika izvodi TileCache koji poziva Mapnik biblioteku za iscrtavanje. Na kraju petog poglavlja opisano je stvaranje interaktivne aplikacije i osnove korištenja iste.

Namjera ovog diplomskog rada je, osim pregleda i pojašnjavanja specifikacije i implementacije mrežnih modela prostornih podataka, pokazati da se geografski servisi mogu i trebaju temeljiti na slobodnom softveru i slobodnim prostornim podacima.

Poglavlje 2

Specifikacija mrežnih prostornih podataka

2.1 Što je to prostor?

Čovjek odavna pokušava definirati i modelirati prostor koji ga okružuje. Prvi matematičar koji proučava prostor je Euklid oko 300 g. p. n. e. Proučavao je odnose između udaljenosti i kutova na osnovu kojih je postavio osnove geometrije. Isaac Newton je prostor smatrao apsolutnim, odnosno da on postoji trajno i neovisno o tome da li se u njemu nalazi materija. Gottfried Leibniz je za prostor rekao da je to skup odnosa između objekata definiran međusobnom udaljenošću i smjerom. U 18. stoljeću Immanuel Kant opisuje prostor i vrijeme kao dijelove sistematske osnove koju čovjek koristi kako bi izgradio svoje iskustvo.

U 19. i 20. stoljeću se razvijaju teorije o "neeuklidskim" prostorima, stoga postoje hiperbolički i eliptički prostori koji imaju svoja pravila. Moderna definicija prostora [Britannica, 2009] kaže da je prostor neograničen, trodimenzionalni odsječak u kojem se pojavljuju objekti i događaji čija je pozicija i smjer relativna.

Prostor se može jednostavnije razumjeti napravimo li podjelu prostora na kategorije. Zubin, [Zubin, 1989], dijeli prostor na četiri kategorije:

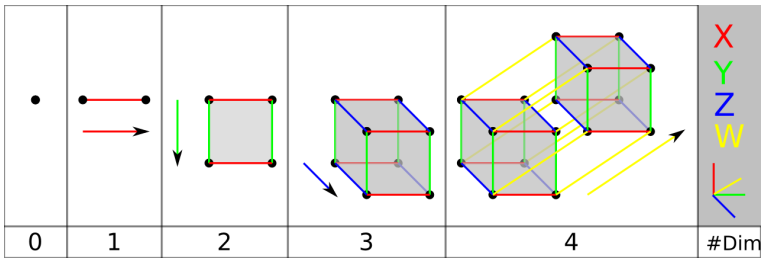
- prostor koji sadrži objekte koje svakodnevno koristimo (telefon, knjige, ključevi)
- prostor koji sadrži objekte koji su veći od ljudi, ali se mogu percipirati iz jedinstvene perspektive (zgrade, automobili)

- prostor koji sadrži objekte koje nije moguće percipirati iz jedinstvene perspektive (krajolici, oceani)
- prostor koji sadrži objekte koji su preveliki da bi ih čovjek mogao iskusiti (Sunčev sustav, galaksije)

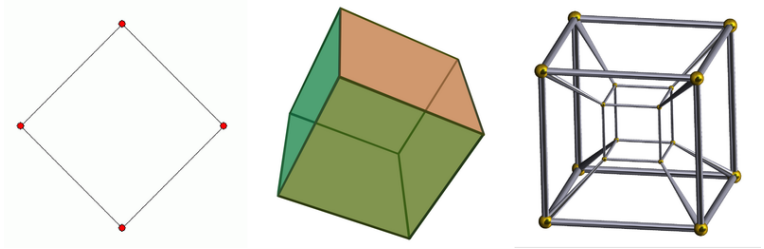
Temeljni koncept različitih pogleda na prostor je *geometrija*. Geometrija omogućava formalni prikaz apstraktnih obilježja i struktura prostora, a temelji se na načelu nepromjenjivosti (stalnosti) tih obilježja.

Uz pojam prostora veže se i njegova *dimenzionalnost*. Najboljim uvodnim tekstom o dimenzionalnosti prostora smatra se satirična novela "Flatland", o društvenom poretku Viktorijanske kulture, autora Edwina A. Abbota koja datira iz 1884. godine.

Dimenzija prostora se prikazuje kao uređene n-torke realnih brojeva. Na slici 2.1a mogu se vidjeti primjeri prostornih dimenzija. U 0D (bezdimenzionalnom) prostoru nalaze se točke. Ukoliko se pomakne 0D objekt u proizvoljnom smjeru dobit će se linija koja se nalazi u 1D prostoru. Pomicanjem linije u proizvoljnom smjeru dobit ćemo kvadrat koji se nalazi u 2D prostoru. Slijedeći korak je kocka koja se nalazi u 3D prostoru, a njenim pomicanjem dobiva se hiperkocka u 4D prostoru. Slika 2.1b predstavlja 2D prikaz kvadrata, kocke i hiperkocke.



(a) Jednostavan prikaz dimenzija prostora



(b) Kvadrat, kocka i hiperkocka

Slika 2.1: Dimenzionalnost prostora

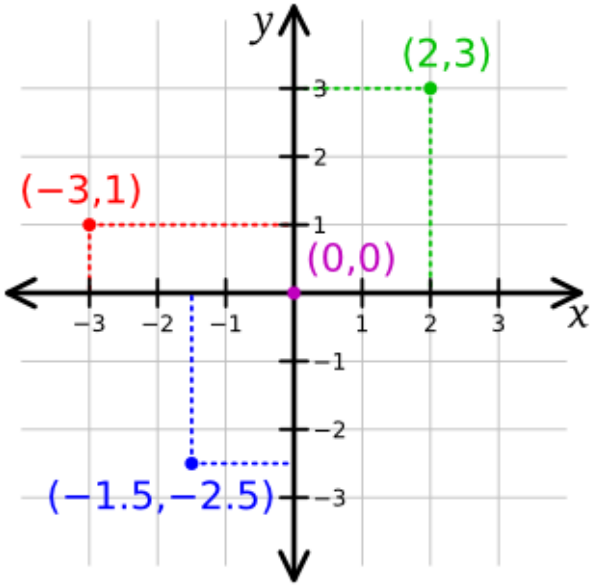
2.2 Modeli prostora

Ovaj diplomski rad se temelji na euklidskom prostoru (2.2.1), odnosno euklidskoj geometriji, koja je najjednostavnija za razumijevanje. Ostale implementacije prostora spomenut će se kasnije (odjeljak 2.2.2).

2.2.1 Euklidski model prostora

Prostorni objekti su uklopljeni u koordinatni sustav koji omogućava izračunavanje udaljenosti i kutova između točaka koordinatnog sustava. Takav sustav je osnova euklidskog prostora koji prostorne atribute (koordinate) nekog objekta prikazuje kao uređene n-torke realnih brojeva.

Euklidski prostor može imati proizvoljan broj dimenzija, ali zbog jednostavnosti, one su ograničene na dvije dimenzije, tzv. 2D prostor. Za svaki prostor može se postaviti *koordinatni sustav* koji ima jednu fiksnu polazišnu točku, *ishodište*. Kroz ishodište prolaze međusobno okomite linije koje nazivamo osi koordinatnog sustava (Slika 2.2).



Slika 2.2: Koordinatni sustav

Osnovni element euklidskog prostora je *točka*. Točka je opisana jedinstvenim parom realnih brojeva (X, Y). Brojevi X i Y predstavljaju udaljenosti mjerene od ishodišta u smjeru koordinatnih osi (Slika 2.2).

Slijedeći element je *linija*, definirana dvjema točkama. Linija može biti pravac (ako je neomeđena), zraka (ako je omeđena s jedne strane) ili pak linijski segment (ukoliko je omeđena s obje strane).

Više linijskih segmenata (određen broj) tvori *poliliniju*. Linijski segmenti se u ovom slučaju nazivaju *bridovi*, a točke *vrhovi*. Ukoliko je polilinija zatvorena ona se naziva *poligon*, a polilinija postaje *rub poligona*.

U euklidskom prostoru definirane su i *transformacije*. Transformacijom se smatra funkcija koja preslikava jednu točku u neku drugu točku. Razlikujemo slijedeće tipove transformacija:

euklidski koji čuvaju oblik i veličinu objekata (translacija)

sličnosti koji čuvaju oblik, ali ne nužno i veličinu objekata (promjena veličine)

Afini koji čuvaju afina svojstva, npr. paralelnost (rotacija, refleksija, smicanje)

projekcijski koji čuvaju projekcijska svojstva, npr. centralna projekcija (krug se preslikava u elipsu)

topološki koji čuvaju topološke osobine prostora (susjedstvo, povezanost)

2.2.2 Ostali modeli prostora

Osim euklidskog modela prostor se može prikazati i pomoću *teorije skupova*. Po toj teoriji u prostoru se nalaze elementi ili članovi koje je moguće modelirati. Skupine tih elemenata tvore skup, i uglavnom se broj članova skupa smatra određenim. Odnosi među elementima i skupovima kojima pripadaju članovi skupa naziva se članstvo. Skupovi, iako apstraktni, koriste se za modeliranje odnosa među elementima. Na popisu koji slijedi opisani su samo neki alati za modeliranje:

jednakost koja postoji samo ako oba skupa sadrže iste elemente

podskup koji može sadržavati samo elemente tog skupa

kardinalnost koja označava broj elemenata u skupu

prazan skup ili skup bez elemenata

presjek dva skupa daje novi skup koji sadrži samo one elemente koji se nalaze u oba skupa

unija dva skupa daju novi skup koji sadrži sve elemente oba skupa

razlika koja samo elemente prvog skupa koji nisu elementi drugog skupa

Topologija proučava promjene položaja i odnosa, odnosno određeni skup geometrijskih svojstva koji ostaje nepromijenjen nakon topoloških transformacija. U skladu s time može se definirati topološki prostor. U euklidskom prostoru moguće je definirati topološka obilježja:

- točka je početak ili kraj dužine
- točka je na rubu poligona
- točka je unutar poligona
- područje je otvoreno/zatvoreno/jednostavno
- područje je povezano

Tijekom razmatranja euklidskih prostora spominju se analitička i kombinatorna topologija. *Analitička topologija* proučava koncepte kao što su susjedstvo, blizina i otvorenost pomoću kojih zatim definira prostorne odnose kao što su povezanost ili granica. *Kombinacijska topologija* je primjerenija za računalno modeliranje zbog toga što se konačne i diskretne strukture lakše integriraju u računalne modele.

Mrežni prostori su tema ovog diplomskog rada a više se o njima može pročitati u poglavlju 2.3.

Metrički prostor je model prostora koji istražuje koncept udaljenosti između objekata. Primjerice, udaljenost između dva grada može biti definirana kao:

geodetska udaljenost mjeri se uzduž geodetske linije

Manhattan udaljenost jednaka je sumi razlika koordinata

vremenska udaljenost izražena je kao minimalno vrijeme putovanja

leksikografska udaljenost je apsolutna razlika između pozicije gradova na popisu

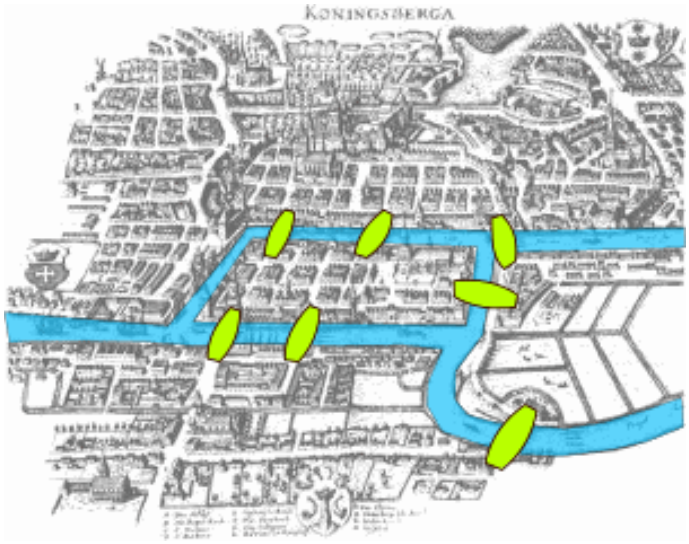
2.3 Mrežni model prostora

Velik broj prostornih problema može se modelirati korištenjem mrežnog modela. Mrežne strukture podataka koriste se svakodnevno, npr. ceste, željeznice ili za internet. Čovjek se svakodnevno susreće s problemima traženja puta od

početne točke A do odredišta B. Svaki takav problem se svodi na rješavanje problema korištenjem mrežnog modela.

Jedan od najpoznatijih problema je tzv. problem *7 mostova Königsberga* za kojeg je 1736. godine Švicarski znanstvenik Leonhard Euler donio negativan zaključak [Wikipedia, 2009a]. Königsberg u Prusiji danas se naziva Kaliningrad i nalazi se u Rusiji i od tadašnjih 7 mostova preostalo je 5, a samo još 2 iz vremena Eulera.

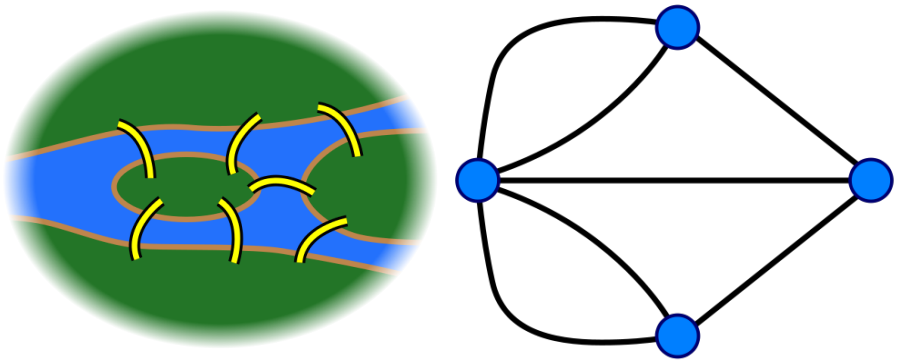
Problem mostova Königsberga svodi se na pitanje je li moguće prijeći preko svih sedam mostova tako da se preko svakog mosta prijeđe samo jednom (Slika 2.3).



Slika 2.3: Sedam mostova Königsberga

Euler iznosi pretpostavku da je izbor puta na kopnu nebitan (Slika 2.4a). Jedino važno svojstvo svake rute je slijed prelazaka preko mostova. Ta pretpostavka omogućuje mu ukloniti sva svojstva, osim kopnenih površina i mostova koji ih povezuju te na taj način stvori apstraktni prikaz problema (Slika 2.4b). U tom slučaju svaka kopnena površina je prikazana čvorom (eng. node) ili vrhom (eng. vertex), a svaki se most prikazuje kao brid (eng. edge). Bridovi povezuju čvorove i tvore matematičku strukturu koja se naziva graf.

Osim krajnjih točaka šetnje, dolazak na čvor i odlazak mora biti preko mosta (brida). Znači tijekom šetnje po grafu, broj puta kada se dođe do čvora, a da čvor nije kraj puta, jednak je broju puta kada se izađe s tog čvora. Ako se svaki most želi proći samo jednom, dakle znači da za svako kopno broj mostova koji ga dodiruju mora biti paran broj. Polovica tih mostova će biti prohodana kada se ide prema kopnu, a polovica kada se ide od kopna. Kako su svi čvorovi



(a) Grafička reprezentacija (b) Model grafa

Slika 2.4: Grafički prikaz problema mostova Konigsberga

spojeni s neparnim brojem bridova (jedan sa 5, ostali s 3), nije moguće proći sve mostove samo jednom.

Euler pokazuje da šetnja po grafu u kojoj se svaki brid koristi samo jednom ovisi o stupnju čvorova. Stupanj čvora predstavlja broj bridova koji ga dotiču. Šetnja će biti moguća samo ako je graf spojen i ako nula ili dva čvora imaju neparan stupanj. Ovakva šetnja po grafu naziva se eulerova šetnja, a ostvariva je samo ako započinje na čvoru s neparnim stupnjem i završi na čvoru s neparnim stupnjem. Postoji još i eulerov krug, u kojem šetnja mora početi i završiti na istom čvoru, i to samo ako je graf spojen i ako nema čvorova s neparnim stupnjem.

Ovakvim načinom razmišljanja Euler je postavio temelje teorije grafova. Osim toga zbog nepromjenjivosti grafa prema različitim prostornim transformacijama, čuva se informacija o odnosima čvorova i bridova, dok je informacija o položaju ili obliku nebitna. Time su postavljene osnove topologije.

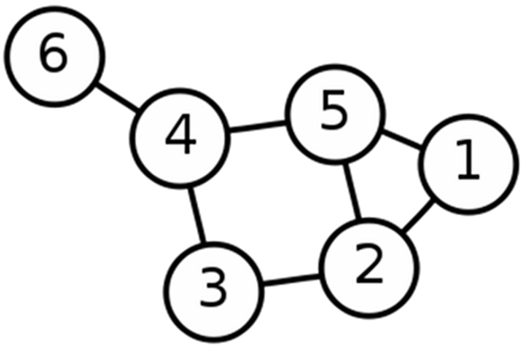
Mrežni prostorni podaci posebno su pogodni za uključivanje informacija o hijerarhiji [Car, 1996]. Hijerarhija skraćuje vrijeme potrebno za pretraživanje grafova jer se pretražuje manji skup podataka. Nadalje, istraživanja su pokazala da čovjek ima tendenciju dijeliti prostor oko sebe u različite hijerarhijske skupine [Montello, 1997]. Primjerice, planiranje putovanja automobilom iz Splita do Zagreba čovjek će prvo započeti razmatranjem autocesta, odnosno prometnica najviših kategorija, a tek na kraju iskoristiti prometnice nižih kategorija i lokalne ceste kako bi došao do željene lokacije.

2.3.1 Modeliranje grafovima

Graf je skup čvorova koji su spojeni sa skupom bridova. Dakle za graf G (Slika 2.5) može se zapisati $G = (V, E)$ gdje V označava skup čvorova, a E skup bridova.

Graf je *konačan* ako je broj čvorova $|V|$ i broj bridova $|E|$ konačan (poznat), suprotnost konačnom grafu je *beskonačan* graf.

Brid povezuje dva čvora koja se nazivaju *krajnji čvorovi*. Ukoliko dva čvora imaju zajednički brid onda se nazivaju *susjedni* čvorovi. Brid (n_i, n_j) se naziva *petlja* ako je $n_i = n_j$. *Paralelni bridovi* imaju zajedničke krajnje čvorove.



Slika 2.5: Prikaz grafa

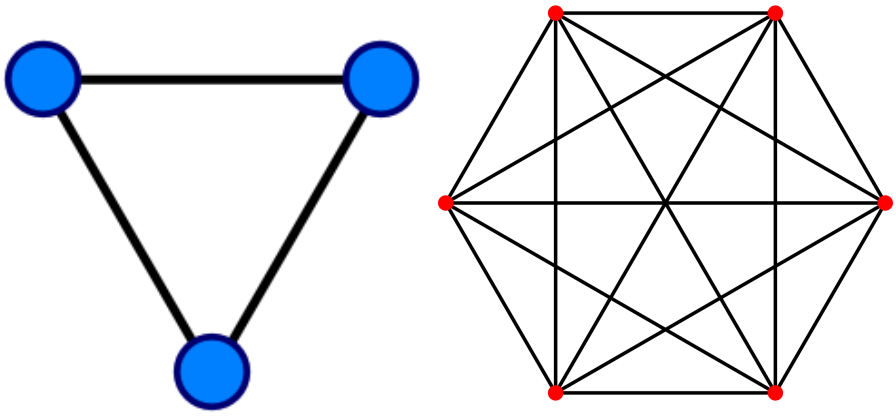
Tipovi grafova

Graf je *jednostavan* (Slika 2.6a) ako nema petlje i paralele bridove. Ako graf sadrži paralelne bridove, a ne sadrži petlje naziva se *multigraf*. Graf je *kompletan* (Slika 2.6b) ako svaki par različitih čvorova tvori jedan brid.

Grafovi u kojima nedostaje samo nekoliko bridova nazivaju se *gusti grafovi*, dok oni koji imaju samo nekoliko bridova su *raštrkani grafovi*. Ako se graf može prikazati u ravnini a da se dva brida nesjeku, dakle bridovi se mogu doticati samo u čvorovima, zovemo *ravninskim grafom* (Slika 2.7).

Usmjereni graf (Slika 2.8) ima određen smjer na svim bridovima. Poredak završnih točaka određuje smjer brida, stoga brid e_1 započinje na čvoru n_1 i završava na čvoru n_2, $e_1 = (n_1, n_2)$. Graf ne mora nužno sadržavati usmjereni čvor od n_2 prema n_1. *Dvosmjerni graf* ili *simetrični graf*, je graf u kojem za svaki brid n_i, n_j postoji brid (n_j, n_i). Graf bez određenih smjerova naziva se *neusmjereni graf*.

Dva grafa G_1 i G_2 su *izomorfna* ukoliko oba grafa imaju ista susjedstva čvorova. Tako da za svaki označeni par u, v čvorova, dakle broj bridova koji



(a) Jednostavan graf (b) Kompletan graf

Slika 2.6: Jednostavan i kompletan graf

spajaju u, v u G_1, jednak je broju bridova koji spajaju u i v u G_2 . Izomorfnost dvaju grafova teško je odrediti.

Za svaki čvor se može odrediti *stupanj čvora*. Koji označava broj bridova spojenih u tom čvoru. Usmjereni graf ima *ulazni stupanj* koji predstavlja broj bridova koji dolaze u čvor, i *izlazni stupanj* koji predstavlja broj bridova koji odlaze iz čvora.

Ukoliko iz grafa maknemo određen broj čvorova ili bridova, dobit ćemo strukturu koja se naziva *podgraf* (Slika 2.9). Micanjem čvora nestaju i svi bridovi koji su povezani s tim čvorom, ali micanjem brida može se pojaviti *izoliran čvor*.

Šetnjom se naziva niz slijedećih čvorova, ali ako taj niz sadrži samo jedinstvene čvorove (bez ponavljanja) naziva se putanja. Ako su prvi i zadnji čvor u tom nizu identični onda se niz naziva ciklus. Na slici 2.5 može se vidjeti da je:

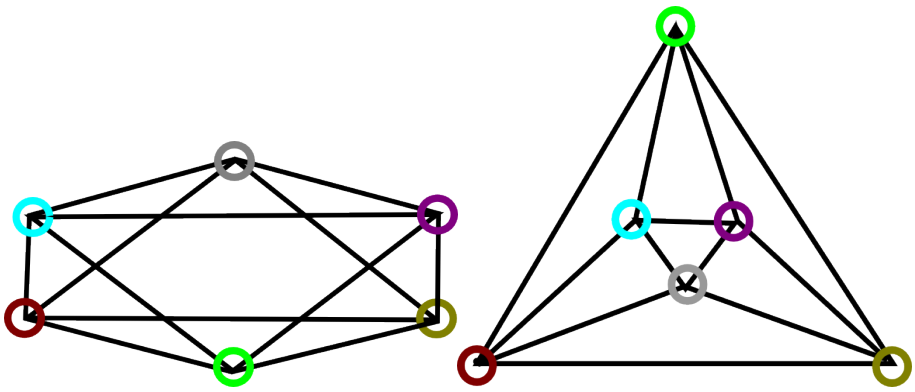
šetnja $(1,2) \to (2,3) \to (3,4) \to (4,5) \to (5,1)$

putanja $(1,2) \to (2,5) \to (5,4) \to (4,6)$

ciklus $(1,2) \to (2,3) \to (4,5) \to (5,1)$

Bilo koja dva čvora n_i i n_j su spojena ako u grafu postoji barem jedna putanja od n_i do n_j. Graf je *spojen* ako postoji najmanje jedan put za bilo koji par čvorova.

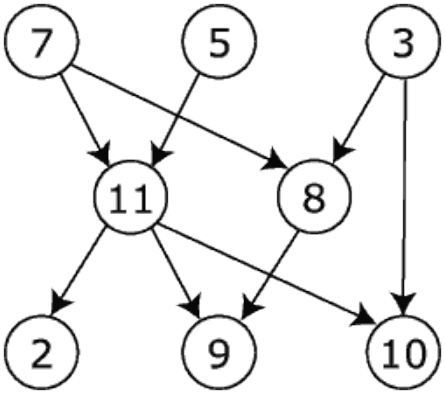
Svaki brid u grafu može imati dodijeljen broj. Taj broj $w(e_i)$ naziva se *težina brida*, a graf s težinom brida naziva se *težinski graf*. Težina brida može predstavljati bilo što, primjerice trajanje putovanja izraženo u satima.



(a) Ravninski graf prije transformacije

(b) Ravninski graf poslije transformacije

Slika 2.7: Ravninski graf

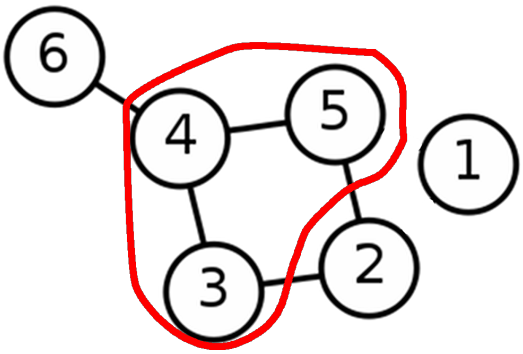


Slika 2.8: Usmjereni graf

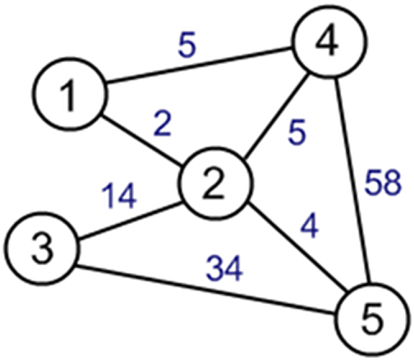
Na slici 2.10 vidi se brid koji spaja dva čvora n_1 i n_2 i ima težinu $w(n_1, n_2) = 2$. Težina putanje će biti suma svih težina čvorova i bridova te putanje. Primjerice putanja koja uključuje bridove $(n_1, n_2) \rightarrow (n_2, n_4) \rightarrow (n_4, n_5)$ imat će ukupnu težinu 65, dok će putanja $(n_1, n_2) \rightarrow (n_2, n_3) \rightarrow (n_3, n_5)$ imati ukupnu težinu 50. Putanja s najmanjom težinom naziva se *najkraća putanja*, što je i tema trećeg poglavlja.

Predstavljanje grafova

Postoji više načina na koji se graf može predstaviti (vizualizirati). Najpogodnije za računalnu primjenu su matrice susjedstva, jer one dobro prikazuju odnose među čvorovima i bridovima.



Slika 2.9: Podgraf i izoliran čvor



Slika 2.10: Težinski graf

$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Matrica susjedstva $A(G)$ grafa G je veličine $n * n$, a n predstavlja broj čvorova u G. Element $A(G)$ ima vrijednost $a_{ij} = 1$ ako postoji brid između čvorova n_i i n_j, a $a_{ij} = 0$ ukoliko brid ne postoji. Matrica će biti simetrična ako graf nije usmjeren. Na sličan način se može predstaviti težinski graf, a u tom slučaju su vrijednosti matrice zamijenjene težinskim vrijednostima.

Grafovi se mogu prikazati pomoću liste susjedstva, koja za svaki čvor grafa n_i sadrži popis čvorova susjednih tom čvoru.

$$\begin{pmatrix} A & C, E \\ B & D, E, F \\ C & A \\ D & B, F \\ E & B, F \\ F & B, D \end{pmatrix}$$

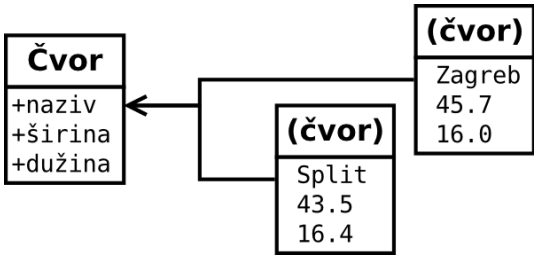
Lista susjedstva se u računalnoj primjeni predstavlja kao vezana lista ili obično polje podatka, ali implementacija najviše ovisi o potrebama projekta.

2.3.2 Objektno orijentirano modeliranje

U ovom modelu svijet je predstavljen skupom objekata, u kojem su objekti istog tipa organizirani u klase. Objektno orijentirani model pruža apstraktni model stvarnosti temeljen na identitetu (jedinstvenosti), enkapsulaciji (sakrivanju), nasljeđivanju, polimorfizmu (višeobličju) i asocijaciji (pridruživanju).

Objekt je nešto što posjeduje samostalnost i ima neke opisive osobine. Objektna klasa opisuje grupu objekata koji posjeduju slične atribute, i zajedničke operacije te zajedničke odnose među ostalim objektima. Svaki objekt je instanca neke klase.

U slučaju mrežnog modeliranja postoji klasa čvorova koja će imati atribute kao što je naziv, geografska dužina i širina, kao što je prikazano na slici 2.11. Gdje se može vidjeti klasa "čvor" i dvije instance te klase. Odnosno čvor s nazivom "Zagreb", širinom "45.7" i dužinom "16.0" je instanca klase "Čvor", isto vrijedi i za čvor s nazivom "Split".



Slika 2.11: Primjer mrežnog objektnog modela

Svaki objekt ima jedinstveni *identitet* koji je neovisan o vrijednostima atributa. Identitet objekta se stvara u trenutku kada se stvara objekt, a uništava se u trenutku uništavanja objekta. Tijekom svog "života" identitet objekta je nepromjenjiv.

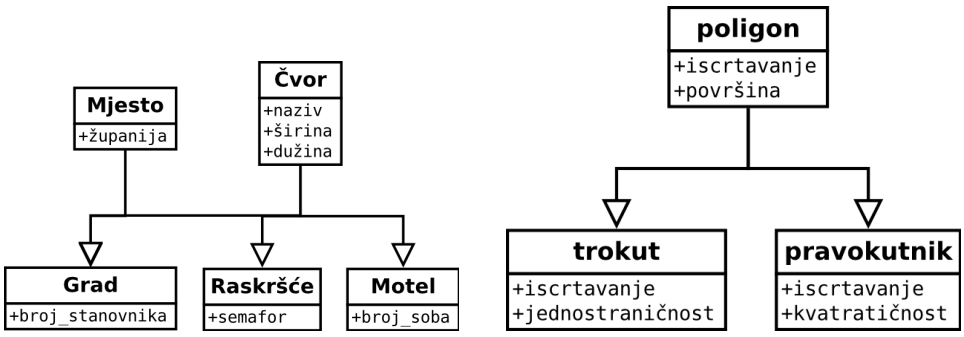
Enkapsulacija osigurava odvajanje unutarnjih mehanizama ponašanja objekta od vanjskih metoda pristupa. Između ostalog enkapsulacija smanjuje

kompleksnost objektnog modela tako što odvaja one osobine koje objekt prikazuje od onoga kako objekt postiže te osobine. Ukoliko se objekt želi ponovo iskoristiti, podkomponenete se moraju ponašati na predvidljiv način, a za to je ključno osigurati izolaciju unutarnjih osobina objekta od vanjskih utjecaja.

Nasljeđivanje omogućuje da se osobine definirane u jednoj klasi iskoriste u novoj, specijaliziranoj klasi. Postoje dva tipa nasljeđivanja: generalizacija i specijalizacija. *Generalizacija* grupira nekoliko klasa objekata sa zajedničkim operacijama u općenitiju superklasu. Obratni proces prerađivanja općenite superklase naziva se *specijalizacija*.

Nasljeđivanje se može smatrati strogo hijerarhijskim ukoliko klasa nasljeđuje osobine samo jedne superklase, i naziva se *jednostruko nasljeđivanje*. U slučaju *višestrukog nasljeđivanja* klasa može naslijediti osobine od više superklasa (Slika 2.12a).

Nasljeđivanje omogućuje objektima drugačije ponašanje u drugačijem kontekstu, ta osobina se naziva *polimorfizam* (Slika 2.12b). Operacije superklase se mogu primijeniti na objekte podklasa, obratno nije moguće, tj. nije moguće primijeniti operacije definirane u podklasi na objekte nadklase, ali podklasa može redefinirati operaciju za vlastitu primjenu.



(a) Nasljeđivanje (b) Polimorfizam

Slika 2.12: Primjer: nasljeđivanje i polimorfizam

Na slici 2.12 se može vidjeti da su čvorovi općenite točke neke cestovne mreže. Svaki taj čvor može predstavljati raskršće, grad ili motel. Svi imaju zajedničke atribute: naziv, geografsku širinu i dužinu, ali svaki od njih može imati i vlastite atribute. Raskršće može imati atribut semafor, koji označava da li se na tom raskršću nalazi semafor, grad broj stanovnika, motel pak broj dvokrevetnih soba. Višestruko nasljeđivanje je vidljivo u slučaju klase grad koja nasljeđuje osobine i operacije klasa čvor i mjesto.

Asocijacija grupira objekte kako bi se modelirale pojave s kompleksnom internom strukturom. Specijalni tip asocijacije je agregacija. *Agregacija* stvara

kompleksne objekte iz jednostavnih. Nekoliko objekata može se kombinirati tako da se stvori objekt višeg reda koji se naziva *agregat*. Dijelovi agregata se nazivaju *komponente*, a svaka komponenta ima vlastitu funkcionalnost. Operacije na agregatima nisu kompatibilne sa operacijama na komponentama. Na slici 2.13 država se sastoji od županija, a općina je dio županije.

Slika 2.13: Primjer: agregacija i propagacija

Propagacija opisuje na koji način su vrijednosti atributa jedne klase izvedene iz vrijednosti atributa druge klase. Na slici 2.13 može se vidjeti kako izgleda postupak izračuna stanovništva. Stanovništvo županije jednako je zbroju stanovništva svih općina u toj županiji, a stanovništvo države jednako je zbroju stanovništva svih županija u državi.

Poglavlje 3

Implementacija mrežnih prostornih podataka

Osnovni problem snalaženja u prostoru je traženje puta od točke A do točke B. Proučavanjem ljudskih procesa koji se događaju prilikom traženja puta može se zaključiti kako čovjek u prostoru oko sebe prvo traži poznate točke. Poznate točke su obično najprepoznatljiviji objekti u okolici, npr. crkva, kazalište, tržnica, odnosno dobro prepoznatljivi objekti. Osim toga čovjek u prostoru oko sebe intuitivno dijeli prostor u različite hijerarhijske kategorije, a istraživanja su pokazala kako će hijerarhija doći do većeg izražaja ako su udaljenosti između točaka veće.

Izraz traženje puta (eng. wayfinding) prvi je koristio Kevin A. Lynch u dijelu *The Image of the City* izdane 1960. U tom dijelu Lynch definira traženje puta kao *konzistentnu upotrebu i organizaciju jasnih osjetilnih znakova iz vanjskog okruženja* [Wikipedia, 2009c]. Rad se temelji na petogodišnjem istraživanju koje proučava kako korisnici opažaju i organiziraju prostorne informacije dok se kreću (snalaze) kroz gradove. Istraživanjem, u tri grada (Boston, Jersey City i Los Angeles), Lynch je primijetio da korisnici shvaćaju svoje okruženje na konzistentan i predvidljiv način. Oni stvaraju mentalne mape sa pet elemenata:

putanja ulice, pločnici, staze i ostali putovi kojima ljudi putuju

bridova uočene zapreke kao što su zidovi, zgrade i obalne linije

područja relativno veliki dijelovi grada, prepoznatljivi po identitetu ili značaju

čvorova točke fokusa, raskršća ili znamenite lokacije

važnih objekata koje je moguće identificirati, i koji služe kao referentne točke

U ovom poglavlju prvo će se pojasniti na koji način se može *hodati* po grafu, odnosno pretraživati čvorove. Ostatak poglavlja bavi se algoritmima za traženje najkraćeg puta u nekom grafu, a na kraju će se spomenuti i poseban problem tzv. problem trgovačkog putnika.

Pronalaženje najkraćeg puta ima veliku primjenu u računalnom softveru zabavnog karaktera, odnosno dio je umjetne inteligencije kojom raspolaže računalo. Osim toga mrežni modeli mogu se iskoristiti za optimizaciju i vizualizaciju računalnih mreža.

Bitno je spomenuti način procjene efikasnosti nekog algoritma. Osnovni načini za takvu procjenu temelje se na vremenu potrebnom da se neki algoritam izvrši i količini memorijskog prostora potrebnog za izvršavanje tog algoritma. Gotovo uvijek vrijeme i prostor ovise o količini podataka koje treba obraditi. Efikasnost algoritma označava se $O(f(n))$ gdje je $f(n)$ funkcija koja predstavlja efikasnost algoritma, dok n predstavlja količinu podataka koji se obrađuju. Ova procjena algoritma uvijek predstavlja najgori mogući slučaj. Tablica (3.1) prikazuje neke izraze za efikasnost algoritma.

$O(1)$	Konstantno	Vrlo brzo neovisno o količini podataka
$O(logn)$	Logaritamsko	Brzo, npr. binarno pretraživanje
$O(n)$	Linearno	Umjereno brzo, npr. linearno pretraživanje
$O(nlogn)$	Sub-Linerano	Umjereno, npr. algoritmi pretraživanja
$O(n^k)$	Polinomno	Sporo, npr. traženje najkraćeg puta
$O(k^n)$	Eksponencijalno	Jako sporo npr. problem putujućeg trgovca

Tablica 3.1: Efikasnost algoritma

3.1 Pretraživanje po grafu

Osnovna operacija u bilo kojem spojenom grafu je sistematsko pretraživanje (hodanje) po grafu, odnosno po njegovim čvorovima. Pretraživanja imaju višestruku primjenu kao primjerice određivanje prostora problema (buduće vjerojatne pozicije u šahu) ili pretraživanje mreže kablova u potrazi za kvarom. Postoje dva osnovna alternativna pristupa:

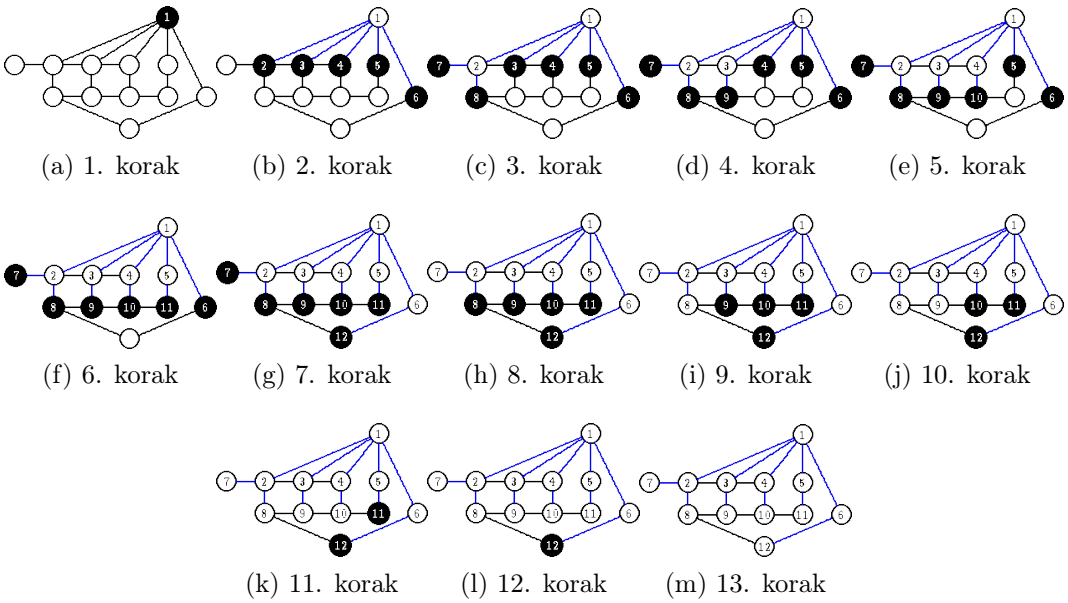
- dubinsko pretraživanje (depth-first)
- širinsko pretraživanje (breadth-first)

Pretraživanje po grafu će dotaknuti svaki čvor u grafu ali samo ako je taj graf spojen. Graf se naziva spojenim ako postoji barem jedan put između svih čvorova u grafu (poglavlje 2.3.1).

3.1.1 Širinsko pretraživanje

Širinsko pretraživanje grafa počinje od ishodišnog čvora te prvo gleda sve čvorove koji su spojeni sa određenim čvorom prije nego krene dublje. Kompozicija slika 3.1 prikazuje korake u radu algoritma za širinsko pretraživanje. Algoritam je zapisan u pseudokôdu te glasi:

```
posjeti(početni_čvor)
red <- početni_čvor
DOK red nije prazan NAPRAVI
  x <- red
  ZA SVAKI y koji tvori brid (x,y) i
           ako y već nije posjećen NAPRAVI
    posjeti(y)
    red <- y
  KRAJ
KRAJ
```



(a) 1. korak (b) 2. korak (c) 3. korak (d) 4. korak (e) 5. korak

(f) 6. korak (g) 7. korak (h) 8. korak (i) 9. korak (j) 10. korak

(k) 11. korak (l) 12. korak (m) 13. korak

Slika 3.1: Širinsko pretraživanje grafa

Prije samog pretraživanja čvorova potrebno je odrediti ishodišni čvor. Taj čvor se stavlja u `red` . Red sadrži sve čvorove koje još treba posjetiti. U prvom

koraku ulazi se u petlju koja će završiti samo ako je red prazan. Uzima se čvor `x` (Slika 3.1a) iz reda traže se svi čvorovi s kojima prvi čvor ima brid. Ukoliko postoji `(x,y)` i čvor `y` nije već posjećen, posjetit će se svaki `y` i staviti na kraj reda. (Slika 3.1b). U ovom koraku posjećeni su čvorovi (2,3,4,5,6).

Ukoliko `red` nije prazan uzima se prvi slijedeći čvor iz reda (2), i ponovo se za njega traže svi bridovi. Kako je čvor (3) već posjećen posjetit će se samo čvorovi (7,8) i staviti na kraj reda (3.1c). Slijedeći zanimljiv korak prikazan na slici 3.1h, prikazuje trenutak u kojem je iz reda uzet čvor (7). Naime kako su svi čvorovi u okolini čvora (7) već posjećeni, red se neće nadopunjavati nego će se iz reda uzeti novi čvor.

Algoritam završava u onom trenutku kada u redu nema više čvorova (Slika 3.1m). Ovakav način pristupa redu naziva se *FIFO*[1] i radi tako da se uvijek iz reda uzima član na početku, a novi član se dodaje na kraj reda.

3.1.2 Dubinsko pretraživanje

Dubinsko pretraživanje grafa počinje od ishodišnog čvora i gleda sve čvorove u nekoj grani prije nego se pomiče natrag na iduću granu. Dubinsko pretraživanje ukratko je prikazano na slici 3.2. Ovaj algoritam je specifičan zato što je tzv. rekurzivan algoritam, zapisan u pseudokôdu glasi:

```
algoritam  dft(x)
   posjeti(x)
   ZA SVAKI y koji tvori brid (x,y) NAPRAVI
     AKO y nije posjećen ONDA
        dft(y)
```

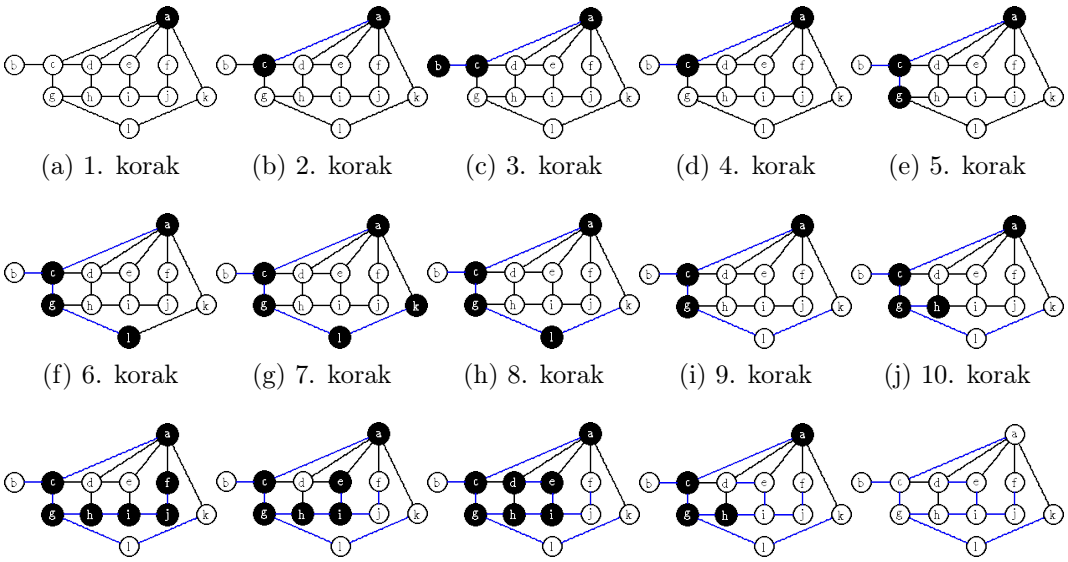
Algoritam posjećuje odabrani ishodišni čvor (a) (Slika 3.2a) i traži slijedeći čvor u grafu s kojim ishodišni ima brid. Ukoliko taj čvor nije posjećen ponovno se poziva algoritam, ali ovaj put se uzima taj drugi čvor kao ishodišni (Slika 3.2b).

Slijedeći korak (Slika 3.2c) u kojem algoritam posjećuje čvor (b) također ima zanimljiv karakter, ali kako je jedini čvor povezan s (b) čvor (c), algoritam se neće ponovo pozivati, nego se vraća jedan korak unatrag (Slika 3.2d) gdje uzima slijedeći čvor s kojim čvor (c) ima brid, čvor (g). Nakon posjećenog čvora (g) traži prvi ne posjećeni čvor s kojim (g) ima brid, u ovom slučaju (l).

Vraćanjem unatrag algoritam na kraju posjećuje sve čvorove grafa (Slika 3.2o). Ovaj način pristupa čvorovima naziva se LIFO[2], odnosno prvo se obra-

[1] eng. first-in-first-out

[2] last-in-first-out



(a) 1. korak (b) 2. korak (c) 3. korak (d) 4. korak (e) 5. korak

(f) 6. korak (g) 7. korak (h) 8. korak (i) 9. korak (j) 10. korak

(k) 11. korak (l) 12. korak (m) 13. korak (n) 14. korak (o) 15. korak

Slika 3.2: Dubinsko pretraživanje grafa

đuje čvor koji je zadnji dodan u red, iako je sam red zbog rekurzije funkcije na neki način sakriven.

3.1.3 Ostali načini pretraživanja

Postoji još nekoliko načina pretraživanja grafa, a jedna od implementacija naziva se *dubinski ograničeno pretraživanje*. Algoritam je identičan običnom dubinskom pretraživanju, ali koristi informaciju o maksimalnoj dopuštenoj dubini pretraživanja. Maksimalna dopuštena dubina definira koliko će čvorova algoritam posjetiti prije nego se počne vraćati natrag. Time su onemogućene beskonačno dugačke putanje i osigurava se završavanje algoritma. Problem je u tome što unatoč njegovom završavanju, algoritam ne osigurava pronalaženje rješenja.

Druga implementacija se naziva *iterativno produbljujuće dubinsko pretraživanje*. Ovaj algoritam upotrebljava dubinski ograničeno pretraživanje, i u svakom idućem koraku povećava maksimalnu dopuštenu dubinu sve dok ne pronađe traženi čvor. Osim toga ova metoda može iskoristiti heurističke metode kao što su *eliminirajuća heuristika* ili *alfa-beta čišćenje odbacivanjem*, koje poboljšavaju vrijeme pretraživanja odbacujući nemoguća rješenja, odnosno najlošija rješenja. Osim toga zbog malene početne dubine pretraživanja algoritam je vrlo brz i pruža dovoljno dobru informaciju o potencijalnom rješenju. Ovakav pristup najčešće se iskorištava kao dio umjetne inteligencije računalnog

softvera zabavnog karaktera.

Slijedeća implementacija naziva se *prvi najbolji* (eng. best-first), koja pretražuje graf tako da za slijedeći čvor bira onaj čvor koji najbolje zadovoljava neko pravilo. Pravilo je obično neka funkcija heuristike čija implementacija ovisi od slučaja do slučaja. Specijalni slučaj te metode je tzv. *pohlepan algoritam* koji prilikom određivanja najboljeg idućeg rješenja to rješenje traži lokalno, bez uzimanja okoline u obzir. Ovaj algoritam uglavnom neće dati optimalno rješenje, ali zbog svoje brzine upotrebljava se za određivanje približnih rješenja.

3.2 Algoritmi za traženje optimalnog puta

Druga osnovna operacija na mrežnim strukturama podataka je traženje optimalnog puta između čvorova u mreži. Algoritmi za traženje najkraćeg puta se koriste kao dio navigacijskih sustava u automobilima. Najkraći put ne mora nužno značiti i metrički najkraći put. Svrha algoritma je naći optimalno rješenje ovisno o zahtjevima korisnika. Tako primjerice kriterij može biti vrijeme, ili najmanja cijena putovanja sa obzirom na cestarine. U ovom diplomskom radu razmatranja će biti usredotočena na najkraći metrički put.

Najpoznatiji algoritam za traženje najkraćeg puta osmislio je Edsger Dijkstra, koji po njemu nosi ime. Algoritam Dijsktra i recimo A* svrstavaju se u grupu algoritama koji traže najkraći put iz jednog početnog (čvora), dok recimo Floyd-Warshall algoritam traži sve najkraće putove za sve čvorove.

3.2.1 Algoritam Dijkstra

Algoritam Dijkstra je osmislio nizozemski računalni znanstvenik Edsger Dijkstra, rođen 1930. godine u Rotterdamu. Osim algoritma za traženje najkraćeg puta Dijsktra je zadužio računalnu znanost i sa raznim algoritmima za alociranje resursa kao i implementacijama višezadaćnih operacijskih sustava.

Algoritam funkcionira na spojenim težinskim grafovima koji imaju samo pozitivne težine bridova. Najkraćim putem smatra se onaj koji ima najmanju akumuliranu težinu. Kako je već prije navedeno Dijkstrin algoritam svrstava se u grupu algoritma koji imaju jedan početni (ishodišni) čvor, što znači da će algoritam za svaki čvor u grafu pronaći najkraći put u odnosu na ishodišni čvor.

Primjerice ako čvorove smatramo gradovima, a bridove vezama između gradova, Dijkstrin algoritam pronaći će najkraće udaljenosti između početnoga grada i svih ostalih gradova.

Dijkstrin algoritam će biti obrađen primjerom prikazanim na slici 3.3, a njegov pseudokôd je slijedeći:

```
ZA SVAKI čvor v u grafu:
    udaljenost[v] := beskonačno
    prethodni[v] := nedefinirano
udaljenost[ishodišni_čvor] := 0
Q := skup svih čvorova u grafu

DOK Q nije prazan skup:
    u := čvor iz Q sa najmanjom udaljenosti()
    AKO JE udaljenost[u] = beskonačno:
       PREKINI
    MAKNI u iz Q
    ZA SVAKOG susjeda v od u:  //ako v postoji u skupu Q
        putanja := udaljenost[u] + udaljenost_između(u, v)
        AKO JE putanja < udaljenost[v]:
            udaljenost[v] := putanja
            prethodni[v] := u
```

U prvom koraku za svaki čvor v u grafu *udaljenost*, koja predstavlja akumulirane težine do tog čvora, postavlja se vrijednost beskonačno. Također svi *prethodni* čvorovi, oni koji čine optimalnu putanju od ishodišnog čvora, su nedefinirani. Udaljenost ishodišnog čvora postavlja se na 0.

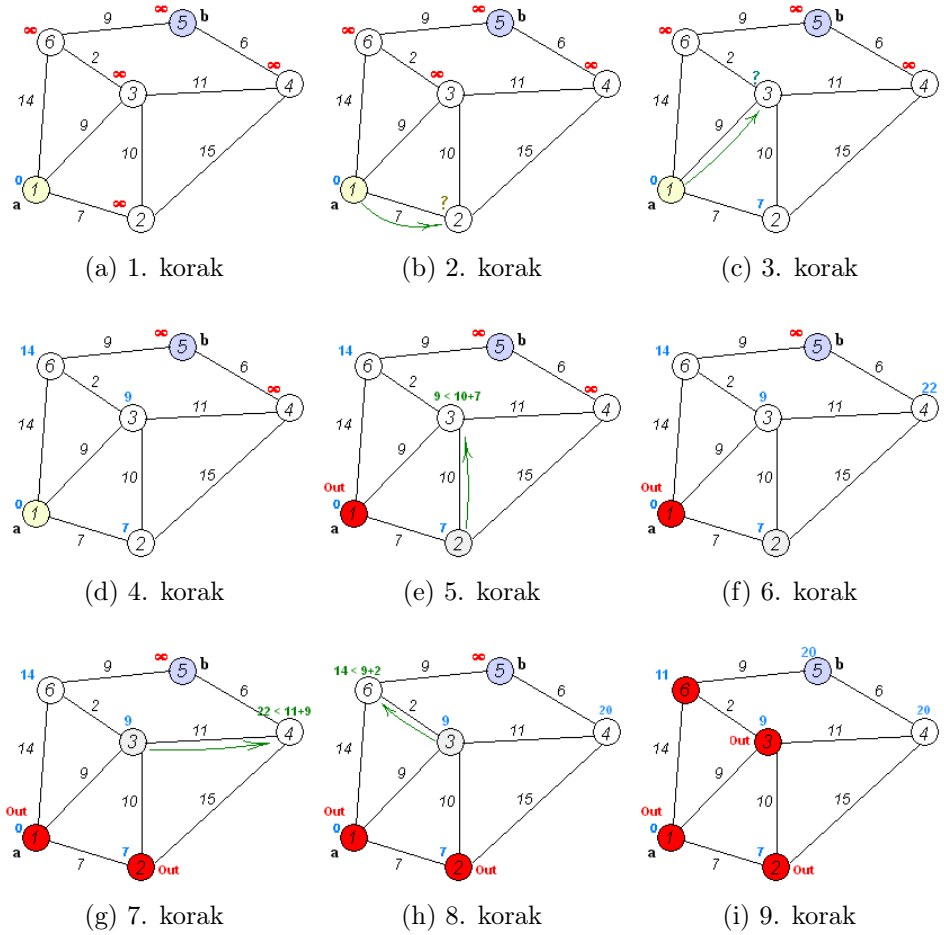
Neka Q predstavlja skup čvorova u grafu, a kako graf još nije obrađen svi čvorovi se nalaze u Q. Slijedeći korak je pretražiti cijeli graf i za svaki čvor odrediti udaljenost. Prvo se mora odrediti čvor s najmanjom udaljenošću, kako svi čvorovi, osim ishodišnog, imaju beskonačnu udaljenost, za prvi čvor uzima se ishodišni (Slika 3.3a).

Čvor u briše se iz skupa Q i pristupa se slijedećem koraku u kojem se za svaki susjedni čvor računa udaljenost ali samo ako se taj čvor nalazi u skupu Q. Ako se taj čvor ne nalazi u Q onda to znači da je algoritam već obradio taj čvor i maknuo ga iz Q. Ukoliko putanja, koja predstavlja novo akumuliranu težinu, ima manju vrijednost od postojeće udaljenosti, udaljenost se izjednačava sa putanjom, a čvor u postaje prethodni čvor. Slike 3.3b, 3.3c i 3.3d prikazuju izračun udaljenosti za čvorove 2, 3 i 6.

Slijedeći korak predstavlja ponovno traženje čvora s najmanjom udaljenosti, a u ovom slučaju to je čvor 2 i izračun putanja za sve susjedne čvorove koji se nalaze u skupu Q. Na slici 3.3e može se vidjeti da akumulirana putanja ima veći iznos od postojeće $9 < 10 + 7$ pa se ta mogućnost odbacuje. Algoritam će još obraditi čvor 4 izračunavanjem udaljenosti kako je prikazano na

slici 3.3f. Nakon toga izabire se čvor 3 i računa se akumulirana putanja prema čvoru 4 koja iznosi $22 < 11 + 9$ i koja je manja od 22 pa se uzima kao najkraća udaljenost (Slika 3.3g). Analogno tome obrađuje se i čvor 6 kako je vidljivo na slici 3.3h.

U posljednjim koracima obrađuju se čvorovi 5 i 4 ali kako oba čvora imaju akumuliranu udaljenost 20 algoritam završava (Slika 3.3i).



(a) 1. korak (b) 2. korak (c) 3. korak

(d) 4. korak (e) 5. korak (f) 6. korak

(g) 7. korak (h) 8. korak (i) 9. korak

Slika 3.3: Prikaz algoritma Dijkstra

Nakon završetka algoritma *prethodni* će imati strukturu koja zapravo opisuje graf koji je podgraf orginalnome grafu, ali mu nedostaju neki bridovi. Traženje stvarnog najkraćeg puta između ishodišnog i bilo kojeg drugog čvora u grafu može se napraviti upotrebom neke od metoda pretraživanja, koje su navedene u poglavlju 3.1, npr. dubinsko pretraživanje.

Efikasnost algoritma Dijkstra je $O(n^2)$ gdje n predstavlja broj čvorova. Iako su određene implementacije Dijkstra algoritma i efikasnije, recimo isko-

rištavanje binarnih stabala, one su moguće samo za uska područja problema (raspršeni grafovi).

3.2.2 Algoritam Bellman-Ford

Bellman-Ford algoritam traži najkraći put za jedan početni čvor u težinskome usmjerenom grafu. Međutim, za razliku od Dijkstrinog algoritma graf smije imati negativne težine. Algoritam ima lošije performanse od algoritma Dijsktra, ali ponekad je nemoguće modelirati neki problem bez korištenja negativnih težina.

Ovaj algoritam je u osnovi vrlo sličan Dijkstrinom algoritmu, ali umjesto odabira čvora s najmanjom udaljenošću za slijedeći čvor, algoritam pristupa svim bridovima. Ta operacija se izvodi $v - 1$ puta, gdje je v broj čvorova u grafu. Ta ponavljanja omogućuju precizno propagiranje udaljenosti kroz graf, prvenstveno zbog prirode najkraće putanje koja svaki čvor može posjetiti samo jednom.

Efikasnost Bellman-Ford algoritma je $O(v*e)$ gdje v označava broj čvorova, a e broj bridova u grafu.

Algoritam se prvenstveno koristi u mrežnim usmjernicima kako bi se odredio optimalni put mrežnih paketa u nekoj računalnoj mreži.

3.2.3 Algoritam A*

Algoritam A* (eng. a star) se koristi u slučajevima kada je potrebno pronaći najkraći put između jednog ishodišnog (početnog) čvora i jednog završnog (ciljnog) čvora. Temelji se na *najbolji prvi* metodi pretraživanja opisanoj u poglavlju 3.1.3. Ovaj algoritam 1968. godine opisali su Peter Hart, Nils Nilsson, i Bertram Raphael u radu *A Formal Basis for the Heuristic Determination of Minimum Cost Paths* [Hart et al., 1968].

Algoritam funkcionira tako da se za neki čvor prilikom odabira slijedećeg potencijalnoga čvora u obzir uzimaju dvije stvari: trenutnu udaljenost za taj čvor $g(x)$ i predviđenu udaljenost od trenutnog čvora do završnog $h(x)$. Sumom te dvije funkcije, dobit će se funkcija $f(x) = g(x)+h(x)$ čiji rezultat predstavlja težine slijedećih potencijalnih čvorova. Vidljivo je da će se prvo odabrati čvor s najmanjom potencijalnom težinom.

Funkcija $g(x)$ predstavlja akumuliranu udaljenost za trenutni čvor, a funkcija $h(x)$ je tzv. heuristika, odnosno ona služi za određivanje predviđene udaljenosti između trenutnog čvora i završnog čvora. Ukoliko imamo cestovnu mrežu, u kojoj svaki čvor ima definirane koordinate, funkcija $h(x)$ može biti jednostavno određivanje Euklidske udaljenosti. Za funkciju $h(x)$ bitno je da

predviđena udaljenost uvijek bude manja od stvarne udaljenosti. Ukoliko nije moguće definirati funkciju $h(x)$, A* algoritam se ne može koristiti.

A* započinje na odabranom čvoru, koji ima udaljenost 0. Algoritam tada pretpostavlja, odnosno izračunava pomoću heuristike, udaljenost od trenutnog do završnog čvora. Svaki čvor s izračunatom heuristikom se sprema u tzv. prioritetni red koji sadrži čvorove poredane uzlazno po iznosu udaljenosti. U slijedećem koraku se iz prioritetnog reda uzima prvi čvor, odnosno čvor s najmanjom udaljenošću. Ukoliko taj čvor nije ciljni čvor, algoritam računa udaljenost svih susjednih čvorova i sprema ih u prioritetni red.

Osim prioritetnog reda algoritam sprema listu već posjećenih čvorova koja se koristi u svrhu optimizacije. Lista posjećenih čvorova funkcionira tako da u slučaju kad je za trenutni čvor izračunata udaljenost veća od udaljenosti koja se nalazi na listi, taj čvor i sve njegove putanje se odbacuju. U suprotnom slučaju, ako je izračunata udaljenost manja od one na listi, trenutni čvor prepisuje onog na listi, a procesiranje se nastavlja od tog čvora.

Efikasnost A* algoritma identična je efikasnosti Dijkstra algoritma i iznosi $O(n^2)$ gdje n predstavlja broj čvorova. Međutim, u praksi A* algoritam pruža osjetna poboljšanja i ubrzanja u odnosu na klasični Dijkstra algoritam.

3.2.4 Algoritam D*

Algoritam D* (eng. D star), poznat kao i algoritam Stenz, prvi je osmislio Anthony Stentz, 1994. Zbog činjenice da A* algoritam daje dobre rezultate samo ako je graf u potpunosti poznat, ne može se iskoristiti za grafove u kojima se, recimo, dinamički mijenjaju težine. Algoritam D* se još naziva dinamičkom verzijom povratnog algoritma Dijkstra i A* algoritma. Najčešća primjena D* algoritma je u navigacijskim sustavima vozila.

Ovaj problem se može riješiti ponovnim izračunom kompletne putanje korištenjem A* algoritma, od trenutne lokacije do krajnjeg čvora, onda kad se pojave nove informacije. Iako je ovo optimalno rješenje, ovakav pristup nije učinkovit. Algoritam D* uključuje ovakve dinamične promjene u izračun na optimalan i učinkovit način tako da se izračunava samo dio tražene putanje.

D* algoritam suprotno od A* algoritma započinje od cilja i kreće prema početnom čvoru. To donosi prednost jer je svaki posjećeni čvor povezan s čvorom koji je dio putanje prema završnom čvoru. Ovaj algoritam će završiti onog trenutka kad dođe do početnog (ishodišnog) čvora.

3.2.5 Algoritam Floyd–Warshall

Floyd-Warshall algoritam pronalazi sve moguće putanje u grafu između svakog para čvorova. Algoritam je 1959. godine osmislio francuski znanstvenik Bernard Roy. Ovaj algoritam može se iskoristiti za težinske usmjerene grafove, ali samo s pozitivnim težinama.

Na graf G sa čvorovima v, primjenjuje se funkcija koja traži najkraći put između čvorova i i j, ali tako da se koriste samo čvorovi $1..k$ kao međučvorovi. Postoje dva moguća rješenja ovog problema, ili će najkraći put sadržavati samo čvorove iz skupa $1..k$ ili postoji neka putanja koja ide od i do $k+1$ čvora pa do j koji je možda kraći. Ukoliko je drugim rješenjem dobiven kraći put, konačno rješenje će nastati stapanjem putanja $i..k+1$ i $k+1..j$.

Prema tome funkcija za traženje najkraćeg puta može se definirati pomoću rekurzivnog izraza:

```
duljina(i,j,k) =
    min(duljina(i,j,k-1), duljina(i,k,k-1) + duljina(k,j,k-1))
duljina(i,j,0) = tezina(i,j)
```

Algoritam funkcionira tako da prvo traži najkraći put, odnosno funkciju $duljina(i, j, 1)$, za sve (i, j) parove čvorova, gdje 1 označava broj čvorova u skupu koji se koristi za traženje najkraćeg puta. Slijedeći korak je izvršavanje funkcije $duljina(i, j, 2)$ za sve (i, j) parove. Algoritam završava u trenutku $k = n$, odnosno kada su pronađene sve najkraće puteve za sve (i, j) parove čvorova.

Efikasnost Floyd-Warshall algoritma iznosi $O(n^3)$ gdje je n broj čvorova u grafu. Algoritam se najčešće koristi u svrhu pronalaženja najkraćeg ili optimalnog puta između dva čvora.

3.2.6 Algoritam Johnson

Johnsononv algoritam će pronaći sve najkraće putanje između svih parova čvorova u raštrkanom usmjerenom grafu koji može imati negativne težine. U implementaciji se koristi Bellman-Ford algoritam kako bi se dobio graf bez negativnih težina, nakon čega na takvom grafu primjenjuje Dijsktrin algoritam. Algoritam se naziva prema Donaldu B. Johnsonu, koji ga je prvi objavio 1977. godine.

Algoritam se sastoji od slijedećih koraka:

1. novi čvor q se dodaje u graf, povezan s nultim težinama sa svakim drugim čvorom

2. koristi se Bellman-Ford algoritam, s ishodišnim čvorom q kako bi se za svaki čvoru u grafu v odredio najkraći put od q do v

3. za svaki čvor u orginalnom grafu postavlja se nova težina izračunata u prethodnom koraku

4. koristi se algoritam Dijkstra kako bi se pronašao najkraći put od ishodišnog čvora s za svaki čvor u novom grafu

Efikasnost Johnson algoritma je $O(n^2 * logn + N * E)$, gdje je n broj čvorova, a E broj bridova, što je povoljnije od Floyd-Warshall čija je efikasnost $O(n^3)$. Ovo je nažalost moguće očekivati samo u raštrkanim grafovima.

3.3 Problem trgovačkog putnika

Problem trgovačkog putnika (eng. Travelling Salesman problem (TSP)) je zaseban matematički problem koji datira iz 1930. godine, i jedan je od najintenzivnije proučavanih problema. Problem je najlakše predočiti zamislimo li skup gradova kojih je potrebno posjetiti na najkraći mogući način tako da se svaki grad posjeti samo jednom. Slika 3.4 prikazuje rješenje kako posjetiti 14 najvećih gradova u Njemačkoj.

Ovaj problem se može primijeniti na težinski graf. Rješenje problema će biti *Hamiltonova putanja*, koja svaki čvor u grafu posjećuje samo jednom. Uzme li se u obzir da graf nije usmjeren, odnosno da je put neovisan o tome da li se ide od čvora a do b ili od čvora b do a, broj mogućih rješenja raste eksponencijalno. Formula po kojoj se može izračunati broj kombinacija je slijedeća:

$$\frac{(n-1)!}{2}$$

Dijeli se s 2 zbog toga što je težina neovisna o smjeru. Primjerice za 10 čvorova, broj kombinacija iznosi 181440, za 20 čvorova 60822550204416000, dok npr. za 100 čvorova broj kombinacija raste na broj s 156 znamenki, a za 1000 čvorova broj s 2565 znamenki. Zanimljivo je, da iako postoji mnogo upotrebljivih heuristika i da su poznate egzaktne metode rješavanja, nije ih moguće generalno primijeniti. Uglavnom se danas govori o optimalnim rješenjima, odnosno rješenjima koja je moguće pronaći u zadovoljavajućem vremenskom roku, a nalaze se unutar 1% od najboljeg mogućeg rješenja.

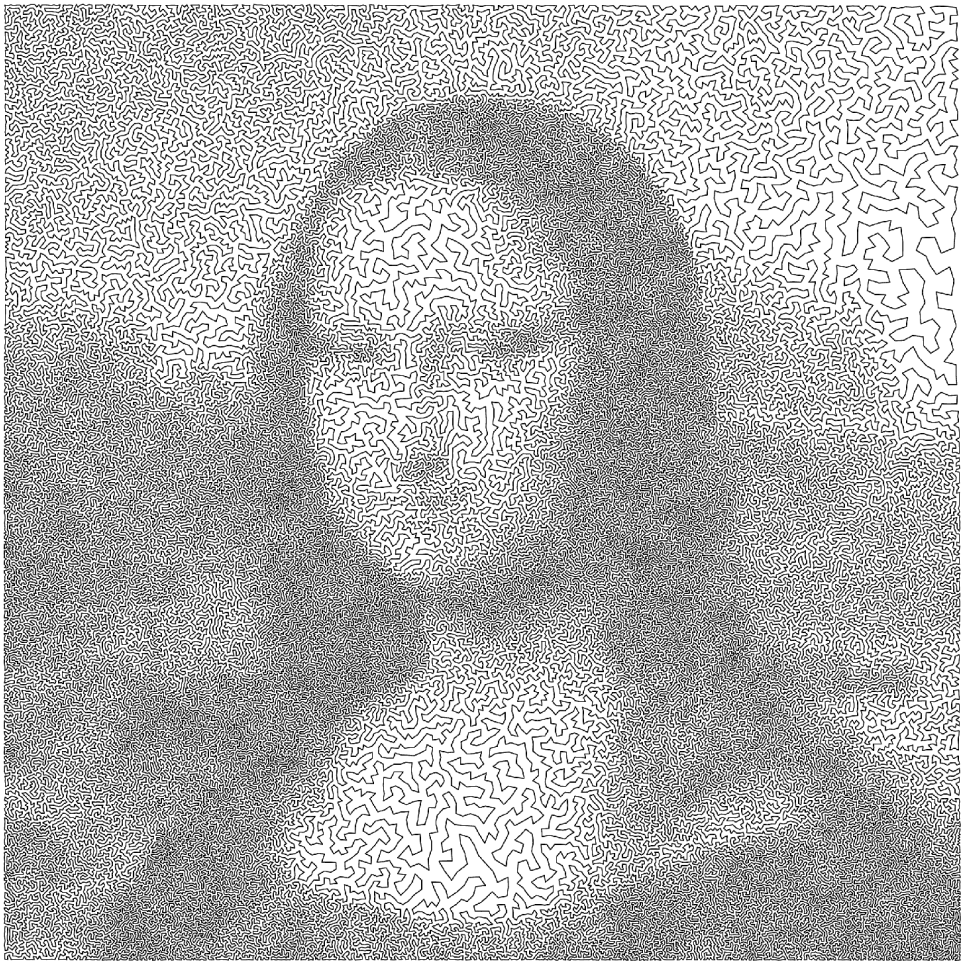
Problem trgovačkog putnika primjenjuje se za planiranje, logistiku ili prilikom izrade mikročipova. Koncept grada postaje kupac, DNA fragment ili mjesto za lemljenje čipa, a koncept udaljenosti postaje vrijeme putovanja ili

Slika 3.4: Problem trgovačkog putnika

mjera sličnosti između DNA fragmenata. Također postoje ponekad posebni uvjeti koji dodatno otežavaju problem.

Zbog ovakvih karakteristika problem trgovačkog putnika svrstava se u probleme koji se nazivaju NP-potpuni problemi. Za te probleme se vrlo jednostavno i brzo (polinomno vrijeme) može provjeriti ispravnost rješenja, ali je vrlo teško pronaći takvo rješenje. Sve NP-potpune probleme je moguće riješiti u eksponencijalnom vremenu, ali nitko nije dokazao da ih je nemoguće riješiti u polinomnom vremenu. Također se pretpostavlja, da se rješenje za jedan NP-potpuni problem može primijeniti na sve ostale NP-potpune probleme.

Zanimljiva je primjena problema trgovačkog putnika na prikazivanje dvobojnih grafika. Slične metode koriste se svakodnevno prilikom pripreme tiska dnevnih novina. Rješavanje se svodi na prikazivanje tamnijih područja grafike gušćim uzorkom točaka, a svjetlijih područja raštrkanim uzorkom točaka. Pravilnim rasporedom točaka i postavljanjem težina može se primijeniti i metoda rješavanja problema trgovačkog putnika. Slika 3.5 prikazuje portret Mona Lise, nakon rješavanja problema, a izgrađen je od 100000 točaka. Zanimljivo je primijetiti da je u topološkom smislu ta grafika prikazana pomoću kruga.



Slika 3.5: Problem trgovačkog putnika - Mona Lisa

Poglavlje 4

Specifikacija softverskog stoga

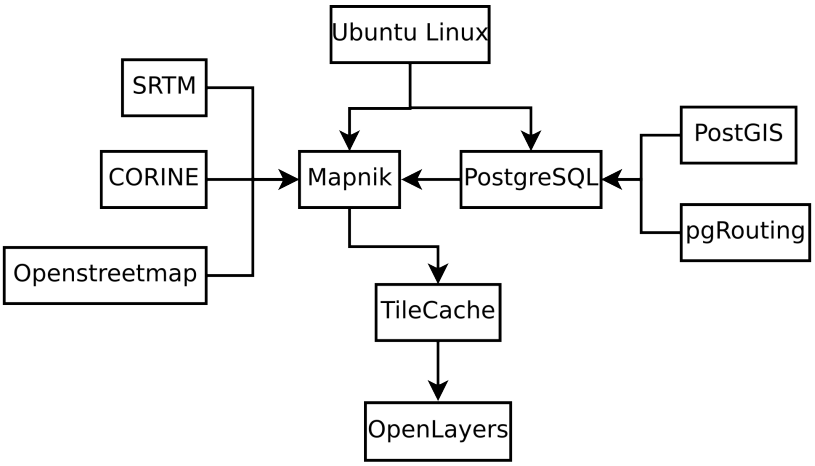
Osnovna ideja praktičnog primjera je pronaći i prikazati optimalnu putanju između dvije točke koje odabere korisnik. Interakcija i vizualizacija podataka omogućena je kroz bilo koji noviji internet preglednik. Naime, osnovna ideja je napraviti Web servis koji će to omogućiti.

Realizacija praktičnog primjera je u potpunosti napravljena korištenjem slobodnog softvera. U pozadini rješenja nalazi se Ubuntu Linux distribucija, inačice 8.04, u poslužiteljskoj inkarnaciji. Osim standardnog Apache Web poslužitelja, potrebno je osigurati i infrastrukturu za prostorne podatke. Ta infrastruktura je izgrađena oko sustava za upravljanje bazama podataka PostgreSQL, koji je proširen PostGIS prostornom nadogradnjom i pgRouting funkcijama za izračun optimalne putanje.

Za vizualizaciju podataka koristi se Mapnik koji može pročitati i iskoristiti različite tipove podatka. Korisnik koristi OpenLayers za interakciju sa servisom, a prazninu između Mapnika i OpenLayersa popunjava TileCache servis koji implementira WMS-C specifikaciju, preko koje OpenLayers pristupa pojedinom dijelu rasterskog mozaika.

U ovom diplomskom radu koriste se samo slobodni prostorni podaci. Temeljni sloj čine SRTM podaci o visinama (odjeljak 4.3.1), iz kojih se mogu izvući slojnice i iscrtati reljef. Pokrov se prikazuje iz Corine (odjeljak 4.3.2) podataka, koji su dostupni za Europu. Zadnji sloj prikazuje ceste i objekte koji su prikupljeni kroz Openstreetmap projekt (odjeljak 4.3.3).

U slijedećim odjeljcima ukratko će se opisati iskorišteni softver, podaci i metode integracije. Grafički dijagram softverskog stoga i iskorištenih podataka može se vidjeti na slici 4.1.



Slika 4.1: Softverski stog i iskorišteni podaci

4.1 Slobodni softver

Slobodni softver je softver koji se može koristiti, proučavati i izmjenjivati bez ograničenja. Također, softver se može kopirati i distribuirati u izmijenjenom ili izvornom obliku bez ograničenja. Jedino ograničenje je osiguranje da budući korisnici naslijede ta ista prava. Slobodni softver je uglavnom besplatan, iako nema nikakvih ograničenja oko naplaćivanja usluge distribucije.

Kako bi se softver mogao distribuirati kao slobodni softver, mora uključivati izvorni kôd programa i informaciju o pravima koja se dodjeljuju korisniku. Informacija o pravima je najčešće licenca koja je u skladu sa definicijom slobodnog softvera ili obavijest o tome da softver pripada javnom dobru.

Pokret slobodnog softvera je 1983. godine osmislio Richard Stallman kako bi zadovoljio potrebu za istim od svih korisnika računala. Dvije godine kasnije 1985. godine osnovana je Free Software Foundation[1] koja pruža organizacijsku strukturu i koja se bavi promocijom slobodnog softvera. Definicija slobodnog softvera korisniku daje četiri temeljne slobode:

0 sloboda pokretanja programa u bilo koju svrhu

1 sloboda proučavanja i modificiranja programa

2 sloboda kopiranja programa (distribucije)

3 sloboda unapređenja programa i distribucije tih izmjena, kako bi cijela zajednica imala koristi

[1] `http://fsf.org`

Na ovim slobodama temelje se licence slobodnog softvera koje provjerava i odobrava FSF ili OSI[2]. Iako se te dvije organizacije razlikuju (FSF promiče slobodni softver, OSI promiče otvoreni kôd), licence koje su odobrene su u potpunosti sukladne.

Postoji nekoliko uobičajenih zabluda koje ograničavaju prihvaćanje slobodnog softvera u poslovnom svijetu. Prvu zabludu predstavljaju licence koje se smatraju virusnim, zbog toga što je licenca identična za sva derivirana djela. Drugi razlog je nedostatak profesionalne podrške i edukacije, a upravo podrška i edukacija temelje jedan od poslovnih modela koji se primjenjuju na slobodni softver. Kao treći razlog navode se brzina promjena i nedostatak definiranih ciljeva budućeg razvoja. Ciljevi dugoročnog razvoja postoje, za primjer se mogu uzeti softverski paketi Mozilla Firefox i OpenOffice.org. Brzina promjena apsolutno se nikad ne može navoditi kao nedostatak, uvijek se može nastaviti koristiti ista inačica programa.

Oko projekta slobodnog softvera stvorila se kultura koja promiče slobodnu razmjenu svih znanja i sadržaja. Popularizacijom brzog pristupa internetu, razmjena više ne ovisi o tradicionalnim kanalima distribucije. Internet, kao globalna mreža, osim što potiče razmjenu znanja, postaje temeljem te razmjene. Također unutar takve kulture svi imaju jednaka prava i mogućnost slobodnog sudjelovanja, što omogućava veću mobilnost i agilnost same kulture. Creative Commons[3] je neprofitna organizacija koja pruža nekoliko licenci koje se mogu iskoristiti za licenciranje vlastitih djela.

4.2 Softverski stog

Softverskim stogom se naziva skup softvera koji čini jednu cjelinu. Kako bi se cjelina mogla ostvariti, nužno je osigurati softversku interoperabilnost. Jedna od glavnih prednosti slobodnog softvera je njegova interoperabilnost, prvenstveno zbog toga što se pridržava otvorenih standarda. Ovaj diplomski rad temelji se isključivo na slobodnom softveru.

4.2.1 Ubuntu Linux

Operacijski sustav Ubuntu Linux je distribucija slobodnog softvera koja se temelji na Linux kernelu. Razvoj Ubuntua započinje u travnju 2004. godine odvajanjem od Debian projekta. Mark Shuttleworth okuplja manju grupu ljudi koji razvijaju kôd koji 20. listopada 2004. izdaju prvu inačicu Ubuntu

[2] `http://opensource.org`
[3] `http://creativecommons.org/`

Linux operacijskoga sustava, kodnog imena "Warty Warthog". Naziv Ubuntu dolazi iz Zulu i Xhosa jezika što znači "humanost prema drugima" opisujući osnovnu filozofiju "ja sam to što jesam zbog toga što smo svi mi zajedno".

Razvoj Ubuntua financira kompanija, Canonical Ltd. koju je osnovao Mark Shuttleworth. Ubuntu se trudi postati jednostavan i pristupačan operacijski sustav za sve korisnike. Najveća značajka Ubuntu projekta je konstantna promocija slobodnog softvera i stvaranje zajednice korisnika oko samog projekta.

Ubuntu se prvenstveno razvijao kao operacijski sustav za stolna računala, ali ubrzo se pojavila i inačica za poslužitelje. Specifičnost Ubuntua je šestomjesečni ciklus izdavanja novih inačica, koje se dobro poklapaju s trenutkom izdavanja Gnome radnog okružja. Ključni trenutak se dogodio 2006. godine kad se pojavio Ubuntu 6.06 Dapper Drake LTS , (eng. LongTermSupport), koji ima produženi period službene podrške u trajanju od 3 godine za stolna računala i 5 godina za poslužitelje. Slijedeći LTS se pojavio 2008. godine, inačica 8.04 Hardy Heron. Danas je aktualna inačica 9.04 Jaunty Jackalope koja je izašla 23. travnja 2009. godine.

Za operacijski sustav odabran je Ubuntu 8.04.2 u poslužiteljskoj inačici. Iako se mogla odabrati bilo koja druga Linux distribucija, odabran je Ubuntu zbog jednostavnosti održavanja i velikog broja podržanih softverskih paketa. Instalacijska procedura i konfiguracija operacijskoga sustava neće se opisivati.

Nakon instalacije sustav je proširen softverskim paketima kao što su:

- paketi osnovnog skupa za razvoj softvera bez kojeg nije moguće napraviti niti jedan softver iz izvornog kôda
- razni paketi koji zadovoljavaju međuovisnosti potrebne za izradu sofvera iz izvornog kôda
- softverski paket Apache Web poslužitelj, koji omogućuje izvršavanje Web servisa

Više informacija o ostalim softverskim paketima može se naći u zasebnim odjeljcima koji slijede.

4.2.2 PostgreSQL

PostgreSQL je sustav za upravljanje objektno-relacijskim bazama podataka, razvijen na Kalifornijskom Sveučilištu Berkeley.

Početak razvoja PostgreSQL-a započinje 1986. godine pod nazivom *POSTGRES*, koji sponzorira Agencije za napredna obrambena istraživanja (DARPA[4]),

[4]Advanced Research Project Agency

Ureda za vojna istraživanja (ARO[5]) i znanstvene udruge (NSF)[6]. Zbog velikog broja korisnika, održavanje kôda uzimalo je previše vremena koje je trebalo biti utrošeno na razvijanje same baze podataka. Zbog toga POSTGRES službeno prestaje sa razvojem 1993. godine, sa finalnom verzijom 4.2. U tom je razdoblju POSTGRES korišten u svrhu mnogih znanstvenih istraživanja.

Razvoj POSTGRES-a nastavlja se 1994. godine kao slobodni softver pod BSD licencom, s novim nazivom *Postgres95*. Najveća novost u odnosu na POSTGRES bilo je uvođenje SQL[7] jezika umjesto dotadašnjeg PostQUEL-a. Novi naziv *PostgreSQL* odabran je krajem 1996. godine, jer naziv Postgres95 loše predstavlja dinamičnu razvojnu okolinu projekta.

Od tada pa do danas PostgreSQL se razvio u najnapredniji sustav za upravljanje bazama podataka temeljen na slobodnom kôdu, koji se sa svojom stabilnošću mjeri sa vodećim komercijalnim sustavima za upravljanje bazama podataka. U trenutku pisanja ovog diplomskog rada zadnja ažurna inačica PostgreSQLa je 8.3.7, dok se nestrpljivo iščekuje inačica 8.4 koja bi trebala uskoro ugledati svijetlo dana.

PostgreSQL je odabran zbog mogućnosti jednostavne nadogradnje prostornom komponentom PostGIS (odjeljak 4.2.3) te podrškom za određivanje optimalne putanje pgRouting (odjeljak 4.2.4).

Instalacija PostgreSQL softverskog paketa i njegovih međuovisnosti izvodi se upotrebom standardnih Ubuntu alata za instalaciju:

```
apt-get install postgresql-8.3
```

Tokom instalacije automatski se stvara korisnik administrator "postgres" koji upravlja svim postavkama PostgreSQL sustava za upravljanje bazama podataka. Osim korisnika na sustavu inicijalizira se PostgreSQL i postavljaju skripte koje služe za pravilno podizanje i spuštanje istog. Administracija sustava se svodi na uređivanje konfiguracijskih datoteka i stvaranje korisnika i manipuliranje njihovim pravima. Za ovu svrhu ostavljeni su osnovni konfiguracijski parametri.

4.2.3 PostGIS

PostGIS je slobodni softver koji proširuje PostgreSQL sustav za upravljanje bazama podataka geometrijskim operacijama i tipovima podataka. PostGIS je u skladu sa "Simple Features for SQL" specifikacijom predloženom od "Open Geospatial Consortiuma". PostGIS sadrži slijedeće komponente:

[5]Army Research Office
[6]National Science Foundation
[7]Structured Query Language

- geometrijske tipove za definiciju točaka, linija, poligona, multitočaka, multilinija, multipoligona i geometrijskih kolekcija
- prostorne izraze za određivanje interakcija između geometrija korištenjem $3x3$ Egenhoferove matrice
- prostorne operatore za određivanje prostornih mjera kao što su površina, dužina i opseg
- prostorne operatore za određivanje prostornih operacija na skupovima kao što su unija, razlika i tampon zona prostora[8]
- prostorno indeksiranje podataka (R-stablo + GiST) koji poboljšavaju upite nad prostornim podacima

Prva inačica je objavljena 2001. godine od strane Refractions Research kompanije pod GNU GPL licencom. Ključan trenutak dogodio se 2006. godine kada je PostGIS dobio certifikat od Open Geospatial Consortiuma da je potpuno u skladu sa "Simple Features for SQL" specifikacijom.

Instalcija PostGISa

Instalacija softverskog paketa PostGIS obavlja se na standardan način izvršavanjem naredbe:

```
apt-get install postgresql-8.3-postgis
```

Osim PostGISa instaliraju se i dva dodatna paketa:

GEOS (Geometry Engine - Open Source) je reinkarnacija Java Topology Suita (JTS) u C++ progamskom jeziku. GEOS uključuje sve prostorne izraze i operatore koje definira OpenGIS Simple Features for SQL specifikacija kao i napredne topološke funkcije koje pruža JTS.

proj4 je biblioteka koja omogućuje transformacije između različitih prostornih sustava i kartografskih projekcija ovisno o zadanim parametrima

Nakon instalacije softverskih paketa potrebno je napraviti predložak baze podataka koji će se iskoristiti za stvaranje ostalih PostGIS baza podataka. Ovaj korak nije nužan, ali je poželjan jer uvelike smanjuje vrijeme stvaranja nove PostGIS baze podataka. Stvaranje predloška PostGIS baze podataka sastoji se od slijedećih koraka:

[8] eng. buffer

1. stvaranja prazne baze podataka:

   ```
   createdb postgis_template
   ```

2. dodavanja proceduralnog jezika *plpgsql* u bazu (proceduralni jezik je nužan zbog interakcije s vanjskom PostGIS bibliotekom)

   ```
   createlang plpgsql postgis_template
   ```

3. punjenja baze podataka PostGIS objektima:

   ```
   psql -f /usr/share/postgresql-8.3-postgis/lwpostgis.sql\
    -d postgis_template
   ```

4. punjenja baze podataka definicijama prostornih sustava:

   ```
   psql -f /usr/share/postgresql-8.3-postgis/spatial_ref\
   _sys.sql -d postgis_template
   ```

Ovim postupkom stvorena je nova baza podataka *postgis_ template* koja će poslužiti kao predložak za sve ostale baze podataka koje će imati PostGIS funkcionalnost. Nova baza podataka se stvara naredbom:

```
createdb -T postgis_template ime_nove_baze
```

4.2.4 pgRouting

pgRouting je projekt koji pruža funkcionalnost određivanja najkraćeg puta za PostGIS bazu podataka. pgRouting je dio PostLBSa koji je osnovica za LBS (lokacijske servise) temeljene na slobodnom softveru.

pgRouting je nastavak razvoja pgDijsktra softverskog paketa koji razvija tvrtka Camptocamp. Razvojem pgRouting softverskog paketa bavi se tvrtka Orkney iz Japana koja pruža i komercijalnu podršku.

Softverski paket pgRouting implementira slijedeće funkcionalnosti određivanje najkraćeg puta:

Dijkstra osnovni algoritam za određivanje najkraćeg puta (odjeljak 3.2.1)

A* algoritam koristi heuristiku kako bi odredio najkraći put, opisan u poglavlju 3.2.3

Shooting* algoritam ponešto jenapredniji jer tijekom izračuna uzima u obzir listu bridova koji su iskoristivi samo ako je zadovoljena određena težina, (primjerice to mogu biti ograničenja skretanja na raskršćima)

Problem trgovačkog putnika moguće je riješiti upotrebom dodatne biblioteke, GAUL Genetic Algorithm Utility Library

Udaljenost putovanja računa se udaljenost po pojedinom bridu grafa, bitno je napomenuti da su ove udaljenosti neovisne o projekciji, potrebna je dodatna biblioteka CGAL Computational Geometry Algorithms Library

Više o svakoj pojedinoj funkcionalnosti i načinu na koji se može iskoristiti bit će zapisano u slijedećem odjeljku 4.2.4.

Funkcionalnosti

Osnovna funkcionalnost, odnosno algoritam koji pgRouting podržava je algoritam Dijkstra. Funkcija *shortest_path* prima slijedeće argumente:

sql tekst je zapravo upit koji mora imati slijedeće atribute, a rezultat tog upita su podaci koji se koriste u algoritmu

- **id** jedinstveni identifikator brida
- **source** jedinstveni identifikator početnog čvora brida
- **target** jedinstveni identifikator krajnjeg čvora brida
- **cost** težina prolaska kroz taj brid, ukoliko je negativna čvor se neće uključiti u graf
- **reverse_cost** težina prolaska kroz brid u suprotnom smjeru, (ovaj atribut je neobavezan)

source_id integer jedinstveni identifikator ishodišnog čvora

target_id integer jedinstveni identifikator ciljnog čvora

directed boolean definira usmjerenost grafa

has_reverse_cost boolean definira da li graf ima težine prolaska u suprotnom smjeru

Ta funkcija će za rezultat vratiti skup redova, u kojem svaki red predstavlja brid koji je dio rješenja. Kolone svakog reda su:

vertex_id identifikator početnog čvora za svaki brid

edge_id identifikator brida koji je dio najkraće putanje

cost težina brida

Na kraju skupa nalazi se zapis sa identifikatorom ciljnog čvora koji ima težinu 0, što omogućava izračun ukupne težine putanje jednostavnim sumiranjem svih težina skupa.

Slijedeći implementiran algoritam je A*, koji se koristi pozivom funkcije *shortest_path_astar* sa slijedećim argumentima:

sql tekst predstavlja upit s podacima koje koristi algoritam

id jedinstveni identifikator brida

source jedinstveni identifikator početnog čvora brida

target jedinstveni identifikator krajnjeg čvora brida

cost težina prolaska kroz taj brid, ukoliko je negativna čvor se neće uključiti u graf

x1 x koordinata početnog čvora brida

y1 y koordinata početnog čvora brida

x2 x koordinata krajnjeg čvora brida

y2 y koordinata krajnjeg čvora brida

reverse_cost težina prolaska kroz brid u suprotnom smjeru, (ovaj atribut je neobavezan)

source_id integer jedinstveni identifikator ishodišnog čvora

target_id integer jedinstveni identifikator ciljnog čvora

directed boolean definira usmjerenost grafa

has_reverse_cost boolean definira da li graf ima težine prolaska u suprotnom smjeru

Isto kao i prethodna funkcija, shortest_path za rezultat vratit će skup redova, u kojem svaki red predstavlja brid koji je ujedno idio rješenja. Kolone svakog reda su:

vertex_id identifikator početnog čvora za svaki brid

edge_id identifikator brida koji je dio najkraće putanje

cost težina brida

Algoritam Shooting* *shortest_path_shooting_star* implementiran je kroz funkciju koja ima argumente slične kao i prethodna `shortest_path_astar` funkcija. Jedina razlika je u tome što SQL upit čiji se rezultat koristi za izračun optimalne putanje ima dodatna dva atributa:

rule tekstualni zapis jedinstvenih identifikatora (id) bridova, odvojenih zarezom, koji opisuju pravila ograničavajući upotrebu tih bridova, (brid se može iskoristiti samo ako je težina jednaka onoj definiranoj atributom `to_cost`)

to_cost težina ograničenog prolaska

Problem trgovačkog putnika rješava funkcija *tsp* koja se temelji na genetičkom algoritmu. Iako ovaj pristup neće pružiti egzaktno rješenje, sigurno će pružiti optimalno rješenje nakon nekoliko iteracija. Rješenje problema trgovačkog putnika temelji se na poretku čvorova upotrebom ravninske udaljenosti (Euklidska udaljenost 2.2.1) između čvorova. Funkcija `tsp` ima slijedeće argumente:

sql tekst upit čiji se rezultat koristi u algoritmu

- **source_id** jedinstveni identifikator čvora
- **x** x koordinata čvora
- **y** y koordinata čvora

ids skup jedinstvenih identifikatora čvorova za koje se rješava problem trgovačkog putnika

source_id jedinstveni identifikator ishodišnog čvora

Rezultat funkcije će biti popis čvorova poredanih u skladu s definicijom problema trgovačkog putnika.

Za izračun udaljenosti putovanja koristi se funkcija *driving_distance*. Kao i većina do sada opisanih funkcija koristi već opisane argumente. Jedina razlika je dodatni parametar:

distance realni broj definiran u jedinici težine brida

Funkcija će vratiti skup redova koji sadržavaju slijedeće atribute:

vertex_id jedinstveni identifikator početnog čvora svakog brida

edge_id jedinstveni identifikator iskorištenog brida

cost težina koja pripada bridu

Posljednji red skupa ima zapis sa identifikatorom ciljnog čvora koji ima težinu 0, koja omogućuje izračun ukupne težine putanje jednostavnim sumiranjem svih težina skupa.

Instalacija paketa

Prije nego što se instalira softverski paket pgRouting, potrebno je instalirati softverske pakete o kojima ovisi kao i biblioteke GAUL i CGAL. Svi potrebni paketi i njihove međuovisnosti instaliraju se naredbom:

```
apt-get install build-essential subversion cmake \
libboost-graph* postgresql-server-dev-8.3
```

Instalacija CGAL biblioteke, koju koristi algoritam za izračun udaljenosti, provodi se na slijedeći način:

```
apt-get install libcgal*
```

Instalacija GAUL biblioteke je nešto kompliciranija, jer zahtijeva izradu softverskog paketa iz izvornog kôda.

```
wget "http://downloads.sourceforge.net/gaul/gaul-\
devel-0.1849-0.tar.gz?modtime=1114163427&big_mirror=0"

tar xzf gaul-devel-0.1849-0.tar.gz
cd gaul-devel-0.1849-0/
./configure --disable-slang --prefix=/usr
make
make install
```

Prvi korak instalacije je preuzimanje izvornog kôda GAUL biblioteke, nakon čega se izvorni kôd otpakirava. Prije početka izrade potrebno je konfigurirati softverski paket naredbom `configure`. Nakon uspješne konfiguracije izvorni kôd se prevodi u binarni, naredbom `make`. Konačni korak je instalacija prevedenog izvornog kôda naredbom `make install`.

Sada se konačno može preuzeti, prevesti i instalirati softverski paket pgRouting.

```
svn checkout http://pgrouting.postlbs.org/svn/\
pgrouting/trunk pgrouting

cd pgrouting/
```

```
cmake -DWITH_TSP=ON -DWITH_DD=ON .
make
make install
```

Prvo se pgRouting softverski paket preuzima iz Subversion repozitorija kôda. Priprema softverskog paketa se radi na malo drugačiji način pomoću naredbe `cmake`, a ostatak postupka prevođenja je identičan prethodno opisanom.

Integracija s bazom prostornih podataka

Kako bi se pgRouting mogao iskoristiti potrebno je njegovu funkcionalnost uključiti u PostGIS bazu podataka. U tu svrhu iskoristit će se pripremljen predložak `postgis_template` (odjeljak 4.2.3).

Stvara se nova baza podataka `pgrouting_template` koja se temelji na predlošku `postgis_template`.

```
createdb -T postgis_template pgrouting_template
```

Slijedeći korak je punjenje novostvorene baze podataka sa skupovima funkcija:

```
#pgRouting funkcije
psql -f /usr/share/postlbs/routing_core.sql\
 -d pgrouting_template
psql -f /usr/share/postlbs/routing_core_wrappers.sql\
 -d pgrouting_template
psql -f /usr/share/postlbs/routing_topology.sql\
 -d pgrouting_template

#TSP funkcije
psql -f /usr/share/postlbs/routing_tsp.sql\
 -d pgrouting_template
psql -f /usr/share/postlbs/routing_tsp_wrappers.sql\
 -d pgrouting_template

#funkcije udaljenosti putovanja
psql -f /usr/share/postlbs/routing_dd.sql\
 -d pgrouting_template
psql -f /usr/share/postlbs/routing_dd_wrappers.sql\
 -d pgrouting_template
```

Ovime završava instalacija i integracija pgRouting softverskog paketa, a novostvorena baza podataka `pgrouting_template` se može iskoristiti kao predložak za bilo koju drugu bazu podataka.

4.2.5 Mapnik

Mapnik[9] je slobodni programerski alat za razvoj kartografskih aplikacija. Napisan je u C++ programskom jeziku i posjeduje priključke za Python skriptni jezik koji omogućavaju ubrzani razvoj aplikacija. Mapnik se može koristiti za razvoj aplikacija za stolna računala ili Web aplikacija.

Mapnik koristi biblioteku AGG (Anti-Grain Geometry) koja omogućuje visoko kvalitetno iscrtavanje geografskih podataka bez nazubljenosti i sa subpikselnom preciznosti. U potpunosti je napisan u modernom C++ programskom jeziku i koristi opće prihvaćene biblioteke[10] za upravljanje memorijom, datotekama ili uobičajenim programskim zadacima.

Izvori geografskih podataka počinju sa svim OGR i GDAL podržanim rasterskim i vektorskim formatima, ugrađenom podrškom za ESRI Shapefile, PostGIS, Oracle Spatial, TIFF i mnogim drugim izvorima podataka, a jednostavan sustav razvoja priključaka omogućava iskorištavanje bilo kakvih prostornih podataka.

Razvoj Mapnika započinje Artem Pavlenko početkom 2005. godine, trenutno je službena inačica 0.6.0, a oko projekta se okupila velika zajednica programera i korisnika. Najveći projekt koji koristi Mapnik za iscrtavanje kartograskih podloga je Openstreetmap, o kojem će biti više riječi u odjeljku 4.3.3.

Konfiguracija Mapnika, odnosno slojeva podataka, stilova i uvjeta definira se koristeći XML datoteku u kojoj su oni precizno definirani. Moguće je isto postići i direktno u Python skripti, ali uglavnom se koristi samo za manje skupove podataka. Također postoji i C++ integracija, ali ona se uglavnom koristi u slučajevima kada je potrebno integrirati Mapnik u neki drugi softverski paket.

Mapnik koristi slikajući model iscrtavanja, što znači da redoslijed definicije slojeva uvjetuje vidljivost sloja. Odnosno, kasnije definirani slojevi će prekriti ranije definirane slojeve. Svaki sloj podataka koristi jedan ili više stilova. Dočim svaki stil sadrži jedno ili više pravila koje će se iskoristiti ukoliko je zadovoljen uvjet filtra.

Instalacija

Prije same instalacije Mapnik softverskog paketa potrebno je instalirati dodatne softverske pakete o kojima Mapnik ovisi:

```
apt-get install libboost-filesystem* libboost-iostreams*\
```

[9] `http://mapnik.org`
[10] `http://boost.org`

```
 libboost-python* libboost-regex* libboost-serialization*\
 libboost-thread* libltdl3 libltdl3-dev libtiff4\
 libtiff4-dev libtiffxx0c2 python-imaging python-imaging-dbg
```

Ukoliko se želi iskoristiti Cairo programski paket koji omogućuje iscrtavanje u SVG ili PDF formatu zapisa podataka, moraju se instalirati slijedeći softverski paketi:

```
apt-get install libcairo2 libcairo2-dev python-cairo\
 python-cairo-dev libcairomm-1.0-1 libcairomm-1.0-dev\
 libpixman-1-0 libpixman-1-dev
```

Kako bi se Mapnik mogao spojiti na PostGIS bazu podataka ili iskoristiti GDAL i OGR za pristup prostornim podacima, potrebni su slijedeći softverski paketi:

```
apt-get install libgdal-dev python2.5-gdal\
 postgresql-contrib-8.3 gdal-bin
```

Slijedeći paketi su nužni ukoliko se želi omogućiti OGS WMS funkcionalnost Mapnik softverskog paketa:

```
apt-get install libxslt1-dev libxml2-dev python-setuptools
easy_install jonpy
easy_install lxml
```

Instalacija Mapnika započinje preuzimanjem i otpakiravanjem arhive izvornog kôda.

```
wget "http://download.berlios.de/mapnik/mapnik-0.6.0.tar.bz2"
tar jxf mapnik-0.6.0.tar.bz2
cd mapnik-0.6.0
```

Nakon otpakiravanja arhive postupak prevođenja izvornog kôda Mapnik softverskog paketa je slijedeći:

```
python scons/scons.py configure INPUT_PLUGINS=gdal,ogr,\
osm,postgis,raster,shape,sqlite DEBUG=False

python scons/scons.py

python scons/scons.py install
```

Prva naredba konfigurira izvorni kôd prije samog prevođenja koje započinje drugom naredbom. Treća naredba konačno instalira Mapnik i sve njegove datoteke na sustav.

Ovime je instalacija Mapnik softverskog paketa gotova, te je poželjno provjeriti ispravnu prevođenje i instalaciju. Taj postupak je najlakše odraditi slijedećom naredbom, tj. ukoliko se ne pojave greške, može se zaključiti da je Mapnik uspješno instaliran:

```
python -c 'import mapnik'
```

4.2.6 TileCache

TileCache softverski paket je implementacija WMS-C specifikacije koja omogućava pristup već iscrtanim dijelovima rasterskog mozaika. TileCache je napisan u Pythonu i omogućava iskorištavanje različitih mehanizama za skladištenje rastera kao i sustava za iscrtavanje. U elementarnom obliku koristit će zapisivanje na tvrdi disk računala i neki od dostupnih WMS servisa, a korisničke aplikacije će mu pristupati kroz Python CGI sučelje.

Primarna namjena TileCache softverskog paketa je skladištenje bilo kojeg WMS servisa, koje ima svrhu ubrzavanja pristupa. Osim toga TileCache može iskoristiti programske pakete za iscrtavanje kao što su Mapnik (odjeljak 4.2.5) ili Mapserver. Uz već spomenuto zapisivanje na tvrdi disk računala, moguće je koristiti i neki od sustava za skladištenje podataka kao što je Memcached. Klijentske aplikacije, kao što je OpenLayers (odjeljak 4.2.7), mogu zatražiti dio rasterskog mozaika koristeći tri protokola: WMS, WorldWind i TMS. Na poslužitelju TileCache osim CGI sučelja može biti implementiran pomoću ModPython proširenja za Apache Web poslužitelj.

Instalacija

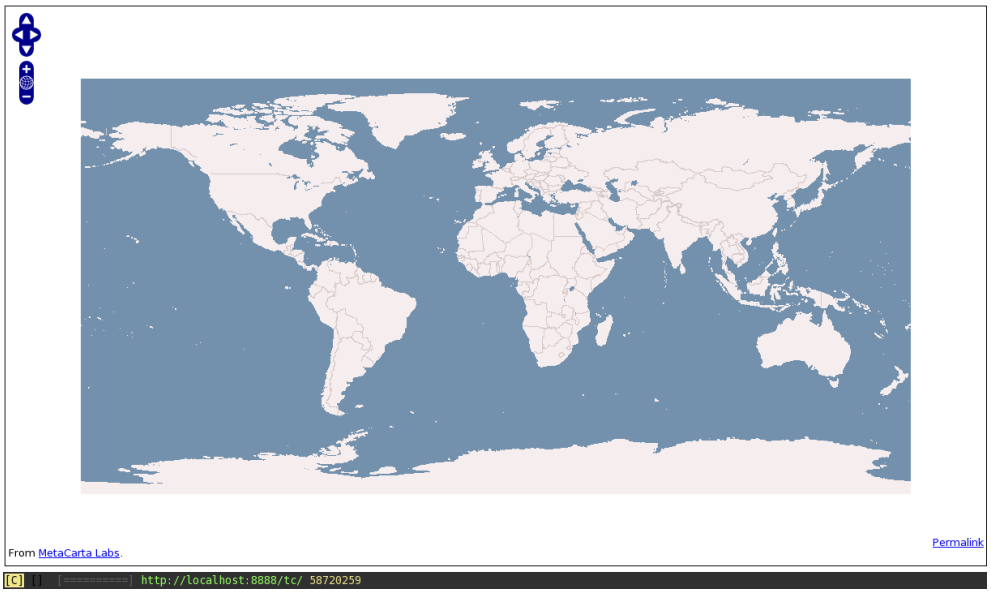
Instalacija TileCache softverskog paketa je trivijalna, svodi se na preuzimanje izvornog kôda i otpakiravanje istog.

```
wget http://tilecache.org/tilecache-2.10.tar.gz
tar zxf tilecache-2.10.tar.gz
```

Zadnji korak je kopiranje sadržaja otpakiranog direktorija u direktorij za koji je omogućeno pokretanje CGI skripti u konfiguraciji Apache Web poslužitelja:

```
cp -r tilecache-2.10/* /var/www/tc
```

Ispitivanje ispravnosti instalacije može se jednostavno napraviti upisivanjem Web adrese, `http://localhost/tc` u omiljeni internetski preglednik. Ukoliko sve funkcionira trebali bi vidjeti prikaz sličan onome na slici 4.2.



Slika 4.2: Osnovni prikaz nakon uspješne TileCache instalacije

4.2.7 OpenLayers

OpenLayers je biblioteka napisana u JavaScript programskom jeziku koja služi za prikazivanje prostornih podataka u svim modernim internetskim preglednicima. OpenLayers implementira JavaScript API za izgradnju raskošnih geografskih Web aplikacija, slično kao i Google Maps ili MSN Virtual Earth, no s jednom ključnom razlikom, OpenLayers je slobodni softver kojeg podržava Open Source Geospatial Foundation[11].

Biblioteka OpenLayers podržava velik broj standardiziranih geografskih formata kao što su OGC WMS, OGC WFS, GeoRSS, ka-Map, WorldWind i mnogi drugi. Osim toga OpenLayers ima ugrađen standardni skup kontrola koji omogućuju povećavanje, smanjivanje i pomicanje karata, oznaka, skočnih prozora, raznih navigacijskih komponenti, kontrolu pomoću tipkovnice i još mnogo toga. Primjer izgleda implementirane OpenLayers biblioteke se može vidjeti na slici 4.2.

Razvoj OpenLayers biblioteke započinje 2005. godine kao odgovor na tek izdani Google Maps API koji nije bilo moguće iskoristiti u nekomercijalne

[11] `http://www.osgeo.org`

svrhu. Naime, upravo zbog pojave OpenLayers biblioteke Google je omogućio upotrebu njihovog proizvoda svima koji su zainteresirani. OpenLayers omogućava jednostavnu integraciju različitih izvora podataka, poštujući pritom standarde, kako bi bilo tko mogao napraviti kartu sa željenim podacima. To se može smatrati najvećom prednošću nad vlasničkim GIS sustavima kod kojih je naglašena zatvorenost prema ostalim izvorima podataka.

Instalacija

Kako je OpenLayers napisan u JavaScript programskom jeziku, izvršavat će se u internetskom pregledniku kod korisnika servisa. Postoje dva načina integracije OpenLayers biblioteke u projekt:

1. integriranjem biblioteke direktno s OpenLayers Web stranica, što se može napraviti dodavanjem slijedeće linije HTML kôda

   ```
   <script src="http://openlayers.org/api/OpenLayers.js">
   </script>
   ```

2. preuzimanjem biblioteke s OpenLayers Web stranice, koja se otpakirava

   ```
   wget "http://www.openlayers.org/download/OpenLayers-2.7.tar.gz"
   tar zxf OpenLayers-2.7.tar.gz
   ```

   nakon čega se slijedeća linija dodaje u HTML kôd

   ```
   <script src="OpenLayers-2.7/OpenLayers.js"></script>
   ```

   ovaj drugi način je poželjniji zato što veza prema OpenLayers poslužiteljima može puknuti, a time će i aplikacija prestati raditi

Za OpenLayers biblioteku postoji opsežna dokumentacija i velik broj primjera koji se mogu pronaći na OpenLayers Web stranicama. Detaljnije o implementaciji OpenLayers biblioteke u diplomskom radu može se pročitati u petom poglavlju.

4.3 Podaci

Kako je već prije spomenuto ovaj diplomski rad temelji se samo na slobodnim podacima. Slobodnim podacima se smatraju oni podaci koje je moguće koristiti bez ograničenja u smislu preuzimanja, uređivanja i distribucije dok god se

podaci koriste u nekomercijalne svrhe. Sva tri skupa podataka *SRTM* (odjeljak 4.3.1), *Openstreetmap* (odjeljak 4.3.3) i *Corine pokrovi* (odjeljak 4.3.2) zadovoljavaju ove uvjete, a više o svakom opisano je u navedenim odjeljcima.

Iako su danas, prvenstveno zbog popularizacije široko pojasnog pristupa internetu, karte i prostorni podaci dostupniji nego ikad, još uvijek ostaje problem vlasništva tih podataka. Države troše novac poreznih obveznika za prikupljanje prostornih podataka, a zatim te podatke ponovno prodaju istim tim poreznim obveznicima. To je stvar uređenja države, poznato je da sve europske države imaju zakone o vlasništvu koji su restriktivni. S druge strane u Americi svi podaci koji su prikupljeni novcem poreznih obveznika postaju javno dobro. Najbolji primjer toga su SRTM podaci o elevacijama terena ili TIGER podaci prikupljeni popisom stanovništva.

4.3.1 SRTM

Za vrijeme Shuttle Radar Topography Mission (SRTM) prikupljeni su podaci o visinama terena u rasponu od 56. južne paralele do 60. sjeverne paralele na globalnoj razini. SRTM podaci Čine cjelovitu topografsku bazu Zemlje, visoke rezolucije. SRTM je iskoristio posebno izmijenjen radarski sustav postavljen na Space Shuttle Endeavour tijekom njegove jedanaesto dnevne misije u veljači 2000. godine. SRTM je internacionalni projekt kojeg su pokrenuli National Geospatial-Intelligence Agency (NGA) i National Aeronautics and Space Administration (NASA).

Elevacijski modeli su uređeni kao mozaik, u kojem svaki dio pokriva 1° geografske širine i 1° geografske duljine. Prema takvoj podjeli svakom dijelu se dodjeljuje naziv, npr. "n45e006" koji predstavlja dio koji se pruža od 45° sjeverne širine i 6° istočne dužine do 46° sjeverne širine i 7° istočne dužine. Rezolucija podataka od 1 lučne sekunde (oko 30m) dostupna je samo za područje Amerike, ostatak područja pokriven je s rezolucijom od 3 lučne sekunde (oko 90m).

Podaci su spremljeni u "hgt" formatu koji može pročitati većina GIS alata. U ovom diplomskom radu za čitanje i manipulaciju "hgt" podacima koristi se biblioteka *GDAL*.

SRTM podaci su javno dobro i kao takvi dostupni su svima bez ograničenja. Osim originalnih SRTM podataka postoje i podaci koji se temelje na SRTM podacima. Primjerice CGIAR Consortium for Spatial Information (CGIAR-CSI)[12] pruža SRTM podatke (rezolucije 90m) koji su popravljeni, odnosno popunjene su rupe u podacima, i dostupni su za preuzimanje u raznim for-

[12] `http://srtm.csi.cgiar.org/`

matima. Iako popravljeni podaci nisu javno dobro, licenca dopušta njihovo iskorištavanje za nekomercijalne projekte.

Postoje još HydroSHEDS[13] podaci, koje priprema U.S. Geological Survey (USGS), koji sadrže globalne i regionalne hidrografske informacije koje sadrže mreže rijeka, poplavna područja i mnoge druge. Nažalost iako su ovi podaci slobodni za nekomercijalne projekte, njihova licenca je restriktivnija od licence CGIAR-CSI podataka.

Preuzimanje orginalnih SRTM podataka moguće je korištenjem FTP protokola, odnosno aplikacije koja podržava komunikaciju FTP protokolom na adresi `ftp://e0srp01u.ecs.nasa.gov/srtm/`. Postoje dvije inačice podataka, "version1" koja sadrži orginalne neuređene podatke i "version2" koja je rezultat znatnih uređivanja, prvenstveno ekstremnih vrijednosti podataka. Preporuča se preuzimanje inačice "version2".

4.3.2 CORINE pokrovi

CORINE (Coordination of information on the environment) je program Europske komisije, pokrenut 1985. godine, koji ima tri cilja:

- prikupljanje informacija o stanju okoliša
- koordiniranje sastavljanja podataka i organizacija informacija
- osiguravanje konzistentnosti i kompatibilnosti podataka

U sklopu CORINE programa razvija se projekt prikupljanja informacija o pokrovima (vodene površine, šume, usjevi, kamenje...). Osnovna svrha je praćenje promjena koje se događaju s vremenom, kako bi se mogle donijeti bolje odluke u budućnosti. Podaci o pokrovima se prikupljaju prvenstveno sa satelitskih snimaka koje su znatno smanjile troškove i vrijeme potrebno za realizaciju projekta. Konačan proizvod je baza prostornih podataka koja se ažurira po ciklusima obrade podataka (za sada su to 1990., 2000. i 2006. godina).

Proces pripreme podataka je takav da svaka zemlja uključena u projekt mora potvrditi točnost podataka. U Republici Hrvatskoj je odlukom Ministarstva zaštite okoliša, prostornog uređenja i graditeljstva, referentna ustanova Agencija za zaštitu okoliša[14]. Umanjeni prikaz pokrova za Republiku Hrvatsku se može vidjeti na slici 4.3.

[13] `http://hydrosheds.cr.usgs.gov/index.php`
[14] `http://www.azo.hr`

Osim podataka CORINE pruža i opsežno dokumentiranu metodologiju rada i klasifikaciju podatka. Podaci se mogu slobodno koristiti za nekomercijalne projekte, ali ukoliko je potrebno može se zatražiti i dozvola za komercijalnu upotrebu. Preuzimanje CLC2000 (CORINE Land cover 2000) podataka, u ESRI SHP formatu, moguće je napraviti na Web adresi `http://www.eea.europa.eu/themes/landuse/clc-download`.

Slika 4.3: CORINE pokrov za područje Republike Hrvatske

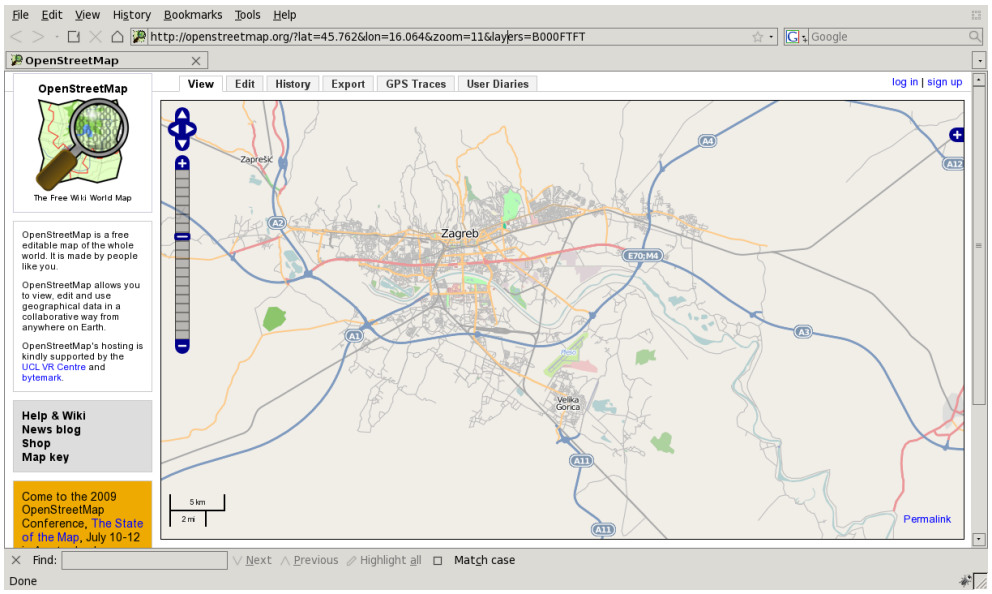
4.3.3 Openstreetmap

Openstreetmap[15] je projekt koji stvara i osigurava slobodne geografske podatke. Projekt je pokrenut zbog toga što većina karata, za koje mislimo da su

[15] `http://openstreetmap.org`

slobodne, ima neki oblik pravnih ili tehničkih ograničenja u načinu na koji će se koristiti.

Projekt počinje u srpnju 2004. godine prikupljanjem prostornih podataka koristeći ručne GPS uređaje. Osnovna ideja je prikupljanje podataka o cestama, ali ubrzo se proširila na gotovo sve prostorne objekte. Prikupljanje podataka prvenstveno rade pojedinci, često okupljeni na tzv. MappingPartyu gdje se sustavno prikupljaju podaci o nekom području. Osnovni prikaz Openstreetmap sučelja može se vidjeti na slici 4.4.



Slika 4.4: Osnovno Openstreetmap sučelje

Sloboda podataka proizlazi iz licence koju prihvaćaju svi korisnici. Naime izabrana licenca Creative Commons Imenovanje - Dijeli pod istim uvjetima, čiji se sažeti tekst može pročitati na Web stranici `http://creativecommons.org/licenses/by-sa/2.0/deed.hr`. Ova licenca omogućava korisnicima da slobodno umožavaju, distribuiraju, objavljuju i prerađuju Openstreetmap podatke dokle god te podatke dijele pod istim uvjetima. Zbog ovakve licence nije moguće jednostavno iskorisiti podatke koji nisu javno dobro. Zanimljivo je da sve više "proizvođača" podataka oslobađa svoje podatke za korištenje u OSM projektu. Najbolji primjer je Yahoo koji dopušta korištenje satelitskih snimaka u svrhu vektoriziranja podataka.

Zajednica korisnika za organizaciju i dokumentiranje koristi Wiki sustav dostupan na Web adresi `http://wiki.openstreetmap.org`. U ovom trenutku OSM projekt ima oko 80000 registriranih korisnika iz cijeloga svijeta, a baza podataka sadrži 1150 milijuna GPS točaka od kojih je 350 milijuna čvorova.

Specifično za projekte otvorenog tipa porast broja korisnika je eksponencijalan, a broj korisnika koji aktivno uređuju OSM podatke mjesečno je oko 10%.

Openstreemap za centralnu bazu podataka koristi PostgreSQL (odjeljak 4.2.2) sustav za upravljanje bazama podataka, dok se za iscrtavanje dijelova rasterskog mozaika pomoću Mapnika (odjeljak 4.2.5) koristi PostGIS (odjeljak 4.2.3). Baza podataka zauzima oko 820 gigabajta prostora, a izvoz podataka zauzima oko 150 gigabajta.

Nažalost u Hrvatskoj zajednica OSM korisnika nije jako razvijena, iako se mogu vidjeti neki pozitivni pomaci. S druge strane država i državne institucije nemaju previše razumijevanja za oslobađanje podataka.

Poglavlje 5

Implementacija softverskog stoga

Ovo poglavlje opisuje postupak pripremanja podataka koji će se koristiti za iscrtavanje kartografske podloge. Prvo se preuzimaju podaci CGIAR-CSI SRTM podaci i iz njih se izvlače slojnice s razmacima od 10m, 50m i 100m. Osim toga SRTM podaci će se iskoristiti i za stvaranje kontinuirano obojanog reljefa po visinama koji će biti temeljna podloga karte. Nakon SRTM podataka objašnjeno je preuzimanje i pripremanje CORINE i Openstreetmap podataka.

Nadalje, objašnjeno je pripremanje Mapnik XML datoteke i iscrtavanje dijelova rasterskog mozaika korištenjem TileCache softverskog paketa. Nakon toga objašnjava se postupak stvaranja servisa koji omogućuje vizualizaciju, pretraživanje i manipulaciju prethodno pripremljenih prostornih podataka.

5.1 Definicije prostornih sustava

U diplomskom radu za kartografski sustav je odabran EPSG:900913. Taj kartografski sustav ima slijedeće parametre:

sferoid WGS_1984

projekcija merkator

početna geografska širina 0.0

centralni meridijan 0.0

faktor skaliranja 1.0

jedinica metar

Definicija te projekcije za PROJ4 softverski paket, spomenut u poglavlju 4.2.3, je slijedeća:

```
+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0\
 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs
```

European Petroleum Survey Group (EPSG) je znanstvena organizacija koja veže europsku naftnu industriju, odnosno stručnjake iz područja izmjere, sa kartografijom o istraživanju naftnih područja. EPSG je sastavila skup geodetskih parametara koji sadrži informacije o Zemljinim elipsoidima, geodetskim datumima te projekcijama. Elementi skupa su definirani numeričkim kodovima i metapodacima o elipsoidu, projekciji, jedinicama, itd.

5.2 Preuzimanje i pripremanje SRTM podataka

CGIAR-CSI SRTM podaci se mogu preuzeti sa slijedeće Web stranice `http://srtm.csi.cgiar.org/SELECTION/inputCoord.asp`. Prije samog preuzimanja potrebno je odabrati interesne odjeljke. Nakon odabira odjeljaka pojavit će se popis i detaljniji prikaz istih u GeoTiff formatu zapisa podataka. Odjeljci koji pokrivaju područje Hrvatske su slijedeće: srtm_39_03, srtm_39_04, srtm_40_03 i srtm_40_04. Odjeljke je najjednostavnije preuzeti na slijedeći način:

```
wget http://srtm.geog.kcl.ac.uk/portal/srtm41/\
srtm_data_geotiff/srtm_39_03.zip
```

Nakon otpakiravanja ZIP arhive može se pristupiti izradi slojnica upotrebom gdal alata "gdal_contour" koji se koristi na slijedeći način:

```
gdal_contour -i 100 -snodata 32767 -a height \
srtm_39_03.tif srtm_39_03_100.shp
gdal_contour -i 50 -snodata 32767 -a height \
srtm_39_03.tif srtm_39_03_50.shp
gdal_contour -i 10 -snodata 32767 -a height \
srtm_39_03.tif srtm_39_03_10.shp
```

Alat "gdal_contour" uzima slijedeće parametre: "-i" označava razmak između slojnica, "-snodata" označava vrijednost koja predstavlja područje bez podataka, "-a height" označava atribut koji predstavlja visinu, "tif" datoteka predstavlja ulaznu datoteku, a "shp" izlaznu. Vidljivo je da je postupak potrebno izvršiti 3 puta za slojnice s razmakom od 100m, 50m i 10m.

SHP datoteke sadrže mnogo podataka koje imaju visinu manju od 0, stoga je takve podatke potrebno izbaciti. Najjednostavnije je napraviti novu SHP datoteku koja ne sadrži te podatke, npr. "srtm_39_03_100_cl.shp". Taj postupak se izvodi slijedećom naredbom:

```
ogr2ogr -overwrite -where 'height> 0'\
 srtm_39_03_100_cl.shp srtm_39_03_100.shp
```

Nakon pretvorbe potrebno je "SHP" datoteke za pojedini razmak slojnica stopiti u jednu, kako bi se kasnije olakšalo manipuliranje i uključivanje iste u Mapnik datoteku stila. Stapanje se radi pomoću "ogr2ogr" alata koji je također dio "gdal" softverskog paketa.

```
ogr2ogr dem_100.shp srtm_39_03_100_cl.shp
ogr2ogr -update -append dem_100.shp srtm_39_04_100_cl.shp -nln dem_100
ogr2ogr -update -append dem_100.shp srtm_40_03_100_cl.shp -nln dem_100
ogr2ogr -update -append dem_100.shp srtm_40_04_100_cl.shp -nln dem_100
```

Ovim postupkom nastaje nova SHP datoteka "dem_100" koja sadrži sve stopljene podatke. Isti postupak se provodi i za ostale slojnice. Nakon stapanja potrebno je napraviti prostorni indeks kako bi se ubrzao pristup podacima. Prostorni indeks se stvara pomoću alata "shapeindex" na slijedeći način:

```
shapeindex dem_100
```

Poželjno je napraviti transformaciju SHP datoteke iz EPSG:4326 kartografskog sustava u EPSG:900913 kako bi se ubrzalo iscrtavanje dijelova rasterskog mozaika. Mapnik može koristiti slojeve podataka koji se nalaze u različitim kartografskim sustavima, odnosno napraviti projekciju u željeni kartografski sustav. No kako će se ta projekcija morati izvoditi svaki put prilikom pristupa tom sloju poželjno ju je napraviti samo jedanput. Projekcija u neki drugi kartografski sustav izvodi se izvršavanjem naredbe:

```
ogr2ogr -t_srs "+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0\
 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs\
 +over" -s_srs "+proj=latlong +datum=WGS84"\
 dem_100_900913.shp dem_100.shp
```

Potrebno je još napraviti kontinuirano obojani reljef po visinama što se može napraviti alatom "demtools"[1] odnosno softverom "color-relief" na slijedeći način:

```
color-relief srtm_39_03.tif scale.txt relief_39_03.tif
```

Prvi argument označava ulazne podatke, odnosno digitalni model reljefa, drugi argument je datoteka koja sadrži informacije o odnosu boja i visina, a treći argument naziv izlazne datoteke.

[1] http://www.perrygeo.net/wordpress/?p=7

Problem s ovim rasterima je što se nalaze u EPSG:4326 projekciji, a kako će se ostali podaci prikazivati u EPSG:900913 projekciji, mora se napraviti reprojekcija rastera. U tu svrhu se koristi alat "gdalwarp" koji je dio GDAL softverskog paketa, na slijedeći način:

```
gdalwarp -t_srs '+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0\
 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs\
+over' -co INTERLEAVE=PIXEL -co TFW=YES -co TILED=YES -of GTiff\
 relief_40_03.tif relief_40_03_900913.tif
```

Nakon uspješnog izvođenja svih navedenih koraka dobiveni su slijedeći slojevi podataka:

- vektorski slojevi podataka, koji sadrže slojnice s različitim razmacima
  - dem_100_900913.shp
  - dem_50_900913.shp
  - dem_10_900913.shp
- rasterski slojevi podataka koji sadrže kontinuirano obojeni reljef po visinama
  - relief_39_03_900913.tif
  - relief_39_04_900913.tif
  - relief_40_03_900913.tif
  - relief_40_04_900913.tif

5.3 Preuzimanje i pripremanje CORINE podataka

CORINE podaci o pokrovima preuzimaju se s Web adrese `http://www.eea.europa.eu/themes/landuse/clc-download` gdje je potrebno odabrati željene odjeljke rasterskog mozaika. Nakon preuzimanja i otpakiravanja datoteka potrebno ih je stopiti kako bi se olakšala kasnija manipulacija. Stapanje se radi korištenjem već spomenutog "ogr2ogr" alata, a poslije stapanja potrebno je indeksirati SHP datoteku.

```
ogr2ogr corine.shp 100KME45N24.shp
ogr2ogr -update -append corine.shp 100KME46N23.shp -nln corine
ogr2ogr -update -append corine.shp 100KME46N24.shp -nln corine
```

```
ogr2ogr -update -append corine.shp 100KME46N25.shp -nln corine
ogr2ogr -update -append corine.shp 100KME47N23.shp -nln corine
ogr2ogr -update -append corine.shp 100KME47N24.shp -nln corine
ogr2ogr -update -append corine.shp 100KME47N25.shp -nln corine
ogr2ogr -update -append corine.shp 100KME47N26.shp -nln corine
ogr2ogr -update -append corine.shp 100KME48N22.shp -nln corine
ogr2ogr -update -append corine.shp 100KME48N23.shp -nln corine
ogr2ogr -update -append corine.shp 100KME48N24.shp -nln corine
ogr2ogr -update -append corine.shp 100KME48N25.shp -nln corine
ogr2ogr -update -append corine.shp 100KME48N26.shp -nln corine
ogr2ogr -update -append corine.shp 100KME49N22.shp -nln corine
ogr2ogr -update -append corine.shp 100KME49N25.shp -nln corine
ogr2ogr -update -append corine.shp 100kmE50N21.shp -nln corine
ogr2ogr -update -append corine.shp 100kmE50N22.shp -nln corine
ogr2ogr -update -append corine.shp 100kmE50N24.shp -nln corine
ogr2ogr -update -append corine.shp 100kmE50N25.shp -nln corine
shapeindex corine
```

5.4 Preuzimanje i pripremanje Openstreetmap podataka

Prvi korak je preuzimanje pomoćnih prostornih podataka koji uključuju granice i obalne linije u dvije razine preciznosti. Preuzimanje se izvodi izvršavanjem ovih naredbi:

```
wget http://tile.openstreetmap.org/world_boundaries-spherical.tgz
wget http://hypercube.telascience.org/~kleptog/processed_p.zip
wget http://tile.openstreetmap.org/shoreline_300.tar.bz2
```

Slijedeći korak je preuzimanje OSM podataka za Hrvatsku. To je moguće napraviti na slijedećoj Web adresi `http://download.geofabrik.de/osm/europe/`, gdje se podaci osvježavaju jednom dnevno.

```
http://download.geofabrik.de/osm/europe/croatia.osm.bz2
```

Prije uvoza OSM podataka u bazu podataka potrebno je stvoriti novu bazu podataka koristeći prethodno pripremljeni predložak (odjeljak 4.2.3).

```
createdb -T postgis_template diplomski
```

Uvoz podataka izvodi se pomoću alata "osm2pgsql" kojeg je prvo potrebno preuzeti u izvornom kôdu i prevesti, nakon čega se mogu uvesti podaci.

```
svn co http://svn.openstreetmap.org/applications/\
utils/export/osm2pgsql/ osm2pgsql
cd osm2pgsql
make
```

Osim uvoza podataka potrebno je u bazu podataka uvesti definiciju kartografskog sustava "900913", što omogućuje transformacije u i prema tom sustavu. Kako bi uvezli OSM podatke u bazu podataka prvo ih je potrebno otpakirati, nakon čega se uvoz podataka se izvodi izvršavanjem slijedeće naredbe:

```
psql -d diplomski -f 900913.sql
bunzip croatia.osm.bz2
osm2pgsql -m -d diplomski croatia.osm
```

Bitno je naglasiti da parameter "-m" u zadnjoj naredbi izvodi transformaciju podataka iz kartografskog sustava "4326" u kartografski sustav "900913".

5.5 Iscrtavanje i vizualizacija pomoću Mapnik, TileCache i Openlayers alata

Kako je opisano u poglavlju 4.2.5, pravila iscrtavanja se nalaze u datoteci XML formata. Priprema te datoteke objasnit će se korak po korak za svaki sloj podataka.

5.5.1 OSM podaci

Prvo je potrebno pripremiti OSM dio XML datoteke, jer on je najopsežniji zbog svih pravila iscrtavanja OSM podataka. Srećom njegova priprema je automatizirana, a se obavlja tako da se prvo preuzmu datoteke iz OSM Mapnik repozitorija:

```
svn checkout http://svn.openstreetmap.org/applications/\
rendering/mapnik
cd mapnik
```

Nakon toga potrebno je otpakirati datoteke koje su preuzete ranije (odjeljak 5.4), i razmjestiti ih po direktorijima.

```
tar xzf world_boundaries-spherical.tgz
unzip processed_p.zip
```

```
mv coastlines/* world_boundaries/
rmdir coastlines
tar xjf shoreline_300.tar.bz2 -C ~/mapnik/world_boundaries
```

Slijedeći korak je uređivanje datoteke "set-mapnik-env" koja postavlja okolinu u kojoj će se stvoriti Mapnik XML datoteka. Uređivanje datoteke, odabranim uređivačem teksta, dovoljno je intuitivno i neće biti detaljnije objašnjeno. Nakon toga postavlja se okolina i stvara početna XML datoteka koja ima skoro 7000 linija.

```
. ./set-mapnik-env
./customize-mapnik-map >$MAPNIK_MAP_FILE
```

5.5.2 SRTM podaci

Već prije je spomenuto kako Mapnik korisiti tzv. slikajući model iscrtavanja rastera, stoga kako bi se slojevi koji sadržavaju SRTM podatke iscrtali prije OSM slojeva, u Mapnik XML datoteci moraju se navesti prije OSM slojeva. Primjer zapisa za jedan prethodno stvoreni raster je slijedeći:

```
<Layer name="dem1" status="on"  srs="+proj=merc +a=6378137\
 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0\
 +units=m +nadgrids=@null +no_defs +over">
  <StyleName>raster</StyleName>
  <Datasource>
    <Parameter name="type">raster</Parameter>
    <Parameter name="file">/data/relief_39_03_900913.tif</Parameter>
    <Parameter name="lox">1113148.557</Parameter>
    <Parameter name="loy">5621412.455</Parameter>
    <Parameter name="hix">1669789.542</Parameter>
    <Parameter name="hiy">6446348.064</Parameter>
  </Datasource>
</Layer>
```

U skladu s time potrebno je definirati i stil naziva "raster" koji će poslužiti za iscrtavanje rastera.

```
<Style name="raster">
  <Rule>
    <RasterSymbolizer>
    </RasterSymbolizer>
  </Rule>
</Style>
```

Na sličan način se definira sloj podataka koji sadrži slojnice. U ovom slučaju prikazane su slojnice s razmakom od 100m. Može se primijetiti kako ovaj sloj podataka koristi dva stila "contours100" koji prikazuje slojnice i "contours-text100" koji ispisuje visinske kote. Stilovi se neće posebno prikazivati zbog njihove kompleksnosti i ograničenosti prostorom.

```
<Layer name="srtm100" status="on" srs="+proj=merc +a=6378137\
 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0\
 +units=m +nadgrids=@null +no_defs +over">
  <StyleName>contours100</StyleName>
  <StyleName>contours-text100</StyleName>
  <Datasource>
    <Parameter name="type">shape</Parameter>
    <Parameter name="file">/data/dem_100_900913</Parameter>
  </Datasource>
</Layer>
```

5.5.3 CORINE podaci

CORINE sloj podataka se postavlja odmah iza SRTM sloja, na sličan način kao i SRTM podaci, definicijom novog sloja i stilova. Jedinu bitnu razliku predstavlja kartografski sustav u kojem se nalaze CORINE podaci. Za CORINE podatke to je "GRS80" elipsoid i ekvivalentna projekcija.

```
<Layer name="Corine" status="on" srs="+proj=laea +lat_0=52\
 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80\
 +units=m +no_defs">
<StyleName>Corine</StyleName>
<Datasource>
  <Parameter name="type">shape</Parameter>
  <Parameter name="file">/data/corine</Parameter>
</Datasource>
</Layer>
```

Jedini stil, "Corine", sadrži definirana pravila iscrtavanja za svaki podatak, ovisno o atributima podataka. Navedena su samo neka pravila:

```
<Style name="Corine">
  <Rule>
    <Filter>[CODE_00] = '111'</Filter>
    <PolygonSymbolizer>
      <CssParameter name="fill">#ffffff</CssParameter>
      <CssParameter name="fill-opacity">0.7</CssParameter>
```

```
    </PolygonSymbolizer>
  </Rule>

  <Rule>
    <Filter>[CODE_00] = '112'</Filter>
    <PolygonSymbolizer>
      <CssParameter name="fill">#ffffff</CssParameter>
      <CssParameter name="fill-opacity">0.3</CssParameter>
    </PolygonSymbolizer>
  </Rule>
...
```

Svako pravilo uključuje filtar, koji ograničava primjenu tog pravila, odnosno parametara iscrtavanja.

5.5.4 TileCache konfiguracijska datoteka

TileCache se u potpunosti konfigurira pomoću jedne datoteke "tilecache.cfg". Datoteka je dobro dokumentirana te se uređivanje svodi na definiciju dva odjeljka "[cache]" i "[layer]". Odjeljak "[cache]" definira tip i lokaciju gdje se spremaju dijelovi rasterskog mozaika. Odjeljak "[layer]" definira parametre sloja te nosi naziv sloja, u ovom slučaju naziv je "osm".

```
[cache]
type=Disk
base=/home/dodobas/tiles

[osm]
type=Mapnik
mapfile=osm.xml
spherical_mercator=true
metatile=yes
metaSize=5,5
metaBuffer=256
extent_type=loose
srs=EPSG:900913
debug=no
```

Od parametara najzanimljiviji su: "mapfile" koji definira koja se Mapnik XML datoteka koristi i "srs" koji definira kartografski sustav.

5.5.5 TileCache stvaranje rasterskog mozaika

TileCache može iscrtavati dijelove rasterskog mozaika ovisno o zahtjevu korisnika, što radi samo ako taj dio rasterskog mozaika već nije iscrtan. Iako je TileCache moguće koristiti i ovako, nikako se ne preporuča.

Zajedno s TileCache softverskim paketom dolazi alat "tilecache_seed.py" koji se koristi za stvaranje dijelova rasterskog mozaika. Primjerice, slijedeća naredba iscrtat će sve dijelove rasterskog mozaika koji se nalaze unutar zadanog pravokutnika za sloj "osm" i za razinu približenja 7.

```
tilecache_seed.py -b 1100000,5100000,2600000,5900000 osm 7 8
```

Nakon iscrtavanja na tvrdom disku, u direktoriju definiranom u TileCache konfiguracijskoj datoteci, stvorit će se mnogo "PNG" datoteka. Vrijeme iscrtavanja ovisiti će o količini i vrsti slojeva koji su definirani u Mapnik XML datoteci. Proces može duže potrajati jer svaki slijedeći nivo približenja sadrži četri puta više datoteka koje tvore rasterski mozaik.

5.5.6 OpenLayers vizualizacija

OpenLayers (odjeljak 4.2.7 koristi se za vizualizaciju sloja podataka koji je stvoren pomoću TileCache alata. Kako bi se taj sloj podataka mogao vizualizirati, potrebno je urediti "index.html" datoteku koja se nalazi u istom direktoriju kao i TileCache instalacija.

Prvo se mora definirati konfiguracija za "map" objekt, koji se kasnije instancira. U konfiguraciji su definirani parametri kao što je "restrictedExtent" koji ograničava kretanje po karti, "numZoomLevels" koji definira broj raspoloživih nivoa približenja ili recimo "maxResolution" koji ograničava gornji nivo približenja.

```
var options = {
  units: 'm',
  maxResolution: 1222.992452344,
  maxExtent: new OpenLayers.Bounds(-20037508, -20037508,
    20037508, 20037508.34),
  restrictedExtent: new OpenLayers.Bounds(1422591, 5150000,
    2404881, 5898620),
  numZoomLevels: 7,
  controls:[new OpenLayers.Control.MouseDefaults()]
  }

map = new OpenLayers.Map( 'map' , options);
```

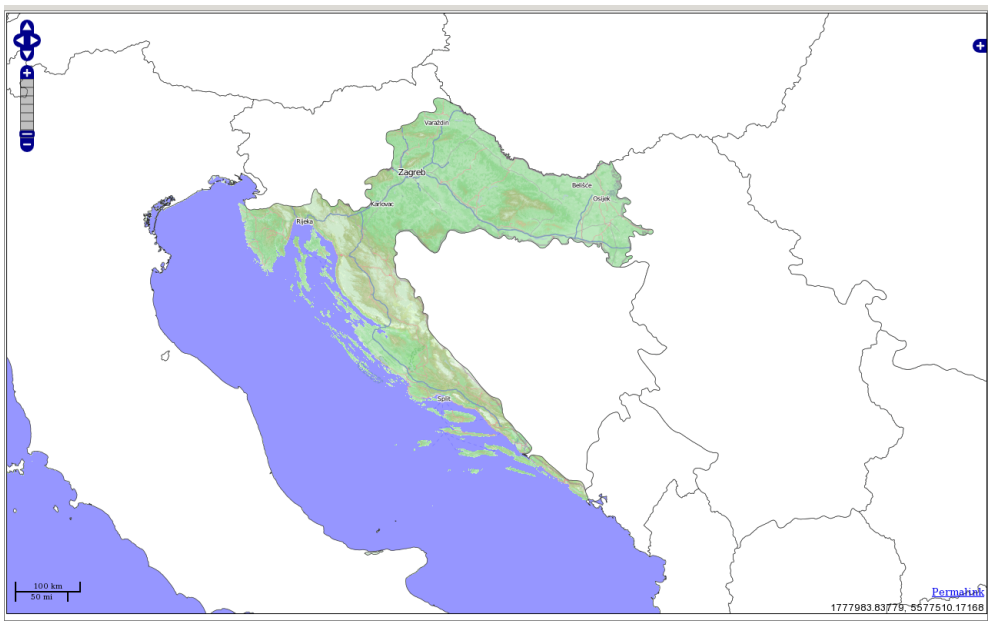
```
layer = new OpenLayers.Layer.WMS( "podloga",
  "tilecache.cgi?", {layers: 'osm', format: 'image/png' } );

map.addLayer(layer);
```

Osim toga definiraju se još i OpenLayers kontrole koje proširuju funkcionalnost i omogućavaju dodatnu interaktivnost sučelja. Recimo kontrola "Permalink" omogućava stvaranje poveznice koja će uvijek prikazivati trenutnu lokaciju.

```
map.addControl(new OpenLayers.Control.Permalink());
map.addControl(new OpenLayers.Control.MousePosition());
map.addControl(new OpenLayers.Control.ScaleLine());
map.addControl(new OpenLayers.Control.KeyboardDefaults());
map.addControl(new OpenLayers.Control.LayerSwitcher(
  {'ascending':false}));
map.addControl( new OpenLayers.Control.PanZoomBar()
```

Rezultat uređivanja "index.html" datoteke prikazuje slika 5.1. Izgled, kontrole te druge mogućnosti OpenLayers softverskog paketa moguće je dodatno prilagoditi ili je moguće razviti posebnu funkcionalnost.



Slika 5.1: OpenLayers vizualizacija TileCache sloja podataka

5.6 pgRouting podaci

Posljednji korak prije izrade interaktivne aplikacije predstavlja stvaranje i punjenje nove baze podatka OSM podacima. Iako nije poželjno koristiti dvije baze podataka s istim podacima, jer je onda potrebno raditi dvostruko ažuriranje, u ovom slučaju se koriste zbog jednostavnosti. Prvo se mora stvoriti nova baza podatka upotrebom prethodno pripremljenog predloška "pgrouting_template".

```
createdb -T pgrouting_template diplomski_rt
```

Uvoz OSM podataka radi se upotrebom alata "osm2pgrouting" kojeg je potrebno preuzeti u izvornom kôdu i prevesti.

```
svn checkout http://pgrouting.postlbs.org/svn/pgrouting/\
tools/osm2pgrouting/trunk osm2pgrouting
make
```

Alat "osm2pgrouting" koristi jednu XML datoteku koja definira klasifikaciju OSM podataka. Pojedinoj kategoriji se dodjeljuje određeni jedinstveni identifikator koji se kasnije koristi u algoritmima traženja optimalnog puta. Primjerice kategoriji "motorway" se dodjeljuje "id" 101 ili recimo kategoriji "primary" kojoj se dodjeljuje "id" 106.

Slijedeća naredba puni bazu OSM podacima te ih priprema za pretraživanje algoritmima. Ova naredba će se izvršavati nešto duže zbog količine izračuna koje je potrebno napraviti.

```
osm2pgrouting -file croatia.osm -conf mapconfig.xml\
 -dbname diplomski_rt
```

5.7 Interaktivna aplikacija

U ovom odjeljku ukratko će se pojasniti funkcioniranje interaktivne aplikacije, a puni izvorni kôd može se pronaći u dodacima (A i B). Interaktivna aplikacija je izgrađena od dvije komponente:

index.html datoteka koja je osnova OpenLayers konfiguracije i interakcije s korisnicima

getdata.cgi Python datoteka koja izvršava upite u bazi podataka te vraća moguće rezultate, kako bi ih OpenLayers mogao prikazati

Prilagođava se “index.html” datoteka koja je napravljena u odjeljku 5.5.6. U dodatku A od linije 3 do linije 29 definiran je CSS stil koji se koristi. U linijama 32 do 52 definirani su parametri stilova “result_style”, “start_style” i “stop_style” koji se kasnije koriste za prikaz rezultata, početne točke pretraživanja i završne točke pretraživanja. Definirana je i nova klasa objekta “SinglePoint” koja dozvoljava korištenje samo jednog početnog i završnog objekta, linije 54 do 63.

Linije 67 do 101 definiraju parametre, slojeve i kontrole OpenLayers softverskog paketa. Osim “osm” temeljnog sloja podataka, definirana su i tri pomoćna sloja “Rezultat”, “Početna točka” i “Završna točka” čija je svrha prikazivanje tih objekata. Osim standardnih kontrola dodane su i dvije kontrole, linije 103 do 107, koje koriste prethodno definiranu klasu “SinglePoint” za iscrtavanje grafičkih prikaza na slojevima “start” i “stop”.

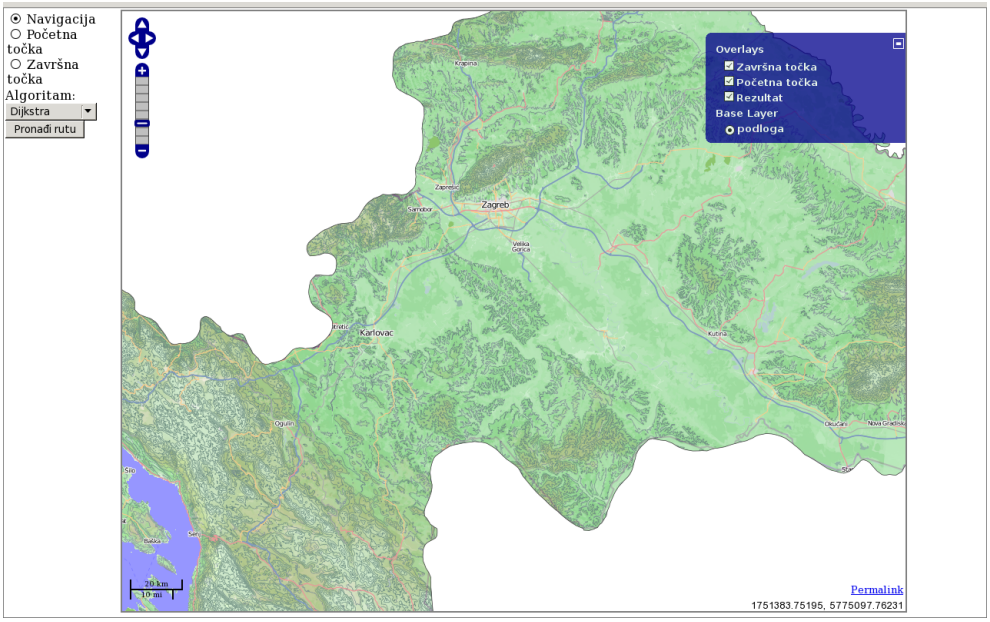
Ključne linije kôda su 123 do 138 kojima je definirana funkcija “compute”. Ta funkcija uzima koordinate početne i završne točke te traženi algoritam pa ih prosljeđuje Python “getdata.cgi” skripti koja ih obrađuje i vraća rezultat upita. Najzanimljivija je linija 135 koja, asinkrono u pozadini koristeći AJAX funkcionalnost, poziva “getdata.cgi” skriptu i prosljeđuje moguće rezultate funkciji “displayRoute”. Funkcija “displayRoute”, linije 140 do 153 obrađuje rezultate tako da prikaže svaki element na prethodno definiranom pomoćnom sloju podataka “result”.

Linije 156 do 186 predstavljaju HTML kôd stranice koja se prikazuje korisniku. Između ostalog definirane su kontrole kojima korisnik definira svoje akcije.

U dodatku B nalazi se kôd Python skripte “getdata.cgi” koja izvršava upit u bazi podataka. U linijama 1 do 6 uvoze se potrebni Python moduli. Funkcija “findNearestEdge” se spaja na bazu podatka i traži najbliži brid ovisno o korisnički zadanim koordinatama. Ukoliko se pronađe takav brid informacije o njemu će se spremiti i vratiti kao rezultat funkcije. U slučaju da brid nije pronađen skripta će završiti s izvođenjem, a rezultat će biti prazan skup.

Slijedeći skup linija, 34 do 41, obrađuje ulazne vrijednosti te poziva funkciju “findNearestEdge” čije rezultate sprema u varijable “pocetnibrid” i “krajnjibrid”. Osim toga sprema vrijednost odabrane metode u varijablu “metoda”. Slijedeći korak ovisno o odabranoj metodi, određuje “sql” kôd koji se izvršava u bazi podataka, linije 43 do 62. Linije 64 do 67 rade upravo to, tj. izvršavaju SQL kôd u bazi podataka i spremaju moguće rezultate u varijablu “data”. Na kraju još ostaje pripremiti XML kôd koji obrađuje prije spomenuta funkcija “displayRoute”, i prikazati rezultate na karti.

Nakon uređivanja svih datoteka korisnici će moći pristupiti servisu koji izgleda kao na slici 5.2. U slijedećem odjeljku ukratko je objašnjen način uporabe izgrađenoga servisa.



Slika 5.2: Prikaz servisa uređivanja datoteka

5.7.1 Osnove korištenja

Standardna funkcionalnost OpenLayers softverskog paketa omogućava pomicanje po karti, jednostavnim povlačenjem karte u proizvoljnom smjeru ili klikom na jednu od ikona sa smjerovima. Razinu približenja je moguće postaviti klizačem u gornjem lijevom kutu karte ili korištenjem kotačića miša. Vidljivost slojeva podataka se kontrolira otvaranjem izbornika koji se nalazi u gornjem desnom kutu u kojem je moguće uključiti ili isključiti određeni sloj podataka. U donjem desnom kutu klikom na poveznicu "Permalink" dobit će se Web adresa s kojom je moguće trajno pristupati trenutnom kartografskom prikazu.

Dodatna funkcionalnost se kontrolira pomoću izbornika s lijeve strane kartografskog prikaza koji ima slijedeće stavke:

Navigacija standardnu funkcionalnost OpenLayers softverskog paketa

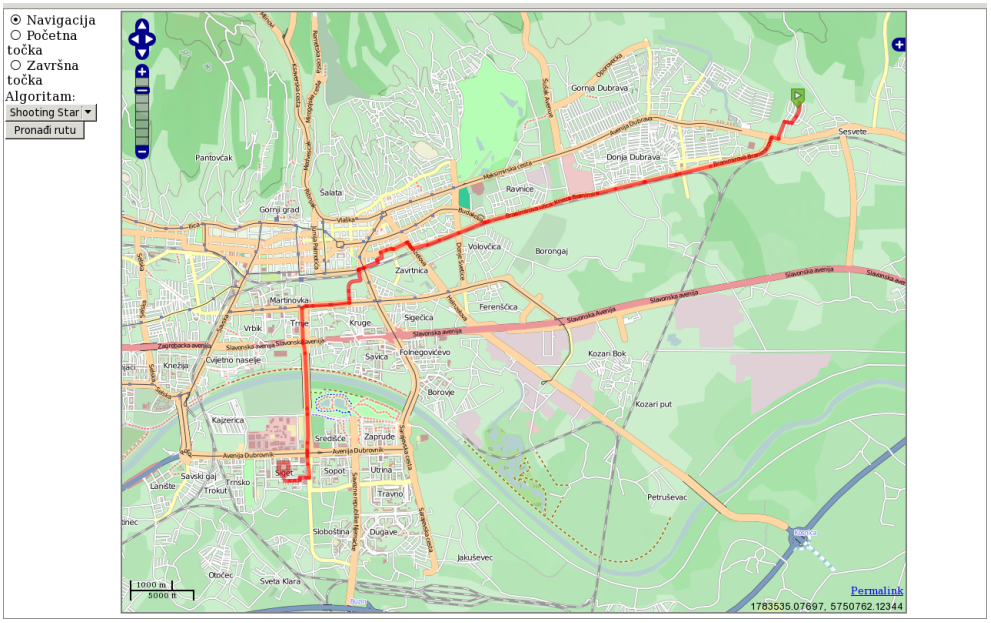
Početna točka omogućava postavljanje početne točke

Završna točka omogućava postavljanje završne točke

Algoritam odabir algoritma za izračun optimalne putanje, moguće je izabrati između "Dijkstra", "A Star" i "Shooting star" algoritama

Pronađi rutu pokreće traženje optimalne putanje za odabrani algoritam

Nakon što korisnik odabere obje točke te pokrene traženje optimalne putanje mogao bi dobiti rezultat kao što je prikazano na slici 5.3.



Slika 5.3: Prikaz rezultata pretraživanja

Bitno je napomenuti da algoritmi "Dijsktra" i "A star" uglavnom daju lošije rezultate od algoritma "Shooting Star". S druge strane zbog loše topologije OSM podataka nije niti moguće očekivati točnija rješenja, odnosno optimalne putove. Osim toga potrebno je bolje definirati "težinu" pojedine klase, kako bi dobili točnija rješenja.

Poglavlje 6

Zaključak

Diplomski rad opisuje specifikaciju mrežnih prostornih podataka, i prikazuje načine kako ih je moguće kvalitetno modelirati. Modeliranje podataka je ključno prije slijedećeg koraka, implementacije. Efikasna implementacija se postiže optimizacijom i specijalizacijom algoritama koji koriste mrežne modele prostornih podataka. Algoritam je moguće proširiti uključivanjem informacije o hijerarhiji prostornih podataka, čime se skraćuje vrijeme pretraživanja zbog manjeg skupa podataka.

Opisani algoritmi, širinsko i dubinsko pretraživanje te Dijkstra, Bellman-Ford, A*, D*, Floyd-Warshall i Johnson su temelj istraživanja problema pretraživanja grafova. Određivanje optimalnog puta prvenstveno ovisi o kriteriju koji definira optimalnost tog puta. Kriterij može biti najkraći put (vremenski ili daljinski), najlakši put (put ne smije uključivati stepenice) ili bilo koji drugi kriterij. Svaki algoritam je dovoljno općenit i moguće ga je iskoristiti u različitim slučajevima. No, problem trgovačkog putnika, za koji ne postoji efikasno rješenje, predmet je istraživanja svjetske znanstvene zajednice.

Korišten je slobodni softver koji je odabran zbog kvalitete, sigurnosti, mogućnosti prilagodbe i zbog toga što implementira postojeće otvorene standarde. Upravo poštivanje i implementacija otvorenih standarda za razmjenu podataka omogućuje brz razvoj i prilagođavanje servisa za korisnika. U diplomskom radu se koristi Ubuntu Linux distribucija na koju su instalirani ostali softverski paketi. PostgreSQL objektno relacijska baza podataka služi za skladištenje prostornih podataka i proširena je PostGIS i pgRouting prostornim nadogradnjama. Mapnik i TileCache se koriste za iscrtavanje i pripremu rasterskog mozaika, a OpenLayers za interakciju i komunikaciju sa korisnicima.

Osim slobodnog softvera koriste se slobodni podaci, a pojavom projekata kao što je Openstreetmap, polako se podiže svijest o nužnosti slobodnih podataka. Ostali korišteni slobodni podaci su SRTM podaci o elevacijama terena i

CORINE baza podataka o pokrovima.

Ovaj diplomski rad između ostalog pokazuje da je moguće izgraditi servis temeljen potpuno na slobodnom softveru i slobodnim podacima, i da se prava vrijednost ne nalazi u sirovim podacima nego u servisima koji se temelje na tim podacima. Vlasnički podaci samo pogoduju razvoju monopolističkog tržišta koje je inertno upravo zbog svoje zatvorenosti. Uvijek će postojati potreba za privatnim podacima, odnosno osiguravanju tajnosti podataka, ali to je u potpunosti moguće riješiti implementacijom sigurnosne politike o pravima pristupa podataka.

Dodatak A

OpenLayers index.html datoteka

```
1  <html xmlns="http://www.w3.org/1999/xhtml">
2  <head>
3    <style type="text/css">
4    #map {
5      float:left;
6      width: 80%;
7      height: 99%;
8      border: 2px solid gray;
9      }
10   #kontrole{
11     width:150px;
12     float:left;
13     }
14   body {
15     padding:0px;
16     margin:2px
17     }
18   #labs {
19     position:absolute;
20     bottom:15px;
21     left:7px;
22     font-size:smaller;
23     z-index: 5000;
24     }
25    ul{
26      margin:2px;
27      padding:0px;
28      }
29   </style>
```

```
30    <script src="/tc/OpenLayers-2.7/OpenLayers.js"></script>
31    <script type="text/javascript">
32      var result_style = OpenLayers.Util.applyDefaults({
33        strokeWidth: 5,
34        strokeColor: "#ff0000",
35        strokeOpacity: 0.6
36        }, OpenLayers.Feature.Vector.style['default']);
37
38      var start_style = OpenLayers.Util.applyDefaults({
39        externalGraphic: "/tc/start.png",
40        graphicWidth: 18,
41        graphicHeight: 26,
42        graphicYOffset: -26,
43        graphicOpacity: 0.7
44        }, OpenLayers.Feature.Vector.style['default']);
45
46      var stop_style = OpenLayers.Util.applyDefaults({
47        externalGraphic: "/tc/stop.png",
48        graphicWidth: 18,
49        graphicHeight: 26,
50        graphicYOffset: -26,
51        graphicOpacity: 0.5
52        }, OpenLayers.Feature.Vector.style['default']);
53
54      var SinglePoint = OpenLayers.Class.create();
55          SinglePoint.prototype = OpenLayers.Class.inherit(
56            OpenLayers.Handler.Point, {
57              createFeature: function(evt) {
58              this.control.layer.removeFeatures(
59                this.control.layer.features);
60              OpenLayers.Handler.Point.prototype.
61                createFeature.apply(this, arguments);
62              }
63          });
64
65      var map, parser, start, stop, result, controls;
66
67      function init(){
68        var options = {
69          units: 'm',
70          maxResolution: 1222.992452344,
71          maxExtent: new OpenLayers.Bounds(-20037508, -20037508,
```

```
72            20037508, 20037508.34),
73          restrictedExtent: new OpenLayers.Bounds(1422591, 5150000,
74            2404881,5898620),
75          numZoomLevels: 8,
76          controls:[new OpenLayers.Control.MouseDefaults()]
77          }
78
79        map = new OpenLayers.Map( 'map' , options);
80        layer = new OpenLayers.Layer.WMS( "podloga",
81          "tilecache.cgi?", {layers: 'osm', format: 'image/png' } );
82
83        result = new OpenLayers.Layer.Vector(
84          "Rezultat", {style: result_style});
85
86        start = new OpenLayers.Layer.Vector(
87          "Početna točka", {style: start_style});
88        stop = new OpenLayers.Layer.Vector
89         ("Završna točka", {style: stop_style});
90
91        map.addLayers([layer, result, start, stop]);
92
93        parser = new OpenLayers.Format.WKT();
94
95        map.addControl(new OpenLayers.Control.Permalink());
96        map.addControl(new OpenLayers.Control.MousePosition());
97        map.addControl(new OpenLayers.Control.ScaleLine());
98        map.addControl(new OpenLayers.Control.KeyboardDefaults());
99        map.addControl(new OpenLayers.Control.LayerSwitcher(
100         {'ascending':false}));
101       map.addControl( new OpenLayers.Control.PanZoomBar() );
102
103       controls = {
104        start:new OpenLayers.Control.DrawFeature(start,SinglePoint),
105        stop: new OpenLayers.Control.DrawFeature(stop, SinglePoint)
106           }
107       for (var key in controls) { map.addControl(controls[key]); }
108
109       if (!map.getCenter()) map.zoomToMaxExtent();
110
111     }
112
113     function toggleControl(element) {
```

```
114         for (key in controls) {
115           if (element.value == key && element.checked) {
116              controls[key].activate();
117           } else {
118              controls[key].deactivate();
119           }
120         }
121       }
122
123       function compute() {
124         var startPoint = start.features[0];
125         var stopPoint = stop.features[0];
126         if (startPoint && stopPoint) {
127            var result = {
128            startpoint: startPoint.geometry.x +
129              ' ' + startPoint.geometry.y,
130            finalpoint: stopPoint.geometry.x +
131              ' ' + stopPoint.geometry.y,
132            method: OpenLayers.Util.getElement('method').value,
133            };
134
135         OpenLayers.loadURL("/tc/getdata.cgi", result,
136           null, displayRoute,null);
137         }
138       }
139
140       function displayRoute(response) {
141         if (response && response.responseXML) {
142           result.removeFeatures(result.features);
143           var edges =
144             response.responseXML.getElementsByTagName('edge');
145           var features = [];
146           for (var i = 0; i < edges.length; i++) {
147             var g = parser.read(edges[i].
148               getElementsByTagName('wkt')[0].textContent);
149             features.push(g);
150           }
151           result.addFeatures(features);
152         }
153       }
154       </script>
155   </head>
```

```
156  <body onload="init()">
157  <div id="kontrole">
158    <ul>
159      <li>
160      <input type="radio" name="control" id="noneToggle"
161        onclick="toggleControl(this);" checked="checked" />
162      <label for="noneToggle">Navigacija</label>
163      </li>
164      <li>
165      <input type="radio" name="control" value="start"
166        id="startToggle" onclick="toggleControl(this);" />
167      <label for="startToggle">Početna<br/>točka</label>
168      </li>
169      <li>
170        <input type="radio" name="control" value="stop"
171          id="stopToggle" onclick="toggleControl(this);" />
172      <label for="stopToggle">Završna<br/>točka</label>
173      </li>
174    </ul>
175
176    Algoritam:<select id="method">
177      <option value="SPD">Dijkstra</option>
178      <option value="SPA">A Star</option>
179      <option value="SPS">Shooting Star</option>
180    </select>
181    <button onclick="compute()">Pronađi rutu</button>
182  </div>
183
184  <div id="map">
185  </div>
186  </body>
187  </html>
```

Dodatak B

Python getdata.cgi datoteka

```
1  #!/usr/bin/python
2  import cgi
3  form = cgi.FieldStorage()
4
5  import psycopg2
6  import sys
7
8  def findNearestEdge(lonlat):
9    conn=psycopg2.connect('host=localhost dbname=diplomski_rt')
10   sql = "SELECT gid, source, target, the_geom, distance(ST_\
11 Transform(the_geom,900913), GeometryFromText('POINT("+\
12 str(lonlat[0])+" "+str(lonlat[1])+")', 900913)) AS dist FROM ways\
13  WHERE ST_Transform(the_geom,900913) && setsrid('BOX3D("\
14 +(str(float(lonlat[0])-200))+" "+(str(float(lonlat[1])-200))+",\
15 "+(str(float(lonlat[0])+200))+" "+(str(float(lonlat[1])+200))+")'\
16 ::box3d, 900913) ORDER BY dist LIMIT 1"
17   cur=conn.cursor()
18   cur.execute(sql)
19   data=cur.fetchall()
20   if data:
21     edge={}
22     edge['gid']=data[0][0]
23     edge['source']=data[0][1]
24     edge['target']=data[0][2]
25     edge['the_geom']=data[0][3]
26     conn.close()
27     return edge
28   else:
29     print 'Content-type: text/xml\n\n<?xml version="1.0"\
```

```
30    encoding="UTF-8"?><route>\n</route>'
31       conn.close()
32       sys.exit(1)
33
34  pocetnatocka=form["startpoint"].value.split(' ')
35  krajnjatocka=form["finalpoint"].value.split(' ')
36
37  pocetnibrid=findNearestEdge(pocetnatocka)
38  krajnjibrid=findNearestEdge(krajnjatocka)
39
40  metoda=form["method"].value
41  sql=''
42
43  if metoda=='SPD':
44    sql = "SELECT rt.gid, AsText(st_transform(rt.the_geom,900913))\
45   AS wkt,length(rt.the_geom) AS length, ways.gid FROM ways,(SELECT\
46   gid, the_geom FROM dijkstra_sp_delta('ways',"+str(pocetnibrid\
47   ['source'])+","+str(krajnjibrid['target'])+",3000)) as rt\
48   WHERE ways.gid=rt.gid;"
49
50  elif metoda=='SPS':
51    sql = "SELECT rt.gid, AsText(st_transform(rt.the_geom,900913))\
52   AS wkt, length(rt.the_geom) AS length, ways.gid FROM ways,(\
53   SELECT gid, the_geom FROM shootingstar_sp('ways',"+str(\
54   pocetnibrid['gid'])+ ","+str(krajnjibrid['gid'])+", 5000,\
55  'length', true, true) ) as rt WHERE ways.gid=rt.gid;"
56
57  elif metoda=='SPA':
58    sql = "SELECT rt.gid, AsText(st_transform(rt.the_geom,900913))\
59   AS wkt, length(rt.the_geom) AS length, ways.gid FROM ways, (\
60   SELECT gid, the_geom FROM astar_sp_delta('ways',"+str(pocetnibrid\
61   ['source'])+","+str(krajnjibrid['target'])+",3000)) as rt WHERE\
62   ways.gid=rt.gid;"
63
64  conn=psycopg2.connect('host=localhost dbname=diplomski_rt')
65  cur=conn.cursor()
66  cur.execute(sql)
67  data=cur.fetchall()
68
69  xml  = 'Content-type: text/xml\n\n<?xml version="1.0"\
70   encoding="UTF-8"?>'
71  xml += "<route>\n"
```

```
72  i=1
73  pathlength=0.0
74  for dat in data:
75    pathlength += float(dat[2])
76    xml += "\t<edge id='"+str(i)+"'>\n"
77    xml += "\t\t<id>"+str(dat[3])+"</id>\n"
78    xml += "\t\t<wkt>"+str(dat[1])+"</wkt>\n"
79    xml += "\t\t<length>"+str(pathlength)+"</length>\n"
80    xml += "\t</edge>\n"
81    i+=1
82
83  xml += "</route>\n"
84
85  conn.close();
86  print xml
```

Bibliografija

Encyclopaedia Britannica. Space. `http://www.britannica.com/EBchecked/topic/557313/space`, 2009.

Iain Campbell. *Reliable Linux: Assuring High Availability.* Gearhead Press, 2002.

Adrijana Car. *Hierarchical Spatial Reasoning: Theoretical Consideration and its Application to Modeling Wayfinding.* PhD thesis, Department of Geoinformation Technical University Vienna, 1996.

Korry Douglas and Susan Douglas. *PostgreSQL.* Sams Publishing, 2006.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In *Systems Science and Cybernetics, IEEE Transactions Issue 2*, volume 4, pages 100–107, July 1968.

Mark Lutz. *Programming Python.* O'Reilly Media, Inc., 2006.

D. Pfoser N. Tryfona M. Raubal, M. J. Egenhofer. Structuring space with image schemata: Wayfinding in ariports as a case study. In Andrew U. Frank Stephen C. Hirtle, editor, *Spatial Information Theory: a theoretrical Basic for GIS*, October 1997.

Damir Medak, Boško Pribičević, Ivan Medved, and Dražen Odobašić. Usporedba komercijalnih i slobodnih sustava za upravljanje bazama prostornih podataka. In Boško Pribičević Damir Medak, Petar Nikolić, editor, *Zbornik radova, Treći hrvatski kongres o katastru s međunarodnim sudjelovanjem*, March 2005.

D. R. Montello. The perception and cognition of environmental distance. In Andrew U. Frank Stephen C. Hirtle, editor, *Spatial Information Theory: a theoretrical Basic for GIS*, October 1997.

Dražen Odobašić and Ivan Grčić. Razvoj open source geoinformacijskog sustava podržanog bazom prostornih podataka na internetu - rad nagrađen rektorovom nagradom, Travanj 2004.

Philippe Rigaux, Michel Scholl, and Agnès Voisard. *Spatial Databases: With Application to GIS*. Elsevier Science, 2002.

Wikipedia. Seven bridges of königsberg. `http://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg`, 2009a.

Wikipedia. Ontology. `http://en.wikipedia.org/wiki/Ontology`, 2009b.

Wikipedia. Wayfinding. `http://en.wikipedia.org/wiki/Wayfinding`, 2009c.

Michael Worboys and Matt Duckham. *GIS A computing perspective*. CRC Press, 2004.

D. Zubin. Natural language understanding and reference frames. pages 13–16, 1989.