Sadržaj specifikacije

1. Zahtevi uređaja

Uređaj opisan ovim dokumentom pronalazi optimalni saobraćajni put između dve lokacije na mapi. Mapa sa koje se biraju lokacije je ograničena na oblast grada Novog Sada.

Korisniku se na mapi Novog Sada daje mogućnost unošenja *polazne lokacije*, kao i lokacije na koju bi hteo da stigne motornim vozilom – *željene destinacije* (slika 1).

Slika 1 Unošenje dveju lokacija na mapi

Po završetku unosa, ukoliko postoji, na mapi se prikazuje optimalna putanja (slika 2). Uz mapu, na terminalu se ispisuje poruka o statusu izlaza uređaja. Ukoliko je korisnik uneo dve lokacije između kojih ne postoji putanja ili ukoliko se dve lokacije poklapaju, to će biti naznačeno porukom na terminalu. U poslednja dva slučaja se ništa ne prikazuje na mapi Novog Sada, osim unetih lokacija.

```
stdout:
Mreza Novog Sada se sastoji iz 9122 temena

...DONE!
-------------
Srecan put :)
-------------
Table deleted
Relaxation history deleted
Path deleted
```

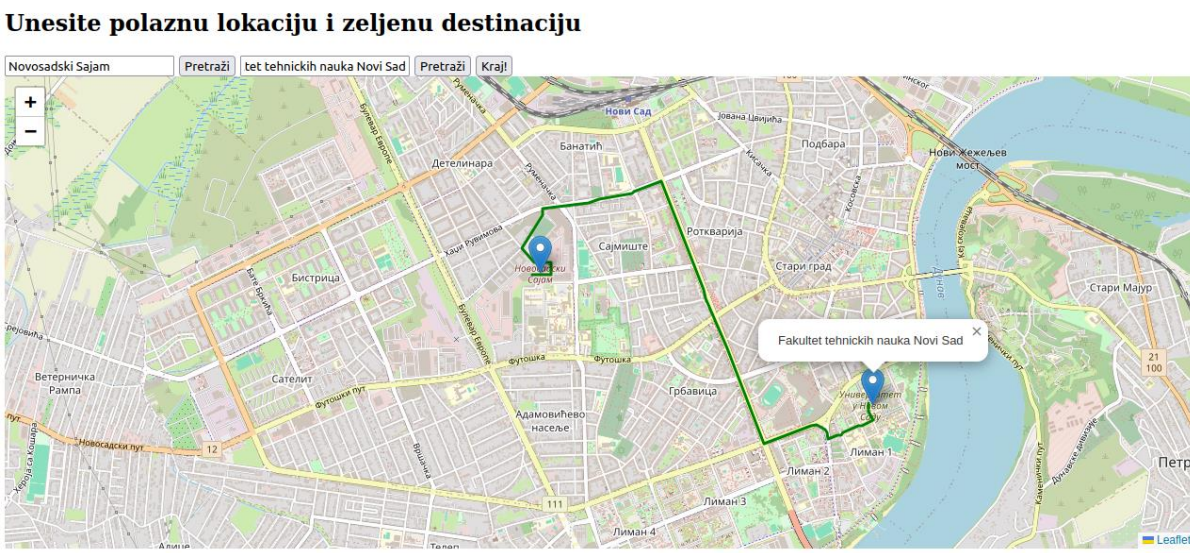
Bash listing 1 Ispis na terminalu ukoliko je putanja pronađena

```
stdout:
Mreza Novog Sada se sastoji iz 9122 temena

...DONE!
-------------------------------------------------
Saobracajnica izmedju unetih lokacija ne postoji.
-------------------------------------------------
Table deleted
Relaxation history deleted
Path deleted
```

Bash listing 2 Ispis na terminalu ukoliko putanja nije pronadjena

```
stdout:
Mreza Novog Sada se sastoji iz 9122 temena

...DONE!
----------------------------------------
Pocetna i krajnja lokacija su ista tacka.
----------------------------------------
Table deleted
Relaxation history deleted
Path deleted
```

Bash listing 3 Ispis na terminalu ukoliko se lokacije poklapaju



Slika 2 Na mapi se prikazuje najkraći saobraćajni put između unetih lokacija

1.1 Ograničenja sistema

Uređaj na kome se nalazi aplikacija je Zybo Z7 10 razvojna ploča. Centralna komponenta ploče je XC7Z010 SoC koji kombinuje ARM Cortex-A9 procesor i FPGA deo čipa.

1.1.1 Hardverska ograničenja

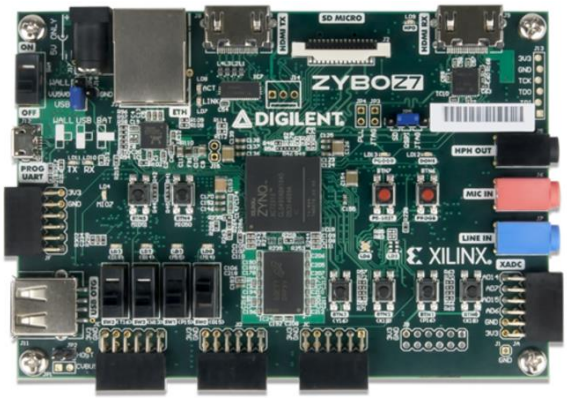
Složenost implementiranog dizajna u FPGA logici je ograničena brojem standardnih ćelija – tabela 1-1 [1]. Ovim je ograničena hardverska akceleracija. RAM memorija je DD3L memorija veličine 1GB sa brzinom do 533 MHz i 32-bitnom magistralom. Pored nje tu je i 16 MB Quad-SPI Flash memorija. Procesor poseduje 8 I/O pinova. FPGA deo poseduje 32 I/O pina. Uređaj se napaja pomoću USB porta ili bilo kojeg drugog spoljašnjeg izvora napajanja od 5V. Uređaj za hlađenje ne postoji.

Broj look-up tabela	17 600
Broj flipflop-ova	35 200
Blok RAM	270 KB

Tabela 1-1 FPGA resursi [1]

1.1.2 Softverska ograničenja

ARM dual core Cortex-A9 je dvojezgarni procesor koji radi na maksimalnoj frekvenciji od 667 MHz. Može da podrži operativni sistem poput PetaLinux-a. Zbog ograničenja ploče nije praktično pokretati operativne sisteme poput Ubuntu-a.



Slika 3 Zybo Z7 10 [1]

2. Teorijske osnove rada sistema

Mapa Novog Sada prikazana na slikama 1 i 2 je preuzeta sa sajta openstreetmap.org [2]. OpenStreetMap (OSM) je besplatna mapa celog sveta koju volonteri stalno dorađuju, dodajući nove proverene podatke postojećim mapama [3].

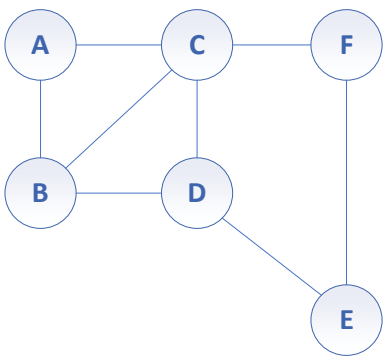
Ovakva mapa je puna detalja koji mogu otežati dalje objašnjenje funkcionalnosti, stoga će se nadalje mapa apstrahovati svojim karakteristikama bitnim za ovaj sistem. Te karakteristike se mogu podeliti u 3 velike grupe:

1. autoputevi i ulična mreža (putevi za motorna vozila)
2. lokacije po gradu povezane putevima za motorna vozila
3. dužina puteva i procenjeno zadržavanje na njima

Ovim se dobija uprošćena slika Novog Sada i **konceptualni model** [4] [5], prikladan za dalji opis sistema. Sistem na uličnoj mreži grada, za dve lokacije, pronalazi skup saobraćajnica koje u zbiru na sebi imaju što manje procenjeno zadržavanje.

U matematici se ovakva slika Novog Sada naziva graf. Graf je struktura sačinjena od temena (čvorova) međusobno povezanih ivicama (granama). Za mapu Novog Sada predstavljenu grafom, putevi za motorna vozila bi predstavljali ivice koje povezuju temena grafa – lokacije po gradu.

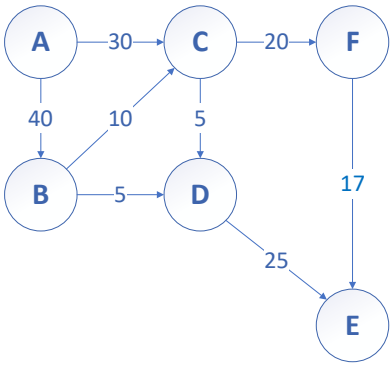
Graf je usmeren ako za svaku ivicu koja spaja dva temena, uopšteno govoreći, A i B, važi da ivica koja spaja teme A sa temenom B nije ekvivalentna ivici koja spaja teme B sa temenom A. Ovo je bitna karakteristika i značaj joj se vidi na primeru jednosmerne ulice.



Slika 4 Primer grafa

Treća grupa osobina se u teoriji grafova naziva težinama. Ne dolazi se jednako brzo od jedne lokacije do druge putujući sporednim ulicama ili putujući bulevarom. Graf kod koga su pridružene težine svakoj ivici se naziva težinski graf. Težine se ovde predstavljaju celobrojnim nenegativnim

brojevima. U slučaju modelovanja saobraćajnica, manje težine se daju putevima kroz koje se saobraćaj brže odvija. Ukoliko je težinski graf i usmeren, naziva se mreža [6].



Slika 5 Primer mreže za graf sa slike 3

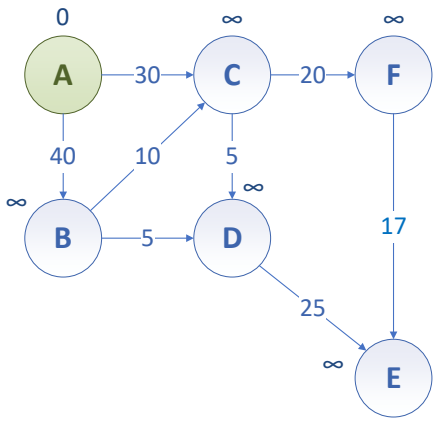
Dakle, **matematički model** sistema govori o mreži grada [4] [5]. Stoga je bitno pri unosu lokacija naglasiti koja je početna, a koja krajnja lokacija.

Problem određivanja optimalne putanje se svodi na određivanje skupa temena koji povezuju početno i krajnje teme. Ivice koje povezuju traženi skup temena imaju najmanju sumu težina i time čine optimalnu putanju.

2.1 Rešenje korišćenjem Dajkstrinog algoritma

Dajkstrin algoritam biće objašnjen na primeru sa slike 5. Podrazumeva se da se na samom početku primene algoritma poznaje mreža, kao i početno i krajnje teme za koje se algoritam primenjuje. Neka početno teme bude teme A, a krajnje teme E.

Inicijalno se sva temena postave u skup, tabelu neposećenih temena. Potom se svakom od temena dodaje promenljiva, tzv. udaljenost od početnog temena, po kojoj se sortiraju temena mreže, u neopadajućem poretku, tako da prvo teme unutar skupa bude uvek najbliže početnom temenu. Pošto se na početku sva temena nalaze unutar skupa neposećenih temena, najmanju udaljenost od početnog temena će imati upravo početno teme i ta udaljenost je 0. Ostalim temenima se na početku dodeljuje beskonačna udaljenost od početnog temena, ∞, kao znak da ta temena još nisu posećena i da put do njih još uvek nije poznat.



Slika 6 Početak Dajkstrinog algoritma

Skup neposećenih temena	
Teme	Udaljenost od početnog temena
A	0
B	∞
C	∞
D	∞
E	∞
F	∞

Tabela 2-1 Skup neposećenih temena na početku algoritma

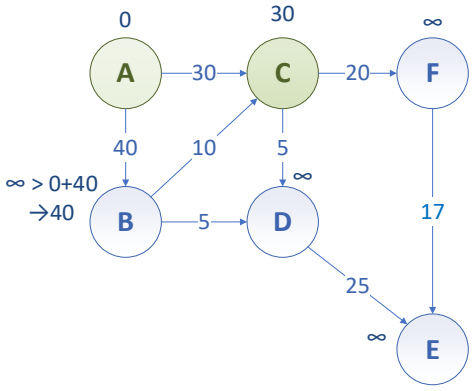
Sledeći deo algoritma se obavlja iterativno. Iz skupa neposećenih temena se uzima teme koje je na najmanjoj udaljenosti od početnog temena. U slučaju da u skupu neposećenih temena više nema temena ili u slučaju da se sva temena u skupu nalaze na beskonačnoj udaljenosti, došlo je do kraja

algoritma. U suprotnom, za trenutno izvučeno teme iz skupa neposećenih temena posmatraju se sve ivice koje vode od trenutnog temena ka susednim.

Za svaku od tih ivica, koja vodi od trenutnog temena do susednog, proverava se da li je udaljenost tog susednog temena od početnog temena veća od zbira udaljenosti trenutno izvučenog temena i težine ivice koja povezuje trenutno izvučeno teme i posmatrano susedno teme. Ukoliko je to slučaj, izvršava se proces relaksacije u kome dato susedno teme dobija novu udaljenost od početnog temena, jednaku zbiru udaljenosti trenutno izvučenog temena od početnog temena i težine ivice koja povezuje trenutno izvučeno teme i posmatrano susedno teme.

Nakon provere svih ivica koje vode od trenutno izvučenog temena, teme se trajno uklanja iz skupa, a skup se sortira u neopadajućem poretku. Prelazi se na sledeću iteraciju, s tim što se pamte sva temena koja su izvršila poslednje relaksacije, kao i temena nad kojima su te relaksacije izvršene (tabela 2-2).

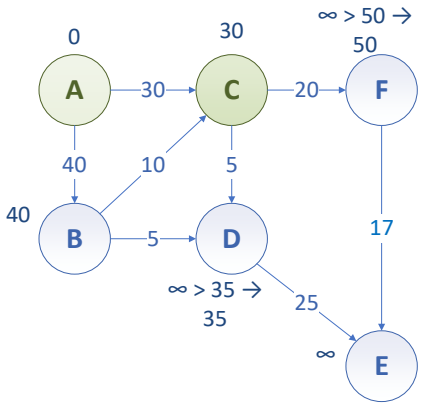
Praćenjem redosleda poslednjih relaksacija izvršenih nad konkretnim temenima, od krajnjeg (E) do početnog temena (A), dobijamo skup temena (za konkretan primer, prikazan tabelom 2-7) koji predstavlja optimalnu putanju od početnog do krajnjeg temena [7] [8] [9].



Slika 7 Stanje na mreži nakon izvlačenja temena A iz skupa neposećenih temena

Skup neposećenih temena		
Teme	Udaljenost od početnog temena	Relaksacije
C	30	A → C
B	40	A → B
D	∞	/
E	∞	/
F	∞	/

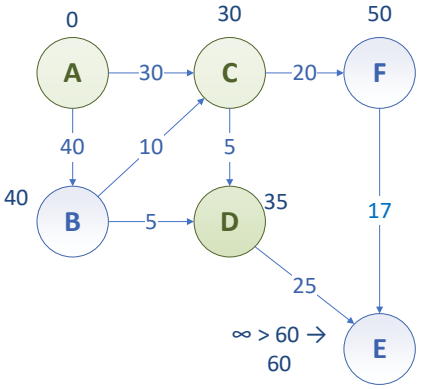
Tabela 2-2 Skup neposećenih temena nakon izvlačenja temena A



Slika 8 Stanje na mreži nakon izvlačenja temena C iz skupa neposećenih temena

Skup neposećenih temena		
Teme	Udaljenost od početnog temena	Relaksacije
D	35	C → D
B	40	A → B
F	50	C → F
E	∞	/

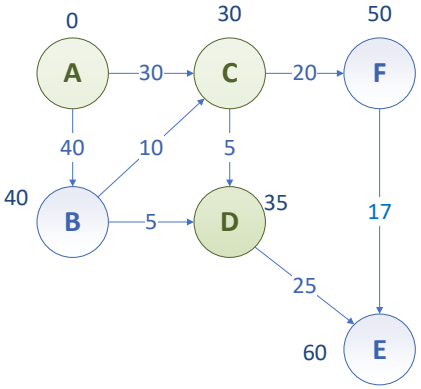
Tabela 2-3 Skup neposećenih temena nakon izvlačenja temena C



Slika 9 Stanje na mreži nakon izvlačenja temena D iz skupa neposećenih temena

Skup neposećenih temena		
Teme	Udaljenost od početnog temena	Relaksacije
B	40	A → B
F	50	C → F
E	60	D→ E

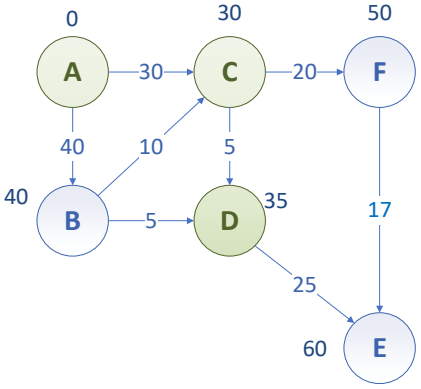
Tabela 2-4 Skup neposećenih temena nakon izvlačenja temena D



Slika 10 Stanje na mreži nakon izvlačenja temena B iz skupa neposećenih temena

Skup neposećenih temena		
Teme	Udaljenost od početnog temena	Relaksacije
F	50	C → F
E	60	D→ E

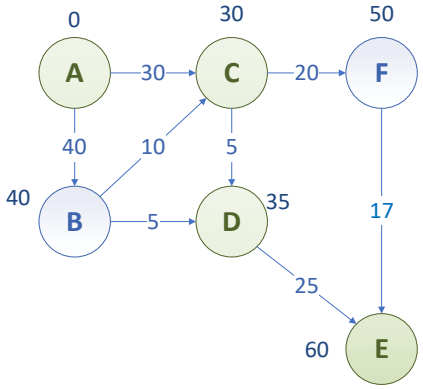
Tabela 2-5 Skup neposećenih temena nakon izvlačenja temena B



Slika 11 Stanje na mreži nakon izvlačenja temena F iz skupa neposećenih temena

Skup neposećenih temena		
Teme	Udaljenost od početnog temena	Relaksacije
E	60	D→ E

Tabela 2-6 Skup neposećenih temena nakon izvlačenja temena F



Slika 12 Stanje na mreži nakon izvlačenja temena E iz skupa neposećenih temena

Relaksacije
D → E
C → D
C → A

Tabela 2-7 Skup neposećenih temena nakon izvlačenja temena E, prikaz poslednjih relaksacija od E nazad do A

Dajkstrin algoritam se koristi i nad težinskim neusmerenim grafovima. Neusmerenu ivicu određene težine možemo uvek zameniti dvema usmerenim ivicama istih težina, međusobno suprotnog smera [9].

Slika 13 Navedeni grafovi su ekvivalentni

3. Izvršna specifikacija sistema

Izvršna specifikacija je modifikovani C++ kod. Kao softverski model, izvršna specifikacija je **sekvencijalna na osi konkurentnosti ESL taksonomije. Pristup izvornim fajlovima je moguć čineći sistem dizajnom (eng. Design) na osi konfigurabilnosti** [4] [5]. Originalni kod, smešten u RoutingKit direktorijum, može da se preuzme sa GitHub-a [10]. Modifikacije su izvršene prvenstveno zbog toga što originalni kod realizuje kontrakciju hijerarhija. To je algoritam baziran na preprocesing-u, što znači da postoji faza (eng. preprocessing phase) u kojoj se vrši uvođenje prečica u polazni graf. Suština algoritma je prikazana na slici 15.

Slika 14 Suština kontrakcije hijerarhija

Ukoliko je dat graf sa tri temena, A, B i C, i ako je polazno teme A, a krajnje C, težina puta od A do C preko B je 20, što je ekvivalentno grafu sačinjenom samo od temena A i C, međusobno povezanih ivicom težine 20. Ivica težine 20 je prečica početnog grafa sa tri temena, a teme B je kontrahovano teme [11] [12].

Početni graf zajedno sa prečicama se posle koristi u fazi u kojoj je poznato početno i krajnje teme (engl. query phase) tako što se na njega primenjuje Dajkstrin algoritam (obično bidirekcioni, što znači da se počinje od početnog temena u jednom smeru i od krajnjeg temena u drugom). Prečice se ubacuju onda kada je to potrebno [11].

Kontrakcija hijerarhija pruža značajno ubrzanje u odnosu na Dajkstrin algoritam i bolja je opcija za velike skupove podataka (eng. dataset), poput kontinentalne mreže puteva [13]. Sa ubrzanjem dolazi znatno kompleksniji model u izvršnoj specifikaciji i teže **održavanje modela**. U većini slučajeva **najjednostavniji model je najbolji** za konkretnu primenu [5]. Činjenica je da mreža Novog Sada predstavlja mnogo manji skup podataka od državne ili pak kontinentalne mreže, te je opravdano koristiti Dajkstrin algoritam.

Izvršna specifikacija sastoji se iz dva dela: `ocitavanje_grafa.cpp` i `main.cpp`. `main.cpp` poziva model Dajkstrinog algoritma, dat izvornim fajlovima `dijkstra_algorithm.hpp` i `dijkstra_algorithm.cpp`. Izvršna specifikacija radi sa celim brojevima i sa brojevima sa pokretnim zarezom čija je predstava u hardveru nepoznata (eng. **property** na osi podataka) [4].

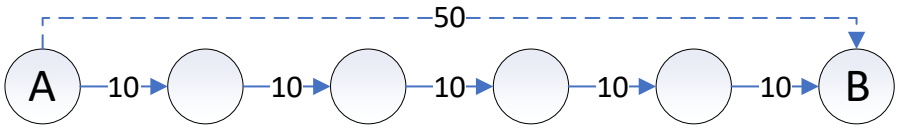
3.1 ocitavanje_grafa.cpp

Svi neophodni podaci za kreiranje mreže Novog Sada dobijaju se iz NoviSad.osm.pbf fajla preuzetog sa OpenStreetMap sajta [2]. PBF, skraćeno za Protocolbuffer Binary Format je alternativa XML fajlu. Ovde se koristi zbog toga što zauzima dosta manje memorijskog prostora i zbog toga što je PBF format brži za korišćenje. Osnovni gradivni blok podataka mape je čvor (eng. node). Čvor predstavlja tačku na Zemlji, određenu geografskom širinom i dužinom. U predstavi tačke na OpenStreetMap-i figurišu, pored geografske širine i dužine, i jedinstveni identifikacioni broj, **node** id, i različiti tagovi. Pomoću identifikacionog broja se čvor (tačka) referencira unutar OpenStreetMap skupa podataka (dataset-a), a tagovi služe da daju deskriptivnu informaciju o čvoru.

```
<node id="1831881213" lat="54.0900666" lon="12.2539381"
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>
</node>
```

Kod listing 1 Primer čvora sa dodatnim tagovima

Kod listing 1 ilustruje čvor sa dodatnim tagovima koji daju informaciju o imenu geografske tačke i o tome šta tačka predstavlja. Listing je predstavljen u XML formatu jer PBF format nije čitljiv. OpenStreetMap-a Novog Sada se sastoji od velikog broja čvorova. Nisu svi čvorovi od značaja pri formiranju saobraćajne mreže grada (neki čine obode zgrada, parkove ili stepenice, delove po kojima saobraćaj nije dozvoljen motornim vozilima). Izuzevši takve čvorove, podskup čvorova koji se razmatra u nastavku su modelujući čvorovi. Modelujući čvorovi služe za predstavljanje “geometrije mape”, nepravilne prirode puteva, reka i slično. Kada stvore sekvencu velikog broja uzastopnih čvorova na mapi, moguće je napraviti bilo koje zakrivljenje. Za samo formiranje grafa nisu potrebni svi modelujući čvorovi, već samo njihov podskup – rutirajući čvorovi. Objašnjenje za to se može ilustrovati na primeru krivog puta sa slike 16.



Slika 15 Nisu svi modelujući čvorovi i rutirajući

Neoznačeni čvorovi zajedno sa dva označena čvora, A i B sa slike 16, čine modelujuće čvorove. Za formiranje grafa od ovog dela mape je dovoljno koristiti samo čvorove A i B. Za podatak o težini između čvorova A i B uzeće se u obzir doprinos ivica koje spajaju ostale modelujuće čvorove i time će se informacija o ovom delu mape očuvati. Svakom rutirajućem čvoru se dodeljuje `ID`, kao vid identifikacije uokviru konkretne aplikacije. To je identifikacioni broj koji ne treba poistovetiti sa **node** id brojem koji imaju svi čvorovi OpenStreetMap-e Novog Sada.

3.1.1 Očitavanje OSM podataka

RoutingKit pruža jednostavan interfejs za "čitanje" PBF fajla, parsiranje bitnih podataka i formiranje mreže na osnovu njih [10]. Interfejs je zaglavlje `<routingkit/osm_simple.h>` u kom se nalazi funcija kojom se formira mreža saobraćajnica Novog Sada na osnovu priloženog PBF fajla. Ta funkcija je pozvana uokviru izvorne datoteke `ocitavanje_grafa.cpp`:

```
// Učitavamo OSM podatke (iz PBF fajla)
auto graph = simple_load_osm_car_routing_graph_from_pbf(BASE_DIR +
"data/NoviSad.osm.pbf");
```

Kod listing 2 Instanciranje grafa

U kod listingu 2, parametar `BASE_DIR + "data/NoviSad.osm.pbf"` je string koji sadrži putanju do PBF fajla. Povratna vrednost funkcije je mreža predstavljena strukturom `SimpleOSMCarRoutingGraph`, prikazanom kod listingom 3.

Rutirajući čvorovi su izabrani i predstavljaju temena grafa. Vektori `latitude` i `longitude` sadrže u sebi geografske širine i dužine temena grafa. Vektori imaju isti broj elemenata, jednak broju temena grafa. Svako teme grafa je numerisano identifikacionim brojem `ID`, nenegativnim celim brojem, počevši od 0. Indeks elementa u nizu(vektoru) odgovara `ID` vrednosti tog elementa.

Za teme sa identifikacionim brojem `ID` uređeni par (element vektora `latitude` sa indeksom `ID`, element vektora `longitude` sa indeksom `ID`) predstavlja georafske koordinate tog temena.

```
struct SimpleOSMCarRoutingGraph{
    std::vector<unsigned>first_out;
    std::vector<unsigned>head;
    std::vector<unsigned>travel_time;
    std::vector<unsigned>geo_distance;
    std::vector<float>latitude;
    std::vector<float>longitude;
    std::vector<unsigned>forbidden_turn_from_arc;
    std::vector<unsigned>forbidden_turn_to_arc;

    unsigned node_count() const {
        return first_out.size()-1;
    }

    unsigned arc_count() const{
        return head.size();
    }

};
```

Kod listing 3 Struktura koja predstavlja mrežu gradskih saobraćajnica

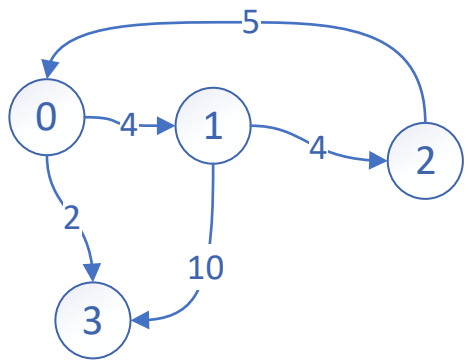
Vektori nenegativnih celobrojnih vrednosti `first_out` i `head` opisuju mrežu saobraćajnih puteva Novog Sada kao niz povezanosti (eng. adjacency-array).

3.1.1.1 Reprezentacija mreže u izvršnoj specifikaciji preko niza povezanosti

Svaka usmerena ivica povezuje dva temena grafa na takav način da iz jednog temena izvire, a u drugo teme ponire. Temena u koja ivice grafa poniru skladište se u vektor `head`. Broj elemenata vektora `head` jednak je broju ivica grafa. Ponorni čvorovi se skladište redom što znači da se skladište najpre `ID` vrednosti svih ponornih čvorova koji sa čvorom `ID == 0` formiraju usmerene ivice od čvora `ID == 0`; nakon toga svi ponorni čvorovi čvora `ID == 1` itd.

Broj elemenata vektora `first_out` je jednak broju temena uvećanom za 1. Njegov prvi elemement mora biti 0, a poslednji broj ivica grafa. Indeks elementa vektora `first_out` predstavlja identifikacioni broj `ID` temena grafa. Vrednost elementa vektora `first_out` je indeks elementa

vektora `head`. Element vektora `head` sa tim indeksom predstavlja prvo naredno teme po `ID` vrednosti koje je ivicom spojeno sa temenom čiju `ID` vrednost predstavlja trenutni indeks vektora `first_out`.



`first_out` prikazanog grafa

indeks	[0]	[1]	[2]	[3]	[4]
vrednost	0	2	4	5	5

`head` prikazanog grafa

indeks	[0]	[1]	[2]	[3]	[4]
vrednost	1	3	2	3	0

Ponorni čvorovi se skladište redom, najpre za teme `ID == 0`, a to su temena `ID == 1` i `ID == 3`

Slika 16 Primer ekstrakcije podataka sa mreže. Temena su određena svojim identifikacionim brojem ID ∈ [0, 3]

Na taj način se, pomoću `first_out` i `head` vektora, može jednoznačno rekonstruisati mreža. Ekstrakcija podataka o mreži u `head` i `first_out` vektor je prikazana na primeru grafa sa slike 17.

Ukoliko ne postoji nijedna ivica od nekog temena, kao što je to teme 3 (`ID == 3`) sa slike 17, tada je `first_out[i] == first_out[i+1]`, gde je `i` indeks elementa vektora `first_out`. U primeru sa slike 17 je `i == 3`.

3.1.1.2 Reprezentacija mreže u izvršnoj specifikaciji preko liste usmerenih ivica

Lista usmerenih ivica (eng. arc list) je jednostavnija predstava grafa. Graf je na ovaj način predstavljen pomoću već opisanog vektora `head` i vektora `tail`.

Vektor `tail`, `std::vector<unsigned> tail` je vektor koji u sebi sadrži izvore odgovarajućih usmerenih ivica. Time je svaka usmerena ivica određena uređenim parom (izvorno teme, ponorno teme). Izvori se smeštaju u vektor `tail`, a ponori u vektor `head`. Unutar izvornog fajla `ocitavanje_grafa.cpp`, vektor `tail` se formira pomoću vektora `first_out` iz strukture `SimpleOSMCarRoutingGraph`.

```
auto tail = invert_inverse_vector(graph.first_out);
```

Kod listing 4 Ekstrakcija izvornih temena mreže

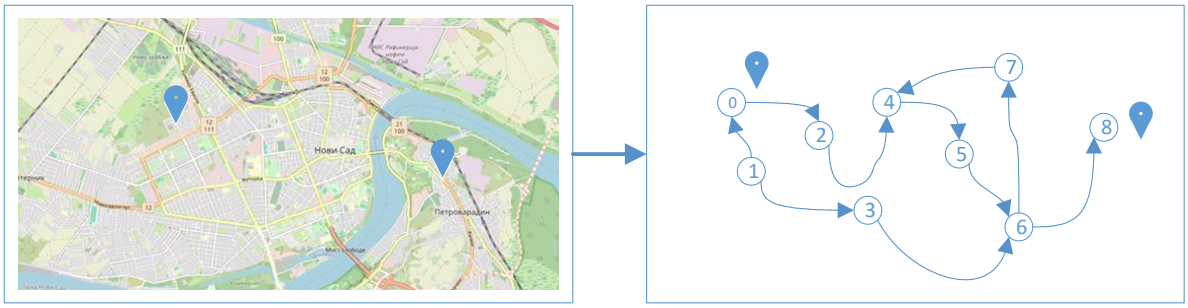
`invert_inverse_vector` funkcija može detaljnije da se pogleda uokviru `inverse_vector.h` zaglavlja [10].

3.1.1.3 Reprezentacija težina usmerenih ivica u mreži

Vektori `travel_time` i `geo_distance` su dve moguće opcije modelovanja težina usmerenih ivica mreže u izvršnoj specifikaciji. `travel_time` predstavlja vreme u milisekundama provedeno na odgovarajućoj ivici, dok je `geo_distance` dužinu ivice koja spaja dva susedna temena u metrima. Izvršna specifikacija koristi `travel_time` kao model težina usmerenih ivica jer on obuhvata i dužinu ivice i dozvoljenu brzinu na toj ivici.

3.1.2 Pretvaranje geografske pozicije u najbliže teme

Skup temena grafa je podskup skupa svih čvorova na OSM mapi, od kojih korisnik može da izabere dva čvora. Zbog toga će se često desiti da korisnik unese koordinate čvora koji ne predstavlja teme mreže Novog Sada. Ukoliko se unesu koordinate koje predstavljaju čvorove koji su ipak dovoljno blizu nekog od rutirajućih čvorova (temena grafa), trebalo bi da može da se pokrene upitnik o optimalnoj putanji (Dajkstrin algoritam). Dva temena na grafu koja su najbliža unetim čvorovima predstavljaće dve tačke između kojih se traži optimalna putanja.

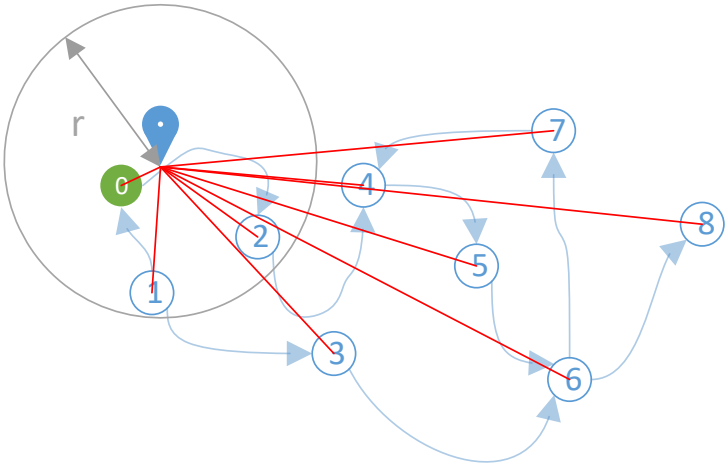


Slika 17 Pretvaranje geografske pozicije u najbliže teme

Dakle, potrebno je obezbediti proveru unetih čvorova i videti na kojoj su oni udaljenosti od temena formirane mreže. U izvršnoj specifikaciji se bira da udaljenost `r` bude neka granična vrednost. Ukoliko se ne pronađe nijedno teme unutar kruga poluprečnika `r` sa centrom u tački određenoj unetim koordinatama smatra se da uneti čvor nije validan i program se završava. Ukoliko temena postoje, za teme grafa koje će predstavljati početno ili konačno teme u modelu Dajkstrinog algoritma se uzima najbliže teme centru kruga poluprečnika `r`. Ove pretrage najbližeg temena se nazivaju i pretrage najbližeg suseda (eng. nearest neighbor searches) [14].

3.1.2.1 Linearna pretraga (naivni pristup)

Pretragu je moguće sprovesti tako što bi se poredile udaljenosti između svakog temena grafa sa unetim čvorom (na slici 19 označene crvenom bojom).



Slika 18 Ilustracija naivnog pristupa za unetu lokaciju. Teme `ID == 0` je najbliže unetoj lokaciji

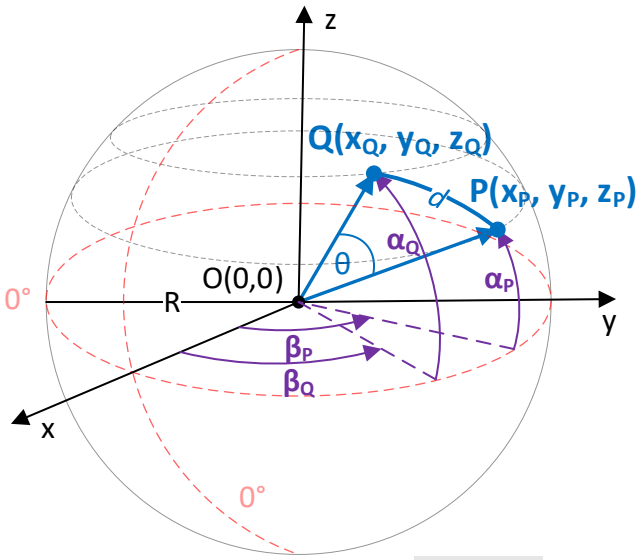
Postupak je poznat i kao naivni pristup. Model naivnog pristupa u izvršnoj specifikaciji je dat kod listingom 5, koji se nalazi uokviru izvorne datoteke `geo_position_to_node.cpp`.

```
for(unsigned i = 0; i < vertex_count; i++)
{
     distance_to_pivot = geo_dist(user_node_latitude,
user_node_longitude, graph_latitudes[i], graph_longitudes[i]);
     if(distance_to_pivot < min_distance && distance_to_pivot <= radius)
     {
          min_distance = distance_to_pivot;
          node_id = i;
     }
}
```

Kod listing 5 Naivni pristup

`vertex_count` je broj temena mreže. `geo_dist` funkcija računa udaljenost između svakog temena grafa, određenog preko svojih koordinta – (`graph_latitudes[i]`, `graph_longitudes[i]`) i

koordinata koje je korisnik uneo – (`user_node_latitude`, `user_node_longitude`). Fukncija uzima u obzir poluprečnik Zemlje. Udaljenost između dve tačke na Zemljinoj kugli, *d*, je određena jednačinama (7) – (9).



Slika 19 Teorijske osnove funkcije `geo_dist`

β_P, β_Q – geografske dužine u stepenima. Meridijan na 0° na slici 20 je označen crvenom bojom.

α_P, α_Q – geografske širine u stepenima. Paralela na 0° na slici 20 je označena crvenom bojom.

Θ – ugao između vektora određenih tačkama P i Q

$$x_P = R\cos\alpha_P\cos\beta_P \quad (1)$$
$$y_P = R\cos\alpha_P\sin\beta_P \quad (2)$$
$$z_P = R\sin\alpha_P \quad (3)$$
$$x_Q = R\cos\alpha_Q\cos\beta_Q \quad (4)$$
$$y_Q = R\cos\alpha_Q\sin\beta_Q \quad (5)$$
$$z_Q = R\sin\alpha_Q \quad (6)$$

$$d = R\theta \quad (7)$$

$$cos\theta = \frac{\vec{P}\cdot\vec{Q}}{\left|\vec{P}\cdot\vec{Q}\right|}, \vec{P} = x_P\vec{\imath_x} + y_P\vec{\imath_y} + z_P\vec{\imath_z}, \vec{Q} = x_Q\vec{\imath_x} + y_Q\vec{\imath_y} + z_Q\vec{\imath_z} \quad (8)$$

$$d = R\cos^{-1}\left(\cos\alpha_P\cos\beta_P\cos\alpha_Q\cos\beta_Q + \cos\alpha_P\sin\beta_P\cos\alpha_Q\sin\beta_Q + \sin\alpha_P\sin\alpha_Q\right) \quad (9)$$

Novi Sad sadrži `9122` temena, što je vidljivo ispisivanjem `vertex_count` vrednosti na terminal (bash listinzi 1 – 3).

Koristeći naivni pristup, pri svakom unosu lokacija izvršna specifikacija preračunava `9122` udaljenosti temena grafa od unetih lokacija. Ideja tzv. razdvajanjem prostora je suziti polje pretrage i time omogućiti efikasniju pretragu temena koje je najbliži sused unetoj lokaciji [14]. Obično se od temena grafa formira struktura čiji su susedni elementi u strukturi ujedno i susedni elementi u (metričkom) prostoru. RoutingKit koristi VP-stablo (eng. Vantage-Point tree) [10]. Struktura se formira tako što se proizvoljno teme proglasi za tačku od interesa (eng. vantage-point). Proračunaju je udaljenosti svakog temena od tačke od interesa i odredi se medijana. Za računanje udaljenosti se koristi ista funkcija kao i u naivnom pristupu, `geo_dist`. Sva temena koja su na udaljenosti manjoj od medijane pripadaće levom podstablu, dok će sva temena na udaljenosti većoj od tačke od interesa pripadati desnom podstablu. Ovim se efektivno skup podataka podelio na dva dela: temena bliža tački od interesa i temena dalja od tačke od interesa. Postupak se ponavlja za oba podskupa sve dok se ne dostigne maksimalan dozvoljen broj članova stabla koji je bez potomaka (bez daljeg grananja). Ovo je dodatna preferenca kojom se poboljšava pretraga [15][16]. Ideja je ipak koristiti naivni pristup, jer je njegova potencijalna implementacija u hardveru lakša nego potencijalna implementacija rekurzivne funkcije koja formira VP stablo u RoutingKit-u [10]. Potrebno je poštovati vremenska ograničenja izrade sistema, eng. **time to market**.

3.2 main.cpp

`main.cpp` izvorna datoteka poziva funkcije definisane uokviru `dijkstra_algorithm.hpp` zaglavlja i odgovarajučeg `dijkstra_algorithm.cpp` izvornog fajla.

Prateći opis algoritma dat u poglavlju 2.1 Rešenje korišćenjem Dajkstrinog algoritma, `dijkstra_algorithm.hpp` koristi tri jednostruko spregnute liste kojima pokriva sve ključne tačke algoritma: skup (tabelu) neposećenih temena, istoriju relaksacija i putanju koju nudi Dajkstrin algoritam kao optimalnu. Zaglavlje i izvorni fajl napisani su u C-ovskom maniru, čvrsto se oslanjajući na knjigu profesora dr Aleksandra Kupusinca [17].

```
typedef struct vertex_st
{
    unsigned ID;
    unsigned cost_from_start_vertex;
    struct vertex_st *next;
}VERTEX;
```

Kod listing 6 Predstavljanje temena

U kod listingu 6, izneta je predstava temena unutar skupa neposećenih temena. Umesto slova, kao u primeru pokrivenom u poglavlju 2.1, izvršna specifikacija koristi celobrojne vrednosti `ID` kao vid identifikacije. `cost_from_start_vertex` je udaljenost od početnog temena[1]. U poglavlju 2.1. je inicijalno ova vrednost beskonačna za sva temena osim početnog. Beskonačnost se modeluje korišćenjem velikog nenegativnog celog broja, `const unsigned inf_weight = 2147483647u`, za sva neposećena temena osim početnog, čija vrednost `cost_from_start_vertex` je jednaka 0.

Istorija relaksacija se čuva u jednostruko spregnutoj listi, gde je jedan element liste (jedna relaksacija) opisan strukturom `RELAXATION`. Relaksaciju izvršava teme sa indentifikacionim brojem `relaxing` nad temenom identifikacionog broja `relaxed`.

```
typedef struct relaxation_st
{
    unsigned relaxing;
    unsigned relaxed;
    struct relaxation_st *next;
}RELAXATION;
```

Kod listing 7 Predstavljanje relaksacije

Ključni deo izvršne specifikacije je modelovanje samog procesa relaksacije udaljenosti temena mreže od početnog temena. Funkcija `relaxationProcess` prolazi kroz skup neposećenih temena i bira ono teme koje trenutno ima najmanju udaljenost od početnog temena. Proverava se da li trenutno teme može da relaksira susede sa kojima je u direktnoj vezi preko ivice. Ukoliko je zbir težine ivice i udaljenosti trenutnog temena od početnog temena manji od udaljenosti suseda od početnog temena, udaljenost suseda se ažurira na ovaj zbir (kod listing 8).

```
if(temp != NULL && temp -> cost_from_start_vertex > edge_weights[i] +
(temp_min) -> cost_from_start_vertex)
{
    temp -> cost_from_start_vertex = edge_weights[i] + (temp_min) ->
cost_from_start_vertex;
    addToRelaxationHistory(prel_head, (temp_min) -> ID, temp -> ID);
}
```

Kod listing 8 Segment funkcije `relaxationProcess`

[1] Promenljivom `cost_from_start_vertex` se zapravo opisuje cena (eng. cost) koja se uloži prilikom dolaska iz početnog u trenutno teme – utrošeni resursi. Ta cena može biti apstrakcija pređenog puta, pa se zapravo govori o fizičkoj udaljenosti, ili pak vremena utrošenog da se dođe do trenutnog temena.

Proces relaksacije je gotov onog trenutka kada je tabela neposećenih temena prazna, ili kada je udaljenost svakog temena u tabeli od početnog temena beskonačna (jednaka vrednosti `inf_weight`). Na osnovu istorije relaksacija, početnog i krajnjeg temena, pozivom funkcije `parsePath` formira se Dajkstrina optimalna putanja kao jednostruko spregnuta lista, čiji su elementi strukture `PATH`.

```
typedef struct path_st
{
    unsigned ID;
    struct path_st *next;
}PATH;
```

Kod listing 9 Element Dajkstrine optimalne putanje

Element tipa `PATH` sadrži samo identifikacioni broj čvora koji pripada putanji. Na osnovu njega se lako pristupa skupu koordinata temena, vektorima `graph_latitudes` i `graph_longitudes` unutar `main`-a. Ovim se poznaje skup koordinata koje čine putanju. Taj skup se prikazuje na mapi Novog Sada (slika 2).

4. Literatura

[1] Zybo Z7 10: https://digilent.com/reference/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf [decembar 2024]

[2] OpenStreetMap: www.openstreetmap.org [novembar 2024]

[3] O OpenStreetMap-u i udruženju: https://wiki.openstreetmap.org/wiki/About_OpenStreetMap [novembar 2024]

[4] Grant Martin, Brian Bailey, Andrew Piziali: *ESL Design and Verification: A Prescription for Electronic System Level Methodology (Systems on Silicon)*

[5] Brian Bailey, Grant Martin: *ESL Models and their Application: Electronic System Level Design and Verification in Practice*

[6] Wikipedia, članak o teoriji grafova: https://sh.wikipedia.org/wiki/Teorija_grafova [novembar 2024]

[7] Wikipedia, članak o Dajkstrinom algoritmu: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm [novembar 2024]

[8] *Shortest Path Algorithms Explained (Dijkstra's & Bellman-Ford):* https://www.youtube.com/watch?v=j0OUwduDOS0 [novembar 2024]

[9] Predavanje Abdula Barija o Dajkstrinom algoritmu: https://www.youtube.com/watch?v=XB4MIexjvY0 [novembar 2024]

[10] RoutingKit: https://github.com/RoutingKit/RoutingKit/tree/master [novembar 2024]

[11] Wikipedia, članak o kontrakciji hijerarhija: https://sr.wikipedia.org/srec/Kontrakcija_hijerarhija [novembar 2024]

[12] *Single agent search video 45: Contraction Hierarchies*: https://www.youtube.com/watch?v=FXYs-fg1y8s [novembar 2024]

[13] *ESA.7.4 Lower Bounds and Approximation Algorithms for Search Space Sizes in Contraction Hierarchies:* https://www.youtube.com/watch?v=3lMruK5e3uE&t=280s [novembar 2024]

[14] Wikipedia, članak o pretrazi najbližeg suseda: https://en.wikipedia.org/wiki/Nearest_neighbor_search [novembar 2024]

[15] *VPPS-ART: An Efficient Implementation of Fixed-Size-Candidate-Set Adaptive Random Testing using Vantage Point Partitioning Strategy*, Rubing Huang, Senior Member, IEEE, Chenhui Cui, Dave Towey, Senior Member, IEEE, Weifeng Sun, Junlong Lian, Maj 2021.

[16] Odnos lokacija upita i članova VP-stabla: https://everyhue.me/posts/similarity-search-101-with-vantage-point-trees [novembar 2024]

[17] Kupusinac, A. : *Programski jezik C++*, Novi Sad: Fakultet tehničkih nauka, 2020.