Rješavanje problema vremenski najkraćeg puta

Dumančić, Ante

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

Ante Dumančić

Rješavanje problema vremenski najkraćeg puta

Završni rad

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET PROMETNIH ZNANOSTI

Ante Dumančić

Rješavanje problema vremenski najkraćeg puta

Solving time-dependent shortest path problem

Mentor: dr. sc. Tomislav Erdelić

Student: Ante Dumančić

JMBAG: 0135259857

Zagreb, 2023.

Sažetak:

U ovom završnom radu rješava se problem vremenskog rutiranja u gradu Zagrebu. Nagli porast broja vozila zahtjeva pravilno usmjeravanje prometa i optimizaciju ruta vozila. Rad se fokusira na pronalaženje vremenski najkraćih ruta za vozilo koristeći prikupljene podatke. Cestovna mreža grada Zagreba modelirana je grafom te su primijenjeni algoritmi poput Dijkstrinog algoritma i Fibonaccijeve hrpe za pronalazak vremenski najkraćeg puta. Izrađeno je grafičko korisničko sučelje u Visual Studio-u koje omogućuje interaktivni prikaz rezultata rutiranja na cestovnoj mreži u gradu Zagrebu, prikazujući najbolje rute na karti Open Street Map.

KLJUČNE RIJEČI: grafičko korisničko sučelje; ruta; Dijkstrin algoritam; Fibonaccijeva hrpa

Summary:

This final project addresses the problem of time-based routing in city of Zagreb. The rapid increase in the number of vehicles requires proper traffic management and route optimization. The project focuses on finding the shortest time routes for vehicles using collected data. The road network of the city of Zagreb is modeled by a graph and algorithms such as Dijkstra's algorithm and Fibonacci heap were applied to find the shortest path in time. A graphical user interface was created in Visual Studio that enables the interactive display of routing results on the road network in the city of Zagreb, showing the best routes on the Open Street Map.

KEY WORDS: graphical user interface; route; Dijkstra algorithm; Fibonacci heap

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI
ODBOR ZA ZAVRŠNI RAD

Zagreb, 5. svibnja 2023.

Zavod: **Zavod za inteligentne transportne sustave**
Predmet: **Algoritmi i programiranje**

ZAVRŠNI ZADATAK br. 7170

Pristupnik: **Ante Dumančić (0135259857)**
Studij: Inteligentni transportni sustavi i logistika
Smjer: Inteligentni transportni sustavi

Zadatak: **Rješavanje problema vremenski najkraćeg puta**

Opis zadatka:

Cilj rada je izraditi programsku podršku za rješavanje problema vremenski najkraćeg puta na mreži. Prvo je potrebno proučiti podatke o cestovnim segmentima u digitalnoj karti Republike Hrvatske zajedno s njihovim profilima brzina, spremljenih u tekstualnoj datoteci. Potom je potrebno učitati podatke u aplikaciju i kreirati graf cestovne mreže na temelju podataka u datoteci. Nakon kreiranja grafa, potrebno je implementirati neki od algoritama za pronalazak prostorno najkraćeg puta na grafu. Potom je nakon provjere rada algoritma, potrebno modificirati algoritam da na temelju profila brzina za svaki cestovni segment u digitalnoj karti, ovisno o odabranom trenutku polaska računa vremenski najkraći put na cestovnoj mreži. Dodatno, za odabir početnih parametara i prikaz rješenja problema potrebno je izraditi interaktivno grafičko sučelje s pozadinskom kartom.

Mentor:

Tomislav Erdelić

dr. sc. Tomislav Erdelić

Predsjednik povjerenstva za završni ispit:

SADRŽAJ

1. UVOD

Vremenski ovisno rutiranje je strategija koja se koristi u prometu kako bi se optimiziralo kretanje vozila, optimizacija se provodi u stvarnom vremenu sa stvarnim uvjetima u prometu. Glavni faktori u prometu koji služe za optimizaciju su gustoća prometa, brzina kretanja vozila, prometne nesreće, vrijeme putovanja i kapacitet prometne mreže. Jedna od zadaća inteligentnih transportnih sustava (ITS) je vezana uz informiranje korisnika o stanju u prometu. Takve informacije su od velike pomoći prilikom optimizacije ruta u velikim gradovima. ITS rješenja koriste senzore, kamere, mobilne aplikacije i druge tehnologije kako bi skupili podatke o stanju u prometu a zatim ih pružili krajnjim korisnicima. Glavna prednost vremenski ovisnog rutiranja je mogućnost izbjegavanja „rush-hour"-a. Vozačima se daju alternativne rute koje skraćuju vrijeme putovanja, a samim time i smanjuju zagađenje u gradovima. Razlika između običnog vremenskog rutiranja i vremenski ovisnog rutiranja leži u pristupu prikupljenim podatcima i analizi podataka o prometu. Obično vremensko rutiranje koristi statičke podatke poput prosječne brzine kretanja vozila na određenim dionicama, dok vremenski ovisno rutiranje koristi stvarne i trenutne podatke koji se prikupljaju u prometu. U ovom radu je korištena metoda vremenskog rutiranja koja ne koristi stvarne podatke, ali koristi vremenski promjenjive prosječne brzine na prometnicama, [1].

Početak problema za pronalazak najbolje rute može se povezati s problemom Königsberških mostova i matematičarom Leonhardom Eulerom. Leonhard Euler je 1736. godine prikazao Königsberške mostove kao graf te je pokušavao izračunati dali je moguće krenuti s jedne strane, prijeći sve mostove jednom te završiti na drugoj strani. Poslije su matematičari pokušavali riješiti problem kineskog poštara, problem pronalaska najkraćeg puta u željeznici, problem najkraćeg puta u gradu i druge. 1956. godine nizozemski znanstvenik Edsger Wybe Dijkstra je stvorio algoritam koji pretražuje i pronalazi najkraći put u grafu, [2]. Danas se taj algoritam zove Dijkstrinim algoritmom te je on jedan od najboljih i najpouzdanijih algoritama koji se koristi u izračunavanju najbolje rute vozila u nekom gradu.

U ovom završnom radu je obrađena tema rješavanja vremenski najkraćeg puta u Windows Forms aplikaciji, pri čemu je kod pisan C# programskim jezikom. Da bi se pronašao vremenski najkraći put u gradu Zagrebu koristi se modificirani Dijkstrin algoritam. Uz Dijkstrin algoritam koristi se i Fibonacciieva hrpa kako bi se ubrzao odabir sljedećeg vrha u grafu. Da bi se koristio Dijsktrin algoritam potrebno je imati graf (listu čvorova i linkova) na kojem se može odrediti najkraća ruta. Za stvaranje grafa u ovom radu korišteni su podatci prikupljeni od strane tvrtke Mireo (u tim podatcima nalazimo popis linkova, popis susjedni linkova, prosječne brzine u petominutnim intervalima i mnoge druge podatke). Rezultat Dijkstrinog algoritma, to jest popis linkova koji tvore najbržu rutu u gradu Zagrebu se prikazuje na grafičkom sučelju i Open Street Map-u.

Rad se sastoji od šest poglavlja, a to su:

1. Uvod
2. Opis i pohrana podataka
3. Graf prometne mreže i algoritam za pronalazak najkraćeg puta na grafu

4. Grafičko korisničko sučelje za prikaz rute i podataka
5. Zaključak

U drugom poglavlju je opisan sastav podatka (kako su podatci podijeljeni) i postupak pohrane podataka koji su preuzeti od tvrtke Mireo d.d. Opisan je i link -214700 i njegov apsolutni profil brzine u petominutnim intervalima.

U trećem poglavlju je opisana teorija grafova, što su to grafovi, koji sve grafovi postoje, što je to Dijkstra algoritam i kako on radi, što je to Fibonaccijeva hrpa, na kojem principu ona radi i glavne operacije Fibonaccijeva hrpe. Dodatno, opisano je i vrijeme trajanja programa uz korištenje samog Dijkstrinog algoritma i vrijeme trajanja programa uz korištenje Dijkstirnog algoritma i Fibonaccijeve hrpe.

U četvrtom poglavlju ovog rada se opisuje izgled grafičkog korisničkog sučelja i opisuju se NuGet paketi koji su se koristili u ovom radu. Ukratko je opisan postupak korištenja programa. Opisana je kompleksnost koda, dijagram klasa i rezultati koji se dobiju pokretanjem programa.

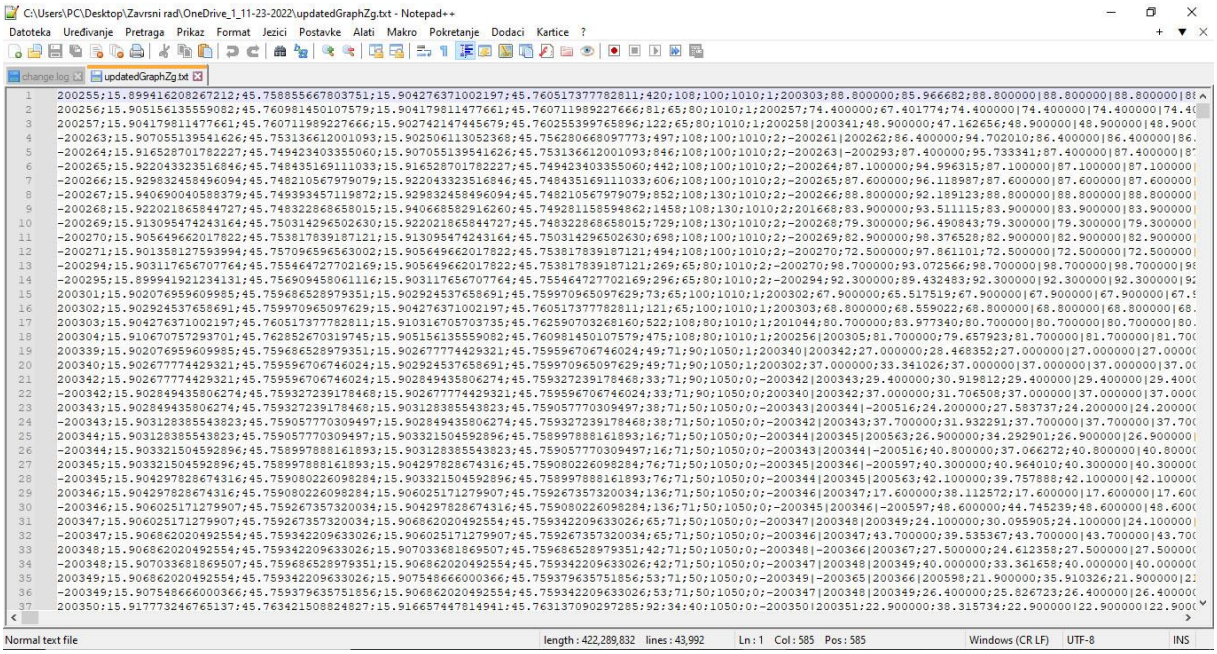
U petom poglavlju je dan zaključak završnog rada.

2. Opis i pohrana podataka

Podaci koji su korišteni prilikom izrade ovog završnog rada prikupljeni su od strane tvrtke Mireo d.d. Valja napomenuti da su ovi podatci stari nekoliko godina, te da su ti podaci prikupljeni za potrebe projekta SORDITO.

Projekt SORDITO (Sustav za optimizaciju ruta u dinamičkom transportnom okruženju) je projekt koji je sufinancirala Europska unija, a partner je bio Mireo d.d. Ovaj projekt je započeo 2014. godine te je trajao do 2016. godine. Cilj projekta je bio korištenjem GPS sustava i radarskih detektora razviti algoritam koji će pronalaziti najbolju rutu u gradu Zagrebu, [3]. Tvrtka Mireo d.d je hrvatska softverska firma koja se bavi izradom GPS navigacijskih sustava. Razvijen GPS nacigacijski sustav može pratiti veliki broj vozila u stvarnom vremenu, [4].

Podaci su spremljeni u tekstualnoj datoteci naziva „updatedGraphZg“. Za potrebe ovog završnog rada je korišten program Notepad++ (slika 1.) za otvaranje datoteke veće količine podataka.



Slika 1. Prikaz podataka u Notepad++ programu
Izvor: izradio autor

Svaki redak u datoteci „updatedGraphZg“ predstavlja jedan cestovni segment u mreži (link) s njegovim podatcima. Korišteni podatci su strukturirani u CSV formatu (eng. Coma Seperated Values). U svakom redu se nalazi preko petnaest vrijednosti koje su odvojene s točka-zarezom (;), a te vrijednosti su redom prikazane u tablici 1. Ukupan broj cestovni segmenata u datoteci je 43,992, datoteka zauzima 550 megabajta. Izgled cestovne mreže je prikazan na slici 2, ovaj je podatak bitan jer govori koliko je velik graf.

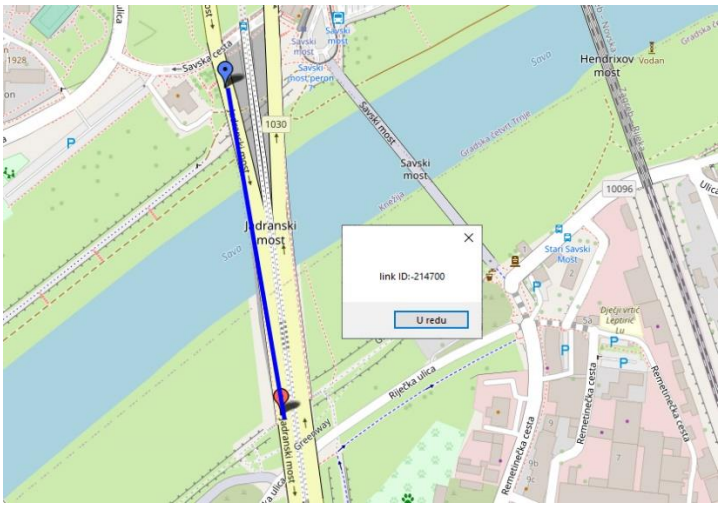
Slika 2. Izgled cestovne mreže
Izvor: Izradio autor

Tablica 1. Opis atributa linka

Atribut	Opis
ID	ID cestovnog segmenta (cijeli broj)
XB	Geografska duljina početne točke linka (B)
YB	Geografska širina početne točke linka (B)
XE	Geografska duljina početne točke linka (E)
YE	Geografska širina početne točke linka (E)
L	Duljina linka u metrima
v_m	Brzina linka u km/h koju je postavio Mireo (statična brzina)
v_{og}	Ograničenje brzine na linku koje je postavio Mireo u km/h
T	Tip linka: 1010 su autoceste, a 1080 kvartovske male ceste, te gradacija ide po 10 između tih vrijednosti
Zastavica smjera	Moguće vrijednosti 0,1,2 ili 3. 0 – po linku se može putovati u dva smjera te– u tom slučaju u datoteci postoje dva retka za link, jedan za pozitivni i jedan za negativni ID linka. Primjer link 215380, odnosno – 215380 1 – po linku se putuje u jednom smjeru od, točke b prema točci E, a ID linka je pozitivna vrijednost

	2- po linku se putuje u jednom smjeru od točke E prema točki B, a ID link je negativne vrijednosti, 3- sve isto kao i za vrijednost 0, ali 3 predstavlja da je cesta zbog nekog razloga zatvorena
Susjedni linkovi	Ovo je bitno za formiranje grafa cestovne mreže. Iz jednog linka se može dalje ići na niti jedan link, na jedan link ili na više njih. Znači između dvije susjedne točke-zarez (;) nalazi se lista susjednih linkova, pri čemu su ID-evi linkova međusobno odvojeni vertikalnom crtom (|)
v_{free}	Brzina slobodnog toka izračunata iz projekta SORDITO u km/h
v_{avg}	Prosječna brzina na linku izračunata iz projekta SORDITO u km/h
Profil brzine u km/h	Između dvije susjedne točke-zarez (;) nalazi se 288 vrijednosti brzina međusobno odvojenih vertikalnom crtom (|). Svaka od vrijednosti predstavlja prosječnu brzinu unutar pojedinog petominutnog intervala. Prva vrijednost predstavlja brzinu unutar intervala 00:00-00:05, druga unutar intervala 00:05-00:10, a zadnja unutar intervala 23:55-00:00.

Na slici 3 se vidi primjer linka -214700 Jadranski most. Link -214700 je dug 326 metara, ima ograničenu brzinu na 60 km/h, po njemu se putuje u samo jednom smjeru, i on ima samo jedan susjedni link -214699.



Slika 3. Izgled linka -214700

Izvor: izradio autor

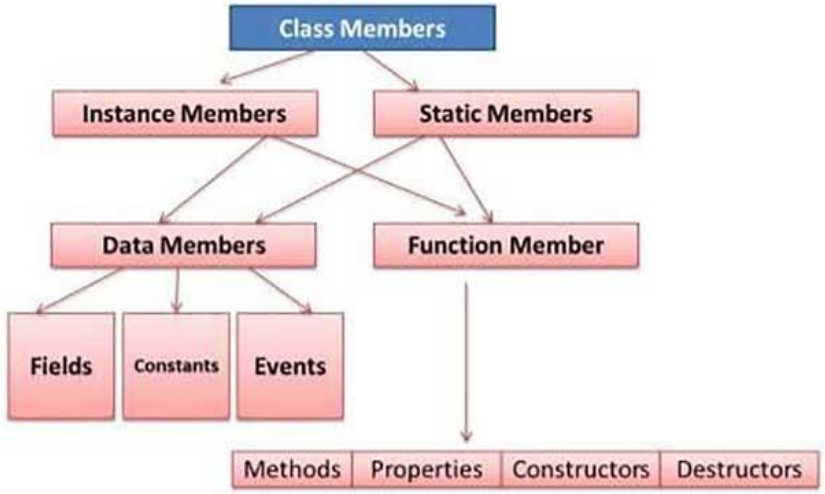
Kako aplikacija radi na principu vremenski ovisnog rutiranja, najvažniji podaci su oni o prosječnim brzinama na linkovima u intervalima. Profil brzine može se definirati kao očekivana brzina nekog vozila za određeni segment tijekom promatranog vremenskog razdoblja. Kako se radi o ponavljajućem zagušenju, promatrana su dva čimbenika koja utječu na zagušenje: godišnje doba i dani u tjednu. Razdoblje jednog dana je podijeljeno na dva dijela zbog manjeg broja podataka u noćnim satima: dnevno razdoblje (05:30h-22:00h) i noćno razdoblje (22:00h-05:30h). Promatrani su i različiti dani u tjednu zbog veće prometne potražnje za vrijeme radnih dana. Zbog tih svih čimbenika koriste se petominutni intervali kao ravnoteža između kvalitete procjene i broja podataka. Za svaki je interval izračunata prosječna brzina prostora $v(tk)$ pomoću jednadžbe $v(tk) = n/\sum_{i=1}^{n} \frac{1}{vi(tk)}$. Ovdje se koristi prosječna brzina prostora jer predstavlja aritmetičku srednju brzinu svih vozila koja koriste jedan segment i daju veću težinu sporijim vozilima što pomaže kod prikazivanja zagušenja, [5]. Na slici 4 se apsolutni profil brzine i kako se on mijenja kroz petominutne intervale.

Slika 4. Graf apsolutnog profila brzine

Izvor: Izradio autor

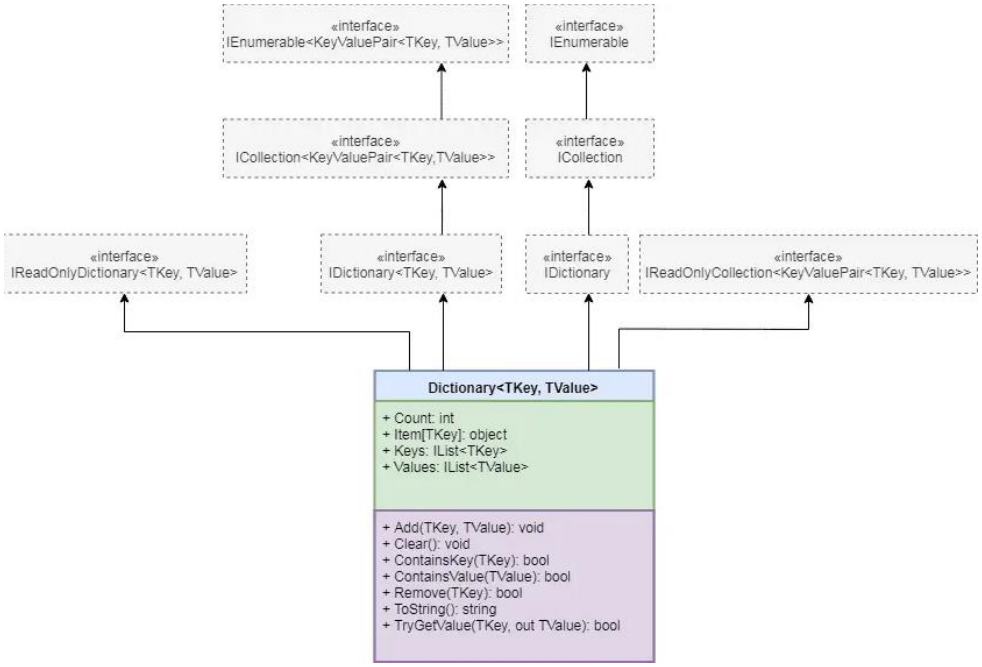
Da bi se moglo pristupiti podacima koji se nalaze u „updatedGraphZg“ tekstualnoj datoteci unutar Windows Forms aplikacije, potrebno je koristiti klasu StreamReader. Klasi StreamReader se pristupa preko imenskog prostora „using System.IO“. Using System.IO označava ulaz i izlaz, to jest omogućuje Visual Studio-u da uzme od korisnika neku datoteku, koristi ju, zatim ispiše neki izlaz nazad korisniku, [6].

Klasa je podatkovna struktura u programskim jezicima koja može kombinirati funkcije i varijable u jednu novu jedinicu. Klasa je nacrt dok objekti sadrže podatke koji se predaju klasi. Objekt ili instanca klase, stvara se prilikom izvođenja programa pa se ujedno naziva i entitetom izvođenja. Nakon pristupa podacima, ti se podaci dijele uz pomoć naredbe „Split()“ te se spremaju u novu klasu koja se zove „Podatci“, [7]. Na slici 5 se prikazuju mogućnosti rada s klasama.



Slika 5. Prikaz klase, [8]

Kad se podatci spreme u klasu „Podatci“, ta se klasa poslije sprema u mapu. Mapa je generička kolekcija koja može spremati ključ-vrijednost parove. Ključ koji se predaje mapi mora biti jedinstven i ne smije biti vrijednosti ništa (engl. Null), dok vrijednosti smiju biti null i duplikat. Vrijednostima se pristupa jako brzo preko ključa, [9]. Na slici 6 se vide mogućnosti strukture podataka mapa.



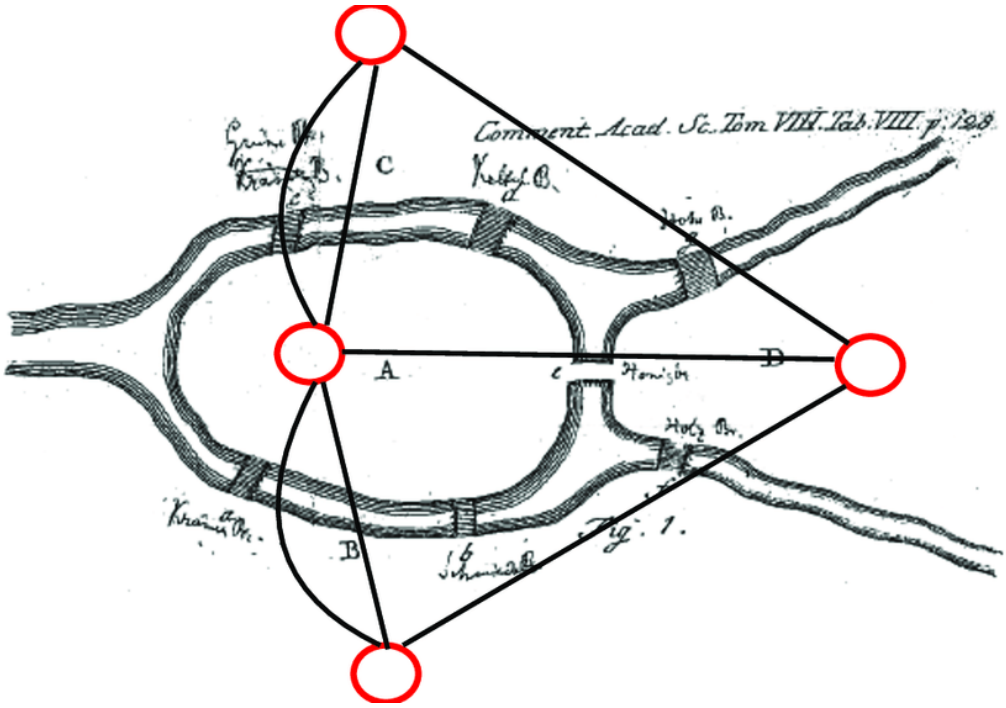
Slika 6. Prikaz mape i njenih mogućnosti, [9]

3. Graf prometne mreže i algoritam za pronalazak najkraćeg puta na grafu

Da bi se mogli koristiti prikupljeni podaci i da bi iz njih mogli računati najbolju rutu u gradu Zagrebu, prvo se ti podaci moraju prebaciti u graf. Tijekom modeliranja cestovne mreže u graf, čvorovi predstavljaju točke na mapi, te su to konceptualno najčešće raskrižja ili čvorovi ceste. Bridovi najčešće predstavljaju cestu koja povezuje ta dva čvora. Zbog načina na koji je spremljena cestovne mreža Zagreba, ali i pogodnosti u uštedi memorije, graf je modeliran na način da jedna cesta (cestovni segment) predstavlja čvor u grafu (početka ceste), a brid predstavlja putovanje s početka tog cestovnog segmenta do početka drugog cestovnog segmenta. Bridovi moraju imati duljinu, brzinsko ograničenje i smjer kretanja. Zatim se iz toga grafa dalje uz pomoć Dijkstrinog algoritma i Fibonaccijeve hrpe računa ruta, [10].

3.1 Teorija grafova

Početak teorije grafova može se povezati s problemom Königsberških mostova. U osamnaestom stoljeću rijeka Preger je odvajala centralni dio Königsberga od ostatka grada. Taj centralni dio se sastojao od dva otoka koja su bila spojena sa ostatkom grada sa sedam mostova. Zbog toga su se ljudi u Königsbergu pitali dali je moguće odabrati početnu točku, prijeći svih sedam mostova točno jednom i vratiti se na početnu točku. 1736. godine Leonhard Euler je uspješno riješio problem Königsberških mostova. Korištenjem „nove“ geometrije (topologije) on sliku mostova pretvara u graf (skup vrhova i bridova). Euler je prikazao kopnene mase kao skup četiri vrha (A, B, C i D) i sedam linija koje su predstavljale mostove koji su spajale te vrhove. Rješavanjem ovog problema Euler je stvorio novu stranu matematike koju nazivamo teorijom grafova. Na slici 7 se može vidjeti problem sedam mostova.



Slika 7. Problem sedam mostova, [11]

Euler je formirao tri glavna zaključka problema Königsberških mostova, [2]:

1. Ako je bilo koje kopno povezano s drugim kopnom neparnim brojem mostova, tada putovanje koje prelazi svakim mostom točno jednom nije moguće.
2. Ako je broj mostova neparan za točno dva kopna, tada je moguće putovanje koje točno prelazi svaki most jednom, ali putovanje mora započeti na jednom kopnu koji ima neparan broj mostova i završiti na drugom.
3. Ako nema kopna povezanog s neparnim brojem mostova, tada putovanje može početi s bilo kojeg kopna i završiti na tom istome kopnu.

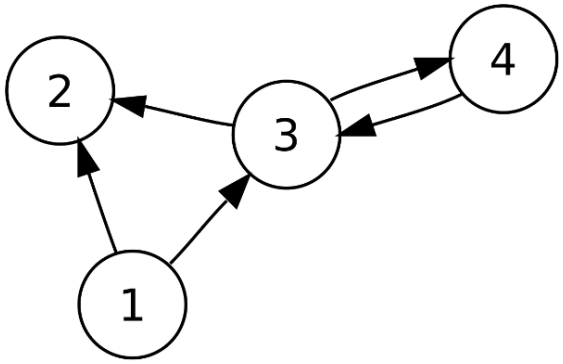
3.2 Grafovi i vrste grafova

Teorija grafova služi za proučavanje grafova. Graf je nelinearna struktura koja se sastoji od vrhova i bridova. Bridovi služe kao poveznice vrhova u grafu. Formalno graf se prikazuje kao uređeni par $G = (V, E)$. V je skup vrhova u nekom grafu dok je E skup bridova u nekom grafu. Grafovi se koriste u različitim granama znanosti: problem usmjeravanja prometa, upravljanja mrežom, optimizacija i planiranje ruta, teorija igara i analiza podataka, [12].

Postoji više vrsta grafova: usmjereni graf, neusmjereni graf, težinski simetrični graf i težinski asimetrični graf.

3.2.1 Usmjereni graf

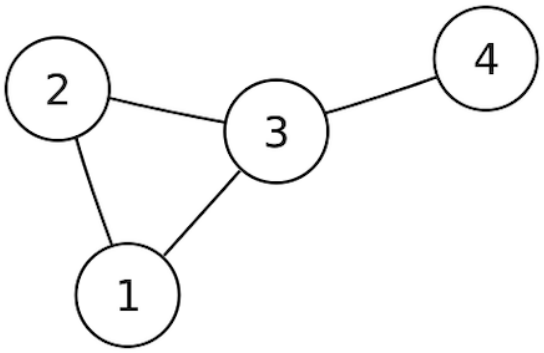
Usmjereni graf je graf u kojem je svaki brid između vrhova ima usmjerenje. Usmjerenje se prikazuje u obliku strelice koja pokazuje u kojem se smjeru može ići. Na slici 8 je prikazan usmjereni graf, [13].



Slika 8. Izgled usmjerenog grafa, [14]

3.2.2 Neusmjereni graf

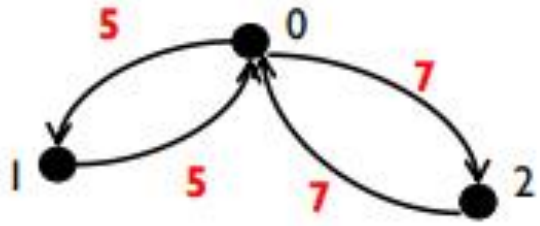
Neusmjereni graf je graf u kojem je svaki vrh povezan s drugim vrhom nekim bridom, taj brid nema usmjerenja te se može ići u oba smjera. Na slici 9 je prikazan neusmjereni graf, [13].



Slika 9. Izgled neusmjerenog grafa, [15]

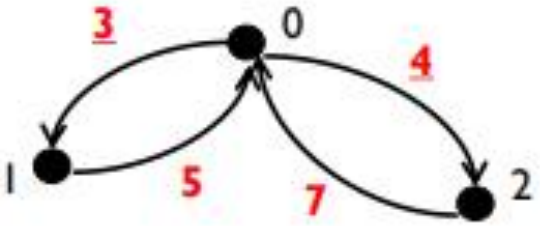
3.2.3 Težinski simetrični graf

Težinski simetrični graf je graf kojem je težina brida između vrhova A i B ista kao i težina brida između vrhova B i A. Na bilo koji smjer da se ide iz jednog vrha u drugi, konačni put bi trebao biti iste dužine. Na slici 10 se može vidjeti težinski simetričan graf, [13].



Slika 10. Izgled težinski simetričnog grafa, [13]

Težinski asimetričan graf je graf u kojem težina brida između vrha A i B nije ista kao težina brida između vrhova B i A, [13]. Na slici 11 je prikazan izgled težinski asimetričnog grafa.



Slika 11. Izgled težinski asimetričnog grafa, [13]

3.3 Dijkstra algoritam

Dijkstrin algoritam je jedan od najčešće korištenih i najpoznatijih algoritama koji se koristi za pronalazak najkraćeg puta u težinskom grafu. Algoritam se može koristiti i u usmjerenim i u neusmjerenim grafovima. Nizozemac Edsger W. Dijkstra je 1956. godine razvio

ovaj algoritam. Dijkstrin algoritam se koristi za planiranje ruta, optimizaciju ruta, GPS navigaciju i mnoge druge stvari, [16].

3.3.1 Izgled algoritma

Srž algoritma je u pronalasku najbližeg susjednog vrha od početnog vrha i svakog sljedećeg vrha. Kada se pronađe najbolji susjedni vrh, algoritam se vraća na početak te opet gleda najbliži susjedni vrh i tako se ponavlja sve dok ne prođe sve vrhove jednom. Na kraju se gleda vrijednost svakog puta i put s najmanjom vrijednošću je najbolji put za dolazak od početnog vrha do krajnjeg vrha. Jedan od najčešćih problema ovog algoritma je taj što s velikim brojem vrhova raste i vrijeme izvršavanja algoritma. Stoga se uz Dijkstrin algoritam često kombiniraju dugi algoritmi koji ubrzavaju rješavanje problema. Na slici 12 se vidi prvi dio Dijkstrinog algoritma, prikazan pseudokodom, [16].

```
Algorithm 1: Djkstra's Algorithm
Input: graph
Input: startNode
Input: targetNode
Output: Path
for node in graph do
    node.score := Inf;
    node.visited := false;
end
startNode.score := 0;
while true do
    currentNode := nodeWithLowestScore(graph);
    currentNode.visited := true;
    for nextNode in currentNode.neighbors do
        if nextNode.visited == false then
            newScore := calculateScore(currentNode, nextNode);
            if newScore < nextNode.score then
                nextNode.score := newScore;
                nextNode.routeToNode := currentNode;
            end
        end
    end
    if currentNode == targetNode then
        return buildPath(targetNode);
    end
    if nodeWithLowestScore(graph).score == Inf then
        throw NoPathFound;
    end
end
```

Slika 12. Pseudokod Dijkstrinog algoritma, [16]

Da bi se pronašao najbolji neposjećeni vrh on mora imati najnižu vrijednost. Na slici 13 se vidi dio algoritma koji računa vrh s najnižom vrijednosti, [16].

```
Algorithm 2: nodeWithLowestScore
Input: graph
Output: node
result := null;
for node in graph do
    if node.visited == false AND node.score < result.score then
        result := node;
    end
end
return result;
```

Slika 13. Pseudokod Dijkstrinog algoritma za pronalazak najboljeg vrha, [16]

Poslije ovoga mora se izračunati novi rezultat za svaki vrh. To je trenutni rezultat za vrh iz kojeg se kreće zbrojeno, s troškom grane (brida) za dolazak do novog vrha iz tog istog vrha, [16].

```
Algorithm 3: calculateScore
  Input: currentNode
  Input: nextNode
  Output: score
  return currentNode.score + currentNode.edgeCost(nextNode);
```

Slika 14. Pseudokod Dijkstrinog algoritam za računanje vrijednosti vrhova, [16]

Kada se stigne do cilja treba se zapisati ruta. Ruta se zapisuje u listu redom kako se pronalazi najbolji prethodni vrh za svaki vrh, [16]. Na slici 15 se vidi proces spremanja vrhova u listu.

```
Algorithm 4: buildPath
  Input: targetNode
  Output: builtPath
  route := new List();
  currentNode := targetNode;
  while currentNode do
  |  route.add(currentNode);
  |  currentNode := currentNode.routeToNode;
  end
  return route;
```

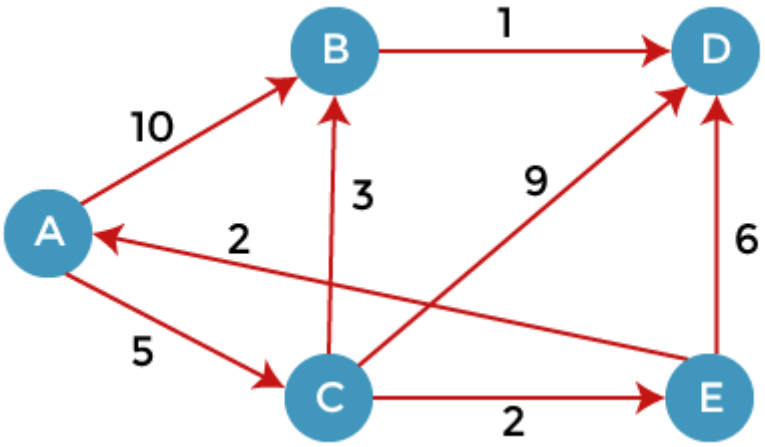
Slika 15. Pseudokod Dijkstrinog algoritam za spremanje vrhova u listu, [16]

3.3.2 Primjer rješavanja Dijkstrinog algoritma

Program koji je napravljen za potrebe ovog rada, radi na principu vremenskog rutiranja, zato će se u ovom poglavlju riješiti isti usmjereni graf sa različitim vrijednostima. Prvi primjer će predstavljati vrijeme putovanja u vremenskom periodu od 00:00h do 08:00h, drugi primjer će pokazivati vrijeme putovanja u vremenskom periodu od 08:00h do 16:00h, treći primjer će pokazivati vrijeme putovanja u vremenskom periodu od 16:00h do 24:00h. Jedino što će se mijenjati u primjerima su težine bridova, a s time i rezultat rutiranja.

3.3.2.1 Primjer Dijkstrinog algoritma za vremenski period od 00:00h do 08:00h

U sljedećem primjeru je riješen usmjereni graf korištenjem Dijkstrinog algoritma za vremenski period od 00:00h do 08:00h. Opisani su detaljno svi koraci rješavanja ovog grafa. Slika 16 prikazuje kako graf izgleda. Graf se sastoji od pet vrhova ($A, B, C, D\ i\ E$) i osam usmjerenih bridova, svaki brid predstavlja vrijeme u minutama potrebno da se dođe iz jednog vrha u drugi. U ovom primjeru se pronalazi najkraći put od vrha A do vrha D, [17].



Slika 16. Usmjereni graf za period od 00:00h do 08:00h, [17]

Prvi korak u rješavanju Dijkstrinog algoritma je postavljanje početnog vrha (A u ovom primjeru) na nulu, a ostale vrhove postavljamo na beskonačno. Kako je početni vrh nula, on ima najkraću vrijednost brida pa se on prvi piše s lijeve strane, prvi stupac s lijeve strane služi za to da se zna u kojem je vrhu u određenom koraku, [17].

Tablica 2. Prvi korak Dijkstrinog algoritma

	A		B	C	D	E
A	0		∞	∞	∞	∞

U drugom koraku se gleda do kojih vrhova se može doći iz početnog vrha A, te se njihove vrijednosti bridova zapisuju u tablicu. U ovome primjeru se može ići u vrh B i vrh C. Bitno je napomenuti ako je vrijednost u tablici manja nego u prijašnjem koraku tada se nove vrijednosti upisuju, inače se prepisuju, [17].

Tablica 3. Drugi korak Dijkstrinog algoritma

	A	B	C	D	E
A	0	∞	∞	∞	∞
C	0	10	5	∞	∞

Pošto je do C vrha manja udaljenost (5<10) od njega se nastavlja Dijkstra algoritam. Dalje se iz vrha C može ići do vrha B, D i E. U sljedećoj tablici se zbrajaju vrijednosti od početnog vrha pa do zadnjeg vrha, to jest ako se ide od vrha A, preko vrha C do vrha B, ukupna vrijednost puta će biti osam, [17].

Tablica 4. Treći korak Dijkstrinog algoritma

	A	B	C	D	E
A	0	∞	∞	∞	∞
C	0	10	5	∞	∞
E	0	8	5	14	7

Od svih vrhova najkraći put je do vrha E, zato gledamo njega sljedećeg. Iz vrha E se može ići do vrha A i vrha D. Pošto je vrh A početni vrh do njega se ne ide, to jest gledaju se samo vrhovi koji još nisu posjećeni, [17].

Tablica 5. Četvrti korak Dijkstrinog algoritma

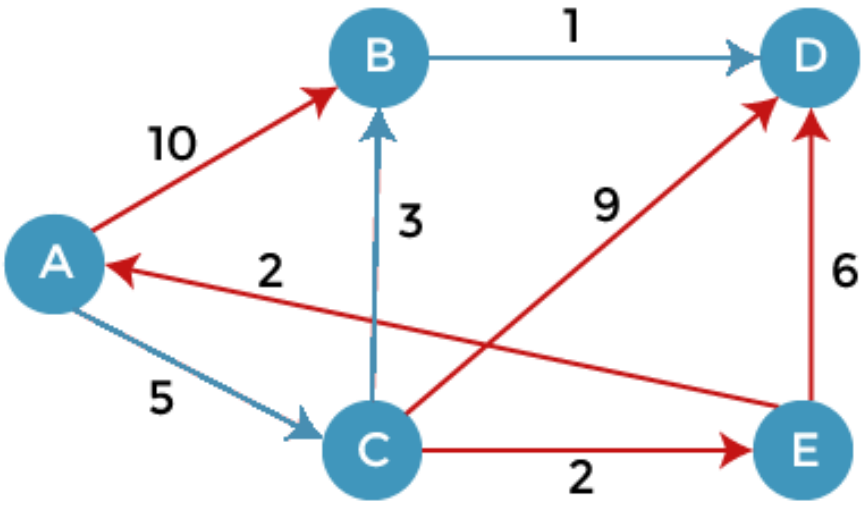
	A	B	C	D	E
A	0	∞	∞	∞	∞
C	0	10	5	∞	∞
E	0	8	5	14	7
B	0	8	5	13	7

U petom i zadnjem koraku algoritam se vraća u vrh B jer je najkraća udaljenost do vrha B, iz vrha B se još može ići do vrha D, [17].

Tablica 6. Peti korak Dijkstrinog algoritma

	A	B	C	D	E
A	0	∞	∞	∞	∞
C	0	10	5	∞	∞
E	0	8	5	14	7
B	0	8	5	13	7
D	0	8	5	9	7

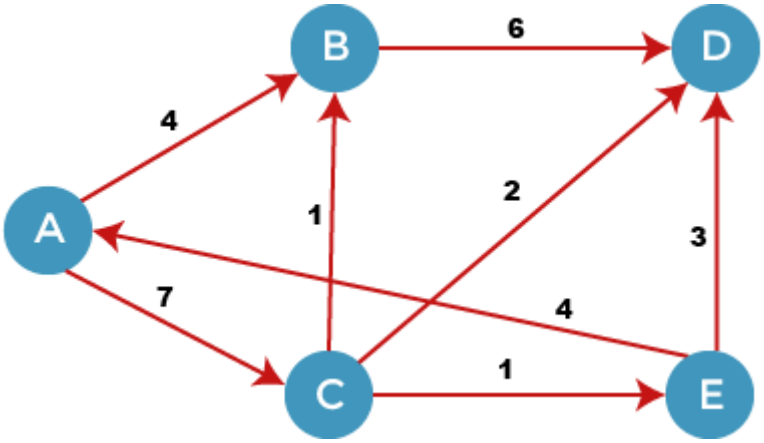
Kada se prođu svi vrhovi Dijkstra algoritam je gotov. Najkraći put od vrha A do vrha D je preko vrhova A, C, B, D i njegova ukupna vrijednost je devet, [17].



Slika 17. Put u usmjerenom grafu od vrha A do vrha D
Izvor: Izradio autor

3.3.2.2 Primjer Dijkstrinog algoritma za vremenski period od 08:00h do 16:00h

U sljedećem primjeru je riješen usmjereni graf korištenjem Dijkstrinog algoritma za vremenski period od 08:00h do 16:00h. Slika 18 prikazuje kako graf izgleda poslije promjene vrijednosti bridova. U ovom primjeru se pronalazi najkraći put od vrha A do vrha D.



Slika 18. Usmjereni graf za period od 08:00h do 16:00h

Izvor. Izradio autor

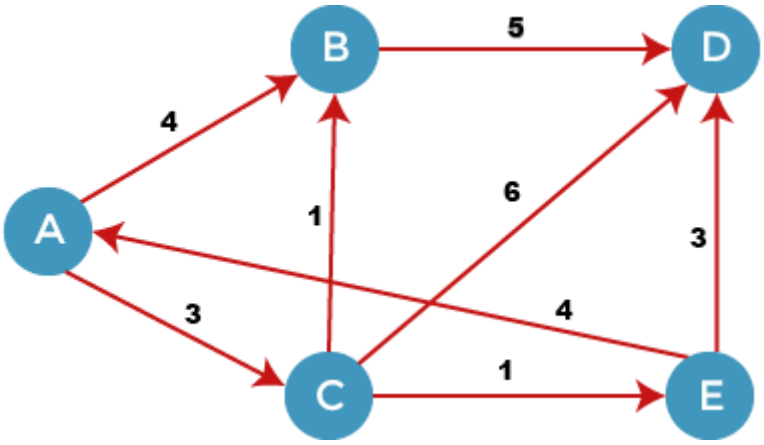
Tablica 7. Izgled tablice i rute za period od 08:00h do 16:00h

	A	B	C	D	E
	0	∞	∞	∞	∞
A	0	4	7	∞	∞
B	0	4	7	10	∞
C	0	4	7	9	8
E	0	4	7	9	8
D	0	4	7	9	8

U ovom primjeru putovanja, u vremenskom periodu od 08:00h do 16:00h najkraći put od vrha A do vrha D je preko vrhova A, C, D. Ukupna vrijednost puta je devet kao i prošlom primjeru.

3.3.2.3 Primjer Dijkstrinog algoritma za vremenski period od 16:00h do 00:00h

U sljedećem primjeru je riješen usmjereni graf korištenjem Dijkstrinog algoritma za vremenski period od 16:00h do 00:00h. Slika 19 prikazuje kako graf izgleda. U ovom primjeru se pronalazi najkraći put od vrha A do vrha D.



Slika 19. Usmjereni graf za period od 16:00h do 00:00h

Izvor: Izradio autor

Tablica 8. Izgled tablice i rute za period od 16:00h do 00:00h

	A	B	C	D	E
	0	∞	∞	∞	∞
A	0	4	3	∞	∞
C	0	4	3	9	4
B	0	4	3	9	4
E	0	4	3	7	4
D	0	4	3	7	4

U ovom primjeru putovanja, u vremenskom periodu od 16:00h do 00:00h najkraći put od vrha A do vrha D je preko vrhova A, C, E, D. Ukupna vrijednost puta je sedam.

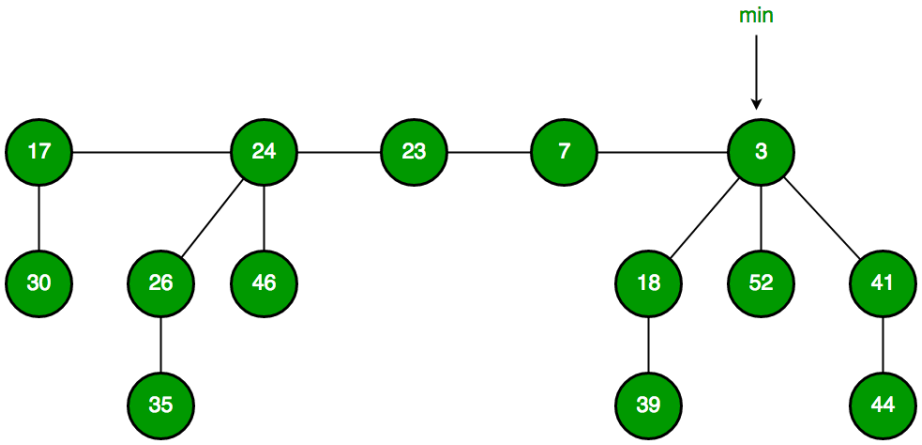
3.4 Fibonaccijeva hrpa

Fibonaccijevu hrpu su razvili Michael L. Fredman i Robert E. Trajan 1984. godine, [18]. Ova hrpa je dobila naziv po Fibonaccijevim brojevima, Fibonaccijevi brojevi se koriste u analizi vremena izvođenja. Fibonaccijeva hrpa je struktura podataka, koristi se za implementaciju prioritetnih redova. Hipovi ili hrpe na hrvatskom jeziku su tipovi podataka koje se koriste za prikazivanje odnosa između djece i roditelja. Imamo dvije vrste hipova, min-hipovi i max-hipovi.

- min-hip je vrsta stabla u kojem su za sve čvorove vrijednost ključa roditelja manja od vrijednosti ključa djece,
- max-hip je vrsta stabla u kojem je za sve čvorove vrijednost ključa roditelja veća od vrijednosti ključa djece.

U Fibonaccijevoj hrpi čvor može imati više od dvoje djece, a može uopće ne imati djece. Glavna prednost Fibonaccijeve hrpe u odnosu na binarnu i binomnu hrpu je brzo amortiziranje vremena izvođenja operacija poput spajanja, izdvajanja i umetanja. Vrijeme izvođenja spajanja je $O(1)$, za izdvajanje $O(log\ n)$ a za umetanje $O(1)$. U Fibonaccijevoj hrpi su stabla složena tako da korijenski čvor bude s minimalnom vrijednosti ključa. Ako se ubacuje novi element u hrpu on se postavlja kao novo stablo s jednim čvorom, ako se spajaju dvije hrpe,

liste korijena jedne hrpe se prebace na listu korijena druge liste. Lijeno spajanje ili odgađanje spajanja stabla dok to nije nužno čini Fibonaccijevu hrpu superiornijom u usporedbi s binarnom i binomnom hrpom, [19]. Izgled Fibonaccijeve hrpe je prikazan na slici 20.



Slika 20. Izgled Fibonaccijeve hrpe, [19]

Za razumijevanje Fibonaccijeve hrpe moraju se poznavati neke osnovne operacije prilikom izvršavanja algoritma:

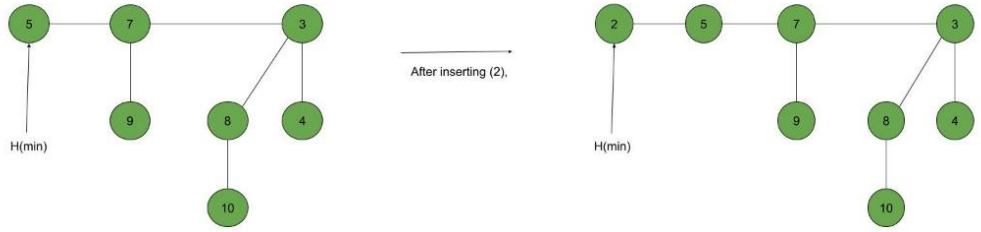
- umetanje elementa u hrpu,
- unija između dvije hrpe,
- izvlačenje minimalne vrijednosti elementa u hrpi,
- smanjivanje vrijednosti elementa u hrpi,
- brisanje elementa u hrpi.

3.4.1 Operacija umetanja

Ako se dodaje novi čvor u već postojeću hrpu, prvo se stvori novi čvor koji predstavlja novi element koji se dodaje toj hrpi. Poslije ubacivanja elementa u hrpu provjerava se dali je hrpa prazna. Ako se utvrdi da je hrpa prazna tada taj novi element postaje korijenski čvor. Pošto nema drugih elemenata on se označava kao minimum. Ako se u hrpi nalazi nalaze elementi taj novi element se samo umetne u listu korijena. U sljedećih par redaka je opisan algoritam za umetanje novog elementa u hrpu, [20]:

1. *Stvori novi čvor „x“,*
2. *Provjeri dali je hrpa H prazna ili ne*
3. *Ako je H prazan onda:*
    - *Postavi „x“ kao jedini čvor u korijenskoj listi,*
    - *Postavi pokazivač H(min) na x,*
4. *Inače:*
    - *Umetni x u korijensku listu i ažuriraj H(min).*

Na slici 21 se vidi izgled hrpe poslije umetanja novog elementa.



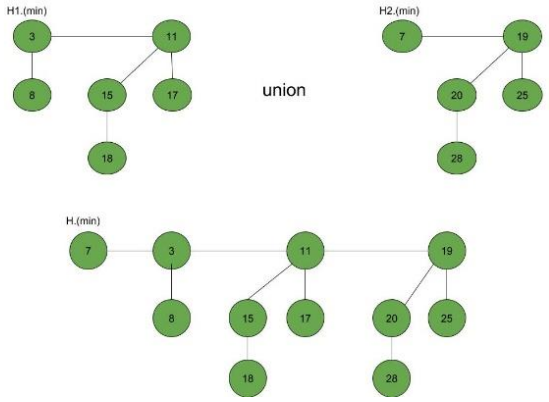
Slika 21. Prikaz umetanja novog elementa u hrpu, [20]

3.4.2 Operacija unije

Prilikom spajanja dvije Fibonaccijeve hrpe, obje se hrpe spoje tako da se jedna lista korijena hrpe poveže s listovima korijena druge hrpe. Kada se dobije nova lista korijena potrebno je ažurirati minimum hrpe tako da hrpa ne gubi svojstva. U sljedećih par redaka je opisan algoritam za spajanje dvije hrpe, [20]:

1. *Spoji korijenske liste Fibonaccijevih hrpa H1 i H2 i stvori jedinstvenu Fibonaccijevu hrpu H,*
2. *Ako je H1(min) < H2(min) tada:*
    - *H(min) = H1(min),*
3. *Inače:*
    - *H(min) = H2(min).*

Na slici 22 se vidi izgled nove hrpe poslije spajanja.



Slika 22. Izgled hrpe poslije spajanja dvije hrpe, [20]

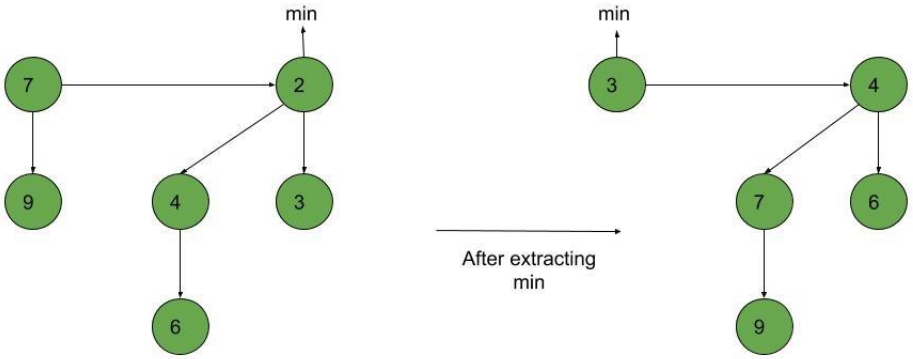
3.4.3 Operacija izvlačenja minimalne vrijednosti

Da bi se mogla obaviti operacija izvlačenja minimalne vrijednosti iz hrpe prvo se mora obrisati sadašnja minimalna vrijednost. Poslije brisanja minimalne vrijednosti postavlja se nova minimalna vrijednost na sljedeći korijen s najmanjom vrijednošću, te se sljedećem minimumu

predaju svi elementi stabla od prijašnjeg minimuma. Zatim se stvara polje jednake veličine maksimalnoj dubini stabla u hrpi prije brisanja minimuma. Postavlja se pokazivač stupnjeva na trenutni čvor. Prolazi se ostalim čvorovima i gledaju se njihovi trenutni pokazivači stupnjeva. Ako je različit stupanj dubine stabla tada se prelazi na sljedeći čvor, ako je isti kao prijašnjem tada se elementi stabla spajaju preko operacija unije. Ponavlja se zadnji i predzadnji korak sve dok ne budu dva ista stupnja dubine stabla u hrpi. U sljedećih par redaka je opisan algoritam za izvlačenje minimalne vrijednosti, [21]:

1. *Obriši minimalni čvor,*
2. *Postavi glavu na sljedeći minimalni čvor i dodaj sva stabla izbrisanog čvora u korijensku listu,*
3. *Stvori polje pokazivača stupnja veličine izbrisanog čvora,*
4. *Postavi pokazivač stupnja na trenutni čvor,*
5. *Pomakni se na sljedeći čvor,*
    - *Ako su stupnjevi različiti, onda postavi pokazivač stupnja na sljedeći čvor,*
    - *Ako su stupnjevi isti, onda spoji Fibonaccijeva stabla preko operacije unije,*
6. *Ponavljaj korake 4 i 5 dok se hrpa ne dovrši.*

Na slici 23 se vidi izgled hrpe poslije izvlačenja minimalne vrijednosti čvora.



Slika 23. Izgled hrpe poslije izvlačenja najmanje vrijednost, [21]

3.4.4 Operacija smanjivanja ključa

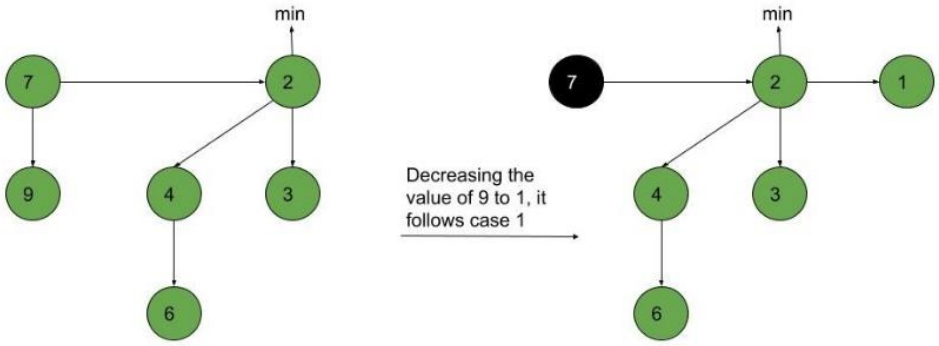
Za smanjivanje vrijednosti bilo kojeg elementa u hrpi postoje tri moguća algoritma. Slučaj jedan, kada nije narušeno svojstvo min-hipa, hrpa se samo ažurira na novu vrijednost koja je zadana. Slučaj dva, kada je narušeno svojstvo min-hipa i kada roditelj tog čvora nije označen. Tad se prekida veza između tog stabla i roditelja, stvaramo novog roditelja od tog stabla te se ono ubacuje na pravo mjesto u hrpi. Slučaj tri, kada je svojstvo min-hipa narušeno i kada je roditelj tog čvora označen. Tad se prekida veza između stabla i roditelja, dodaje se stablo na listu korjenova i ažurira se minimalna vrijednost. U sljedećih par redaka je opisan algoritam za smanjivanje vrijednosti ključa u hrpi, [21]:

1. *Smanji vrijednost čvora „x“ na novu odabranu vrijednost,*
2. *Slučaj 1: Ako svojstvo min-hrpe nije narušeno,*

- *Ažuriraj pokazivač minimalne vrijednosti ako je potrebno,*

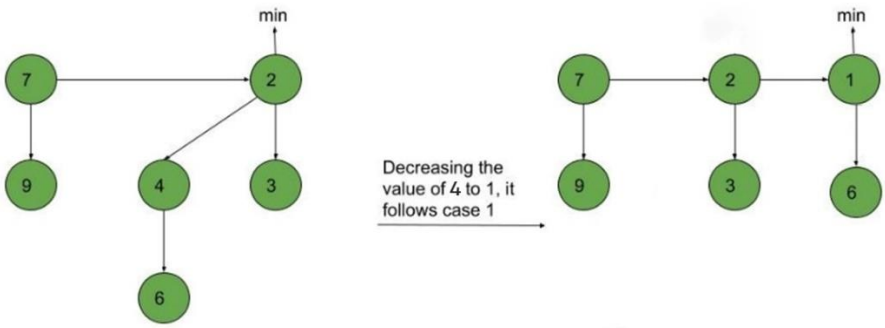
3. *Slučaj 2: Ako je svojstvo min-hrpe narušeno i roditelj „x“ nije označen,*
    - *Prekini vezu između „x“ i njegovog roditelja,*
    - *Označi roditelja od „x“,*
    - *Dodaj stablo s korijenom „x“ u korijensku listu, ako je potrebno ažuriraj pokazivač minimalne vrijednosti*
4. *Slučaj 3: Ako je svojstvo min-hrpe narušeno i roditelj „x“ je označen,*
    - *Prekini vezu između „x“ i njegovog roditelja p[x],*
    - *Dodaj „x“ u korijensku listu, ažuriraj pokazivač minimalne vrijednosti ako je potrebno,*
    - *Prekini vezu između p[x] i p[p[x]],*
    - *Dodaj p[x] u korijensku listu, ažuriraj pokazivač minimalne vrijednosti ako je potrebno,*
    - *Ako je p[p[x]] ne označen, označi ga,*
    - *Inače prekini p[p[x]] i ponovi korake 4.2 do 4.5, uzimajući p[p[x]] kao „x“.*

Na slici 24 se može vidjeti izgled hrpe poslije smanjivanja ključa.



Slika 24. Izgled hrpe poslije smanjivanja vrijednosti ključa 9, [21]

Ako se želi smanjiti vrijednost ključa 4 na vrijednost 1 u Fibonaccijevoj hrpi sa slike 25, koristio bi se treći slučaj operacije smanjivanja ključa. Vrijednost ključa 4 bi se pretvorila u 1 i prekinula bi se veza između ključa i njegovog roditelja (2), nova vrijednost ključa 1 bi se dodala u korijensku listu sa ključem 6. Zatim bi se ažurirala minimalna vrijednost hrpe jer je 1<2 (stare minimalne vrijednosti). Izgled hrpe je prikazan na slici 25.



Slika 25. Izgled hrpe poslije smanjivanja vrijednosti ključa 4

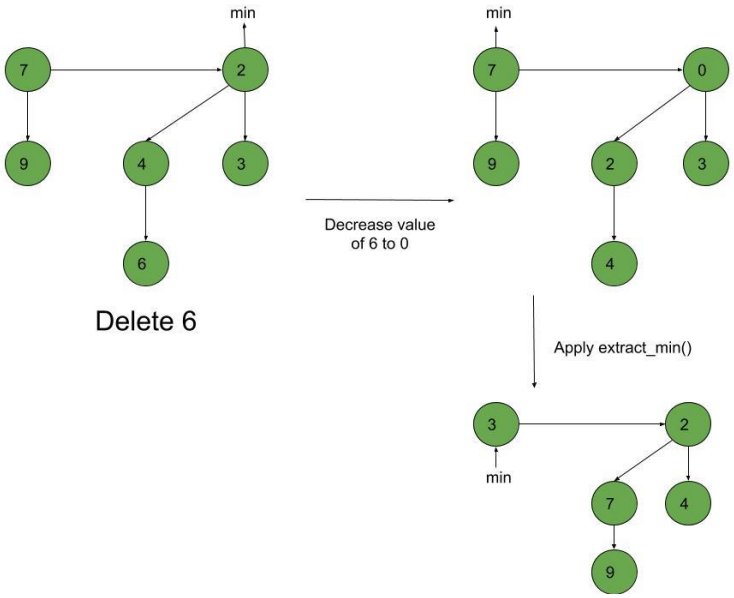
Izvor: Izradio autor

3.4.5 Operacija brisanja

Da bi se obrisao element u Fibonaccijevoj hrpi prvo se treba njegova vrijednost smanjiti na najmanju moguću vrijednost. Nakon smanjivanja na najmanju moguću vrijednost taj se čvor prebacuje na listu korijena, poslije prebacivanja u listu čvorova primjenjuje se operacija uzimanja minimalne vrijednosti (Extract_min()) kako bi se čvor uklonio iz hrpe. U sljedećih par redaka je opisan algoritam za brisanje vrijednosti u hrpi, [21]:

1. *Smanji vrijednost čvora koji će biti izbrisan „x" na minimum pomoću funkcije Decrease_key(),*
2. *Koristeći svojstvo min-hrpe, preuredi hrpu koja sadrži „x", ubaci „x" u korijensku listu,*
3. *Primjeni algoritam Extract_min() na Fibonaccijevoj hrpi.*

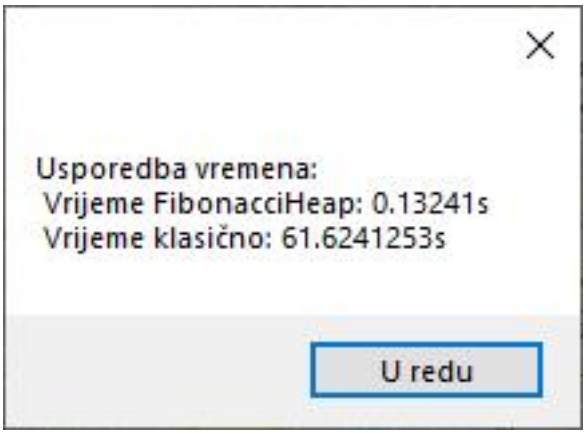
Na slici 26 se može vidjeti izgled hrpe poslije brisanja vrijednosti.



Slika 26. Izgled hrpe poslije brisanja vrijednosti u hrpi, [21]

3.5 Korištenje Fibonaccijeve hrpe u Dijkstrinom algoritmu

Originalni Dijkstrin algoritam koji se koristi za pronalaženje najkraćeg puta ne koristi prioritetni red i zbog toga radi u vremenskoj složenosti $O(V2)$. Ako se ubaci Fibonaccijeva hrpa kao prioritetni red tad algoritam radi u vremenskoj složenosti $O(E + V * logV)$. No ako se radi o grafu slabe gustoće u kojem je $E \approx V$, kakva je razmatran u ovom radu jer jedan vrh u prosjeku ima 1.89775 izlaznih bridova, dolazi se do složenosti $O(Vlog\ V)$. To je trenutno najbrža poznata vremenska složenost za Dijkstrin algoritam. Fibonaccijeve hrpe se najbolje koriste u situacijama u kojim je broj brisanja ograničen i mali u usporedbi s drugim operacijama, [22]. Za usporedbu izvršavanja navedenih implementacija Dijkstrinog algoritma, u programu su dodana dva tajmera koji mjere vrijeme izvršavanja algoritma. Za razumijevanje rezultata bitno je napomenuti veličinu razmatranog grafa, koji ima 87984 vrhova i 43992 bridova. Također, algoritmi su izvršeni na laptopu s procesorom od 2.4 GHz, 8GB rama i 64-bitnim operacijskim sustavom. Na slici 27 prikazano je vrijeme izvršavanja algoritama.



Slika 27. Usporedba korištenja Fibonaccijeve hrpe i klasičnog Dijkstirnog algoritma
Izvor: Izradio autor

3.5.1 Fibonaccijeva hrpa i Dijkstra algoritam u programu

1. *Definiraj javnu statičku metodu „Fibonacci“ s parametrima: „polaziste“, „odrediste“, „vrijemeStarta“, „sviVrhovi“, „_restartDijkstre“*
2. *Inicijaliziraj praznu mapu „sviUHeapu“,*
3. *Inicijaliziraj FibonacciHeap s maksimalnom vrijednošću i spremi ga u varijablu „heap“,*
4. *Inicijaliziraj varijablu „pocetakHeapNode“ s vrijednošću ništa (engl null),*
5. *Postavi „vrijemeDolaska“ polazišta na vrijednost parametra „vrijemeStarta“,*
6. *Za svaki vrh u mapi „sviVrhovi“, ponovi sljedeće korake:*
    - *Postavi „vrijemeDolaska“ vrha na beskonačno,*
    - *Postavi „predhodni“ vrha na null,*
    - *Postavi „vrijemePutovanja“ vrha na 0,*
    - *Kreiraj novi FibonacciHeapNode s podacima vrha i vrijednošću „vrijeDolaska“ vrha, te ga spremi u varijablu node,*
    - *Ako je vReset jednako polazištu, ponovi sljedeće korake:*
        - *Postavi „vrijemeDolaska“ vrha na vrijednost parametara „vrijemeStarta“,*
        - *Postavi „pocetakHeapNode“ na vrijednost „node“*
    - *Inače, ubaci „node“ u heap*
    - *Dodaj „Node“ u mapu „sviUHeapu“ s ključem vrha.linkID,*
7. *Inicijaliziraj brojač $i = 0$,*
8. *Postavi „vrijemeDolaska“ polazišta na vrijednost parametara „vrijemeStarta“,*
9. *Sve dok heap nije prazan ponovi sljedeće korake:*
    - *Ako je brojač $i = 0$, ponovi sljedeće korake*
        - *Postavi „najbolji“ na vrijednost „pocetakHeapNode“,*
        - *Inkrementiraj brojač i,*
    - *Inače ponovi sljedeće korake:*
        - *Ukloni najmanji element iz heapa i spremi ga u varijablu „najbolji“,*
    - *Postavi vrijednost obilježja „obraden“ na vrh podataka „najbolji“,*
    - *Ako heap nije prazan, ponovi sljedeće korake,*

- *Postavi „vrhOd“ na pdatke vrha iz „najbolji“,*
- *Za svaki linkID u „susjednimLinkovima“ „vrhOd“ ponovi sljedeće korake*
  - *Ako mapa „sviVrhovi“ sadrži ključ „linkID“, ponovi sljedeće korake:*
    - *Postavi „vrhDo“ na podatke „sviVrhovi[linkID]“*
    - *Ako je „vrhDo.obraden“ postavljen na true, nastavi,*
    - *Postavi „vrijemePolaska“ na vrijednost „vrhOd.VrijemeDolaska“,*
    - *Postavi „indeksVremena“ na cijeli broj dobiven djeljenjem „vremenaPolaska“ s 5,*
    - *Ako je „indeksVremena“ u intervalu od 0 do 288, ponovi sljedeče korake*
      - *Postavi brzinu na vrijednost „vrhOd.profil[indeksVremena]“,*
      - *Izračunaj „vrijemePutovanja“ kao (vrhOd.duljinaMetri / 1000.0) / brzinom,*
      - *Ako je „vrhDo.VrijemeDolaska“ veće od „vrhOd.VrijemeDolaska“ + „vrijemePutovanja“, ponovi sljedeće korake,*
      - *Postavi „nodeToUpdate“ na vrijednost „sviUHeapu[vrhDo.linkID]“,*
      - *Postavi „vrhDo.VrijemeDolaska“ na „vrhOd.VrijemeDolaska“ + „vrijemePutovanja“,*
      - *Postavi „vrhDo.Predhodni“ na „vrhOd“,*
      - *Smanji ključ „nodeToUpdate“ na vrijednost „vrhDo.VrijemeDolaska“,*
      - *Postavi „vrhDo.UkupnaDuljina na „vrhOd.ukupnaDuljina“ + „vrhDo.DuljinaMetri“,*
      - *Postavi „vrhDo.vrijemePutovanja na „vrhOd.vrijemePutovanja“ + „vrijemePutovanja“*

10. Inicijaliziraj praznu listu „put“,

11. Postavi „trenutni“ na vrijednost odredišta,

12. Sve dok „trenutni“ nije null, ponovi sljedeće korake:

- *Ubaci „trenutni“ na početak liste „put“*
- *Postavi „trenutni“ na vrijednost „trenutni.Predhodni“*

14. *Vrati listu „put“.*

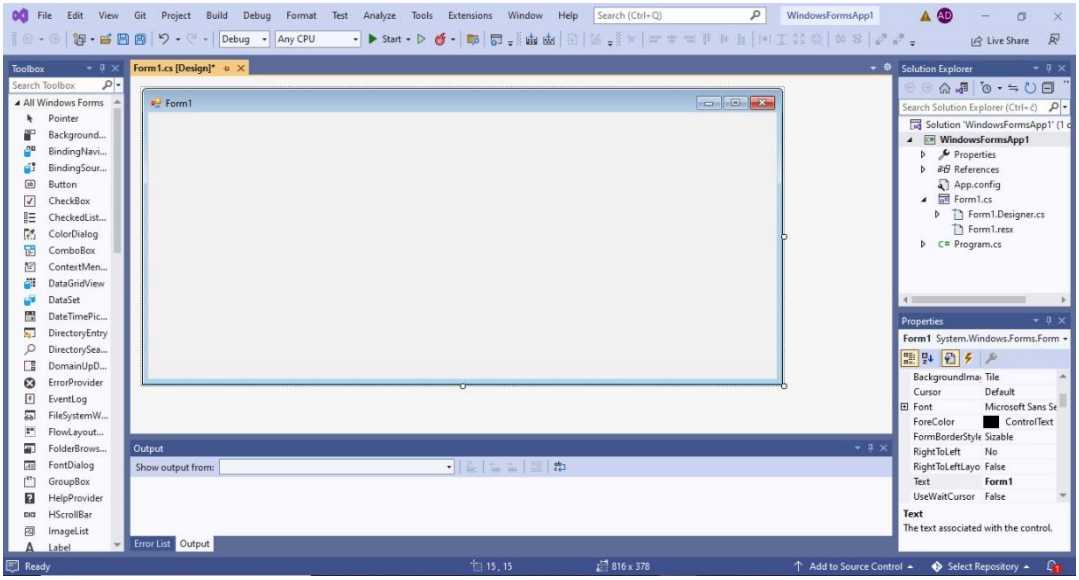
Da se pronađe najbrži put, klasi „Dijkstra“ predaju se podaci polazište (ID početnog linka), odredište (ID zadnjeg linka), vrijemeStarta (vrijeme početka putovanja) i mapa sa svim vrijednostima linkova. Onda se svim ostalim vrhovima u mapi postavlja vrijednost u beskonačno. Ulazi se u petlji koja se kreće izvršavati sve dok Fibonaccijeva hrpa nije prazna. U varijablu „najbolji“ se sprema trenutno najmanja vrijednost hrpe, ako je početak vrtnje i brojač $i = 0$, onda se postavlja varijabla „najbolji“ na „pocetakHeapNode“ koji je zapravo početni vrh. „najbolji“ se dobiva uklanjanjem vrha s najmanjom vrijednosti iz hrpe pomoću metode RemoveMin(). Zatim se postavlja oznaka „obraden“ na true za objekt trenutnog vrha, što znači da je taj vrh obrađen. Ako hrpa nije prazna nastavlja se s obradom susjednih vrhova. U

varijablu „vrhOd“ se sprema objekt trenutnog vrha „najbolji.Data“. Poslije prolaska kroz taj vrh program prolazi kroz sve susjedne linkove vrha „vrhOd“, program provjerava dali vrh s odgovarajućim „linkID“ prisutan u „sviVrhovi“ mapi. Ako je vrh prisutan u mapi, dohvaća se objekt „vrhDo“ iz mape „sviVrhovi“. Provjerava se dali je „vrhDo“ već obrađen, to jest dali je vrh već označen sa „obraden“. Računa se vrijeme polaska („vrijemePolaska“) kao vrijeme dolaska vrha „vrhOd“, izračunava se indeks vremena na temelju „vrijemePolaska“ koje se upisuje u grafičko sučelje. To se vrijeme dijeli na petominutne intervale te se dohvaća brzina iz „profila“ liste vrha „vrhOd“. Zatim se izračunava vrijeme putovanja („vrijemePutovanja“) na temelju duljine linka („vrhOd.duljinaMetri“) i brzine. Ako je vrijeme dolaska vrha „vrhDo“ veće od zbroja vremena dolaska vrha „vrhOd“ i vremena putovanja, ažurira se vrh „vrhDo“. Također se ažuriraju vrijeme dolaska („VrijemeDolaska“), predhodni vrh („Predhodni“) i vrijednost vrha u hrpi („nodeToUpdate“) s novim vrijednostima. Ažuriraju se i dodatni podaci vrha za ispis („vrhDo.ukupnaDuljina“ i „vrhDo.vrijemePutovanja“). Tako se petlja ponavlja za svaki sljedeći susjedni vrh sve dok se ne dobije popis linkova koji čine rutu. Petlja u kojoj se nalazi ovaj dio programa se izvršava sve dok hrpa ne bude prazna.

4. Grafičko korisničko sučelje za prikaz rute i podataka

Grafičko korisničko sučelje (engl. Graphical User Interface, GUI) je digitalno sučelje u kojem se korisnik služi gumbima, ikonama i drugim komponentama. Vizualni prikaz u grafičkom korisničkom sučelju pruža relevantne informacije potrebne korisnicima tog grafičkog sučelja. Prije nastanka grafičkog korisničkog sučelja, korisnici su morali upisivati komande računalima kako bi im računala vratila potrebne informacije. Prvo grafičko korisničko sučelje je napravila tvrtka Xerox 1981. godine. Grafičko korisničko sučelje je nezamjenjivo u današnje vrijeme, od računala, mobitela, televizora do hladnjaka sve ima neki oblik korisničkog sučelja, [23].

Za potrebe ovog završnog rada korišten je Windows Forms Application.NET Framework u razvojnom okruženju Visual Studio 2022. Windows Forms je okvir koji služi za izradu Windows desktop aplikacije. Windows Forms radi na principu drag-and-drop, što ga čini jednim od najlakših alata za izradu aplikacija, [24]. Na slici 28 je prikazan izgled Windows Forms Application sučelja.



Slika 28. Izgled Windows Forms Application sučelja

Izvor: Izradio autor

Za izgled grafičkog korisničkog sučelja ove aplikacije koriste se kontrole poput label, button, textBox, groupBox i NuGet paketi GMap.NET.Core i FibonacciHeap. NuGet sustav u Visual Studiu omogućuje jednostavno dodavanje, uklanjanje i ažuriranje vanjskih biblioteka. GMap.NET.Core omogućava korištenje Open Street Map (OSM) karte, markera, i grafičkih alata za iscrtavanje rute. FibonacciHeap omogućuje korištenje unaprijed napravljenih objekata koji se koriste u Dijkstrinom algoritmu. Na slici 29 može se vidjeti izgled grafičkog korisničkog sučelja aplikacije, [25].

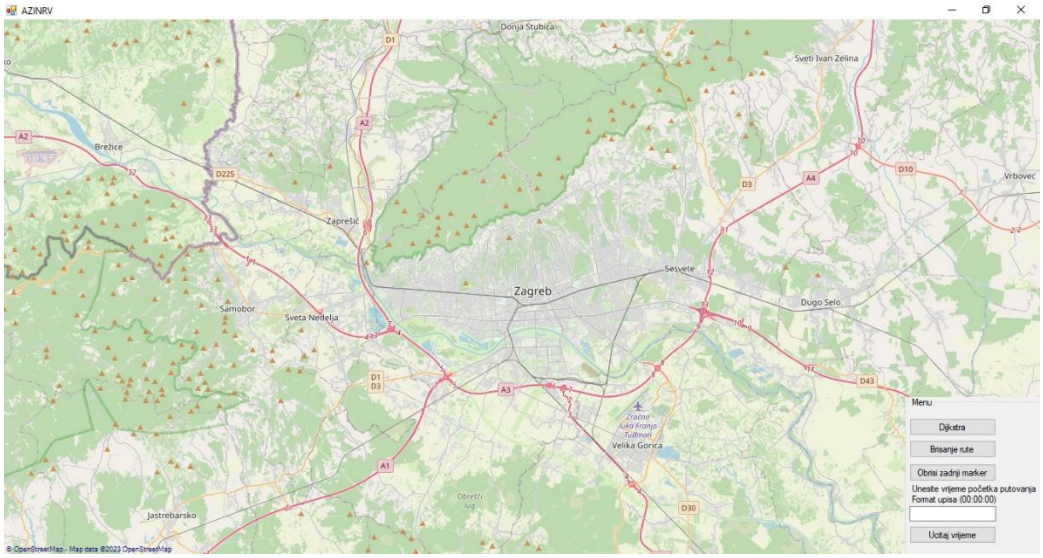
Kontrola Label u Windows Forms aplikaciji se koristi za prikazivanje teksta u sučelju, ona ne sudjeluje u korisničkim unosima ili radnjama miša i tipkovnice. Label je klasa i definirana je u imenskom prostoru System.Windows.Forms, [26].

Button je kontrola koja je interaktivna komponenta koja omogućava korisnicima da komuniciraju s aplikacijom. Button se može stisnuti pomoću tipkovnice ili miša, klikom na njega pokreće se dio koda koji je napisan u C# programskom jeziku, [27].

Kontrola textBox se koristi za upisivanje i prikazivanje stringova ili redova teksta. Ona je interaktivna komponenta koja pomaže korisnicima da komuniciraju s aplikacijom. textBox dozvoljava samo jedan format za prikazani ili uneseni tekst, [28].

GroupBox je kontrola koja služi za smještanje više kontrola u jednu grupu. Svrha GroupBox-a je definiranje korisničkog interfacea gdje možemo kategorizirati povezane kontrole u grupu, [29].

NuGet paketi su dodatni sistemi koji se mogu skinuti i koristiti u Visual Studio-u. Oni omogućuju lagano ubacivanje dodatnih klasa ili kontrola u aplikaciju. U ovom programu su korištena dva dodatna paketa. GMap.NET je besplatni NuGet paket koji omogućuje korištenje rutiranja, geokodiranja i mapa s Yahoo-a, Google-a ili OpenStreet-a u okružiju Visual Studio-a, [30]. FibonacciHeap NuGet paket služi za korištenje već nekih unaprijed sastavljenih funkcija koje se mogu koristi kao dio Fibonaccijevog algoritma u Visual Studio-u.



Slika 29. Izgled grafičkog korisničkog sučelja aplikacije

Izvor: Izradio autor

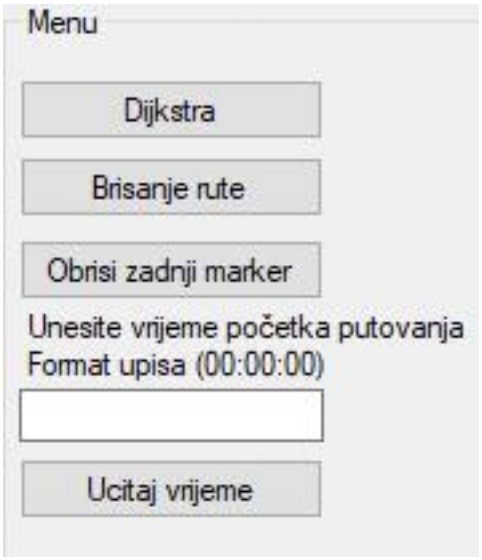
Kada se program pokrene, prvo se pojavi MessageBox koji javlja da su podaci uspješno učitani. Potrebno je nekih desetak sekundi da se svi podaci spreme u mapu. Kad se stisne OK na MessageBox otvara se grafičko korisničko sučelje kao na slici 30.

Slika 30. Izgled MessageBox-a
Izvor: Izradio autor

Duplim lijevim klikom na grafičko korisničko sučelje, stavlja se marker na najbliži link, prvo se postavi početni marker, a zatim krajnji marker. Da bi se dobio najbliži link koristi se par metoda. Varijabla „pLink“ se postavlja na „null“ i u nju će se spremiti najbliži link. Prva metoda „getClosestLink“ prima podatke koji sadrže geografsku širini i duljinu kliknute točke na karti. Da bi se dobile te koordinate koristi se metoda „clickedPointLonLat“. Zatim se u petlji prolazi kroz sve podatke u „sviPodatci.Values“, ova mapa sadrži sve podatke o linkovima. Za svaki „podatak“ se računa udaljenost od kliknute točke na karti uz pomoć metode „getDistanceFromPointToClosestPointOnLine“. Ako je trenutna izračunata udaljenost manja od trenutne najmanje udaljenosti, ažurira se „minDistance“ i „pLink“ postaje trenutni „podatak“. Petlja na kraju vrati vrijednost „pLink“ koja predstavlja najbliži link kliknutom mjestu na karti. Metoda „getDistanceFromPointToClosestPointOnLine“ koristi vektorski proračun i pomak koordinata kako bi odredila najbližu točku od kliknute točke. Metoda „airalDistHaversine“ računa zračnu udaljenost između dvije geografske točke koristeći Haversine formulu. Ova formula uz pomoć geografske širine i duljine računa udaljenost između dvije točke na zakrivljenoj površini Zemlje. Ako se slučajno postavi marker na krivu lokaciju on se može obrisati pomoću duplog klika kola od miša, ili ako se želi obrisati zadnji postavljeni marker to se može odraditi pomoću gumba koji se nalazi u donjem desnom kutu. Na slici 31 može se vidjeti gumb za brisanje zadnjeg markera. Poslije postavljanja markera unosi se vrijeme u kojem se kreće jer se rješava vremenski ovisno rutiranje i na taj način želi se ukazati na povećanje vremena putovanja tijekom vršnih sati. Na slici 32 vidi se grafičko sučelje poslije postavljanja markera.

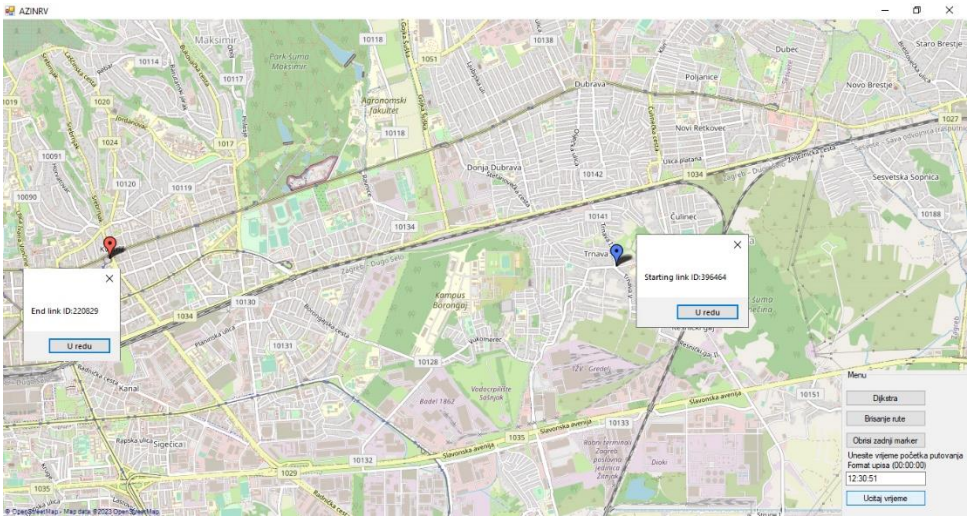
1. *Fukncija gMap_MouseDoubleClick:*
    - *Ako je „brojKlikova“ jednak 1:*
    - *Pronađi najbliži link na karti koristeći funkciju „getClosestLink“ s koordinatama klika*
    - *Nacrtaj plavi marker na karti koristeći funkciju „plotBluePinOnOSM“*
    - *Ako je „brojKlikova“ jednak 2:*
    - *Pronađi najbliži link na karti koristeći funkciju „getClosestLink“ s koordinatama klika*
    - *Nacrtaj crveni marker na karti koristeći funkciju „plotRedPinOnOSM“*

2. *Funkcija „getDistanceFromPointToClosestPointOnLine“:*
    - *Izračunaj vektore „vec_I1P“ i „vec_I1I2“ između točaka na liniji i zadane točke,*
    - *Izračunaj skalarni produkt vektora*
    - *Izračunaj normirano rastojanje „normDist“,*
    - *Izračunaj koordinate točke na liniji „clX“ i „clY“ koristeći „normDist“*
    - *Ako su koordinate izvan granice linije, korigiraj ih na najbližu točku na liniji,*
    - *Izračunaj zračnu udaljenost d između najbliže točke na liniji i zadane točke koristeći funkciju „airalDistHaversine“,*
    - *Vrati zračnu udaljenost „d“*
3. *Funkcija „arialDistHaversine“*
4. *Funkcija „getClosestLink“:*
    - *Inicijaliziraj varijablu „pLink“ na null,*
    - *Inicijaliziraj varijablu „minDistance“ na najveću vrijednost,*
    - *Prolazi kroz sve „podatke“ u listi „sviPodaci“*
    - *Pronađi najbliži link*
    - *Vrati „pLink“*



Slika 31. Gumbi koji nam omogućuju izvršavanje programa

Izvor: Izradio autor



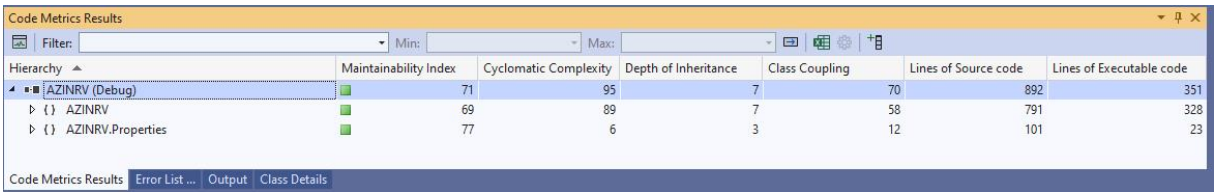
Slika 32. Izgled sučelja poslije postavljanja markera

Izvor: Izradio autor

Pritiskom na gumb Dijkstra pokreće se program koji uz pomoć Dijkstrinog algoritma i Fibonačijeve hrpe ispod sekunde iscrtava najbolju rutu od početnog markera do krajnjeg markera. Gumb se vidi na slici 31.

4.1 Kompleksnost programa i dijagram klasa

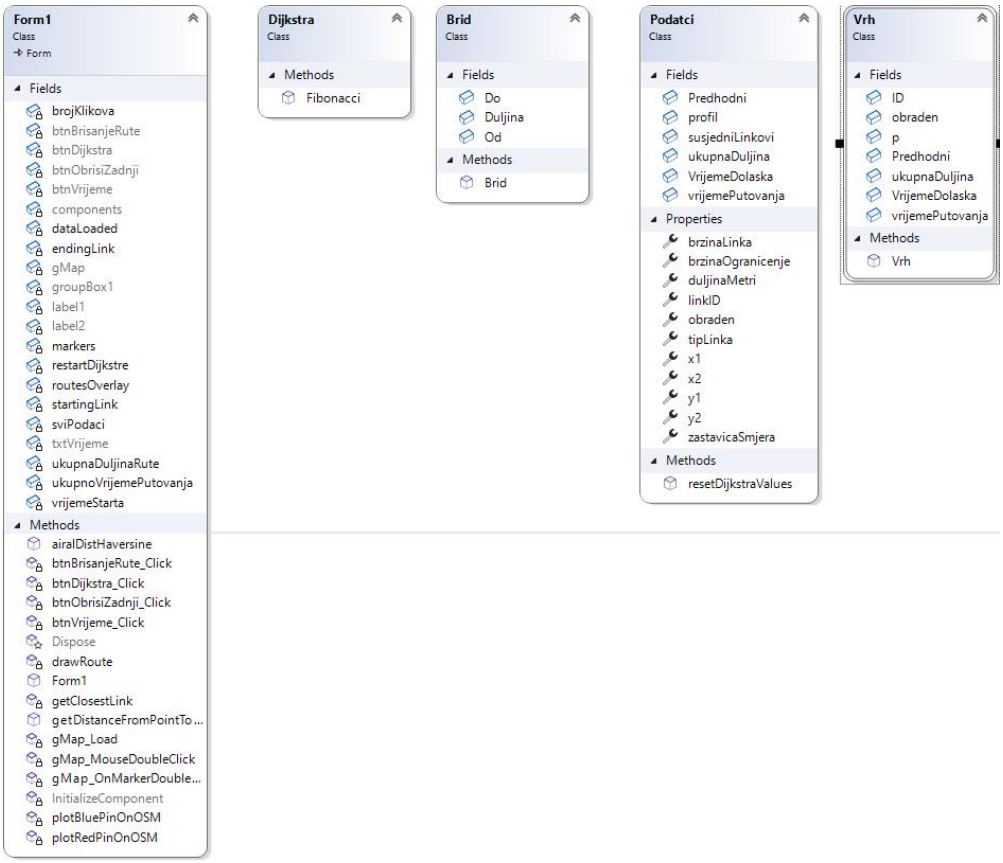
Ukupan broj linija koda (engl. Lines of Source code) u ovome programu je 892, u taj broj se ubrajaju i komentari, prazne linije i varijable. Broj linija izvršivog koda (engl. Lines of Executable code) je 351, tu se ubrajaju samo linije koda koje se izvršavaju u programu. Maintainability Indeks je metrika Visual Studio-a koja ocjenjiva koliko je kod nekog programa održiv, metrika se izračunava na temelju složenosti koda, duljina metoda i broja komentara. Raspon Maintainability indeksa je od 0 do 100, što je bliže 100 to je kod održiviji, kod u ovom programu ima ocjenu 71 te se on smatra održivim. Cyclomatic Complexity je mjera koja procjenjuje složenost programa, temelji se na broju putova u toku izvođenja programa. Cyclomatic Complexity se izračunava na temelju broja grana, petlji i uvjeta u kodu, u ovome programu je 95. Dubina nasljeđivanja (engl. Depth od Inheritance) označava broj klasa koje nasljeđuju jedna od druge, sve do bazne klase. Ona mjeri koliko je duboko svaka klasa u hijerarhiji nasljeđivanja. Dubina nasljeđivanja ovog programa je 7. Zadnja mjera je povezanost klasa (engl. Class Coupling), ona mjeri povezanost jedinstvenim klasama putem parametara, lokalnih varijabli, povratnih tipova, poziva metoda, generičkih predložaka, baznih klasa, polja definiranih na vanjskim tipovima, implementacija sučelja i dekoracija atributa. Visoka povezanost ukazuje na dizajn koji se može teško ponovo upotrijebiti i koji se može teško održavati. Povezanost klasa u ovome programu je 70, [31]. Na slici 33 se rezultati Code Metrics-a u ovom programu, informacije su izvučene preko Visual Studio-a.



Slika 33. Prikaz Code Metrics rezultata

Izvor: Izradio autor

Dijagram klasa (engl Class Dijagram) je vizualni prikaz strukture i veze između klasa u Visual Studio-u. On nam pruža pogled na klase, njihove atribute, metode i vezu između njih, [32].

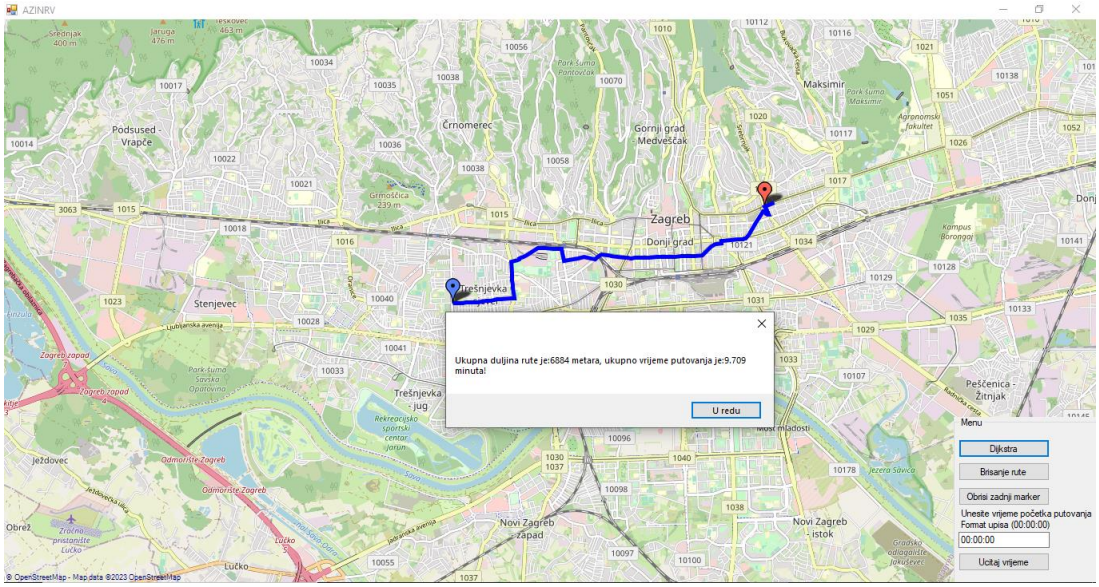


Slika 34. Izgled dijagrama klasa

Izvor: Izradio autor

4.2 Rezultati programa

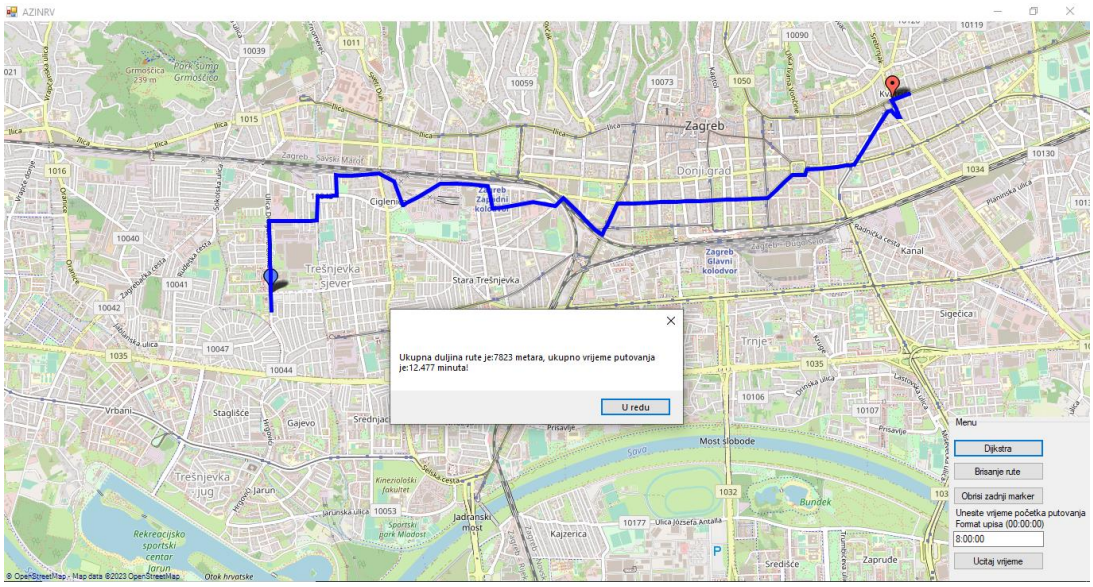
Da bi se uspješno iscrtala ruta na karti mora se koristiti dio GMap.NET biblioteke koji omogućuje iscrtavanje rute i markera na mapi, a taj dio je „GMap.NET.WindowsForms.Markers“. Korištenjem ove klase programeri mogu mijenjati pozicije markera, veličinu, boju i dosta drugih svojstva, [33, 34]. Ruta se prikazuje na mapi unutar jedne sekunde, s prikazom rute prikaže se i MessageBox koji govori koliko je duga ruta i koliko vremena treba da se dođe od početnog markera do krajnjeg markera. Na slici 35 vidi se izgled grafičkog korisničkog sučelja poslije izvršavanja programa.



Slika 35. Izgled sučelja poslije izvršavanja programa i izgled rute (00:00:00h)

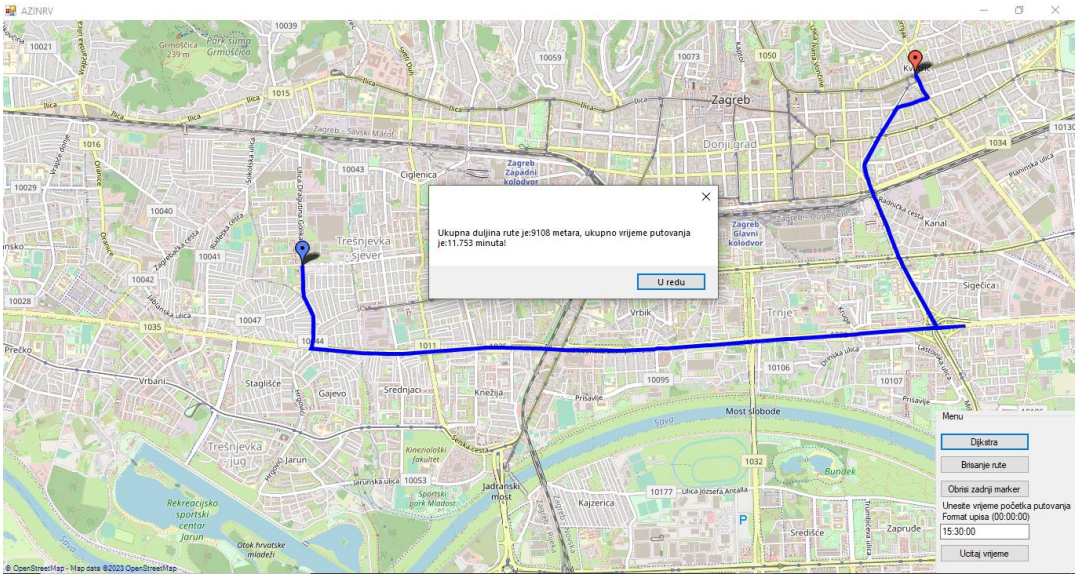
Izvor: Izradio autor

Vremensko rutiranje kroz prometnu mrežu, kao što je prikazano na slici 35, 36 i 37 pruža uvid u promjene rute ovisno o vremenu početka rute. Na slici 35 putovanje započinje u 00:00:00h, kada je zauzeće kapaciteta prometne mreže minimalno. Rezultat prve rute je ruta ukupne duljine 6884 metra, i ukupnog vremena putovanja od 9.7 minuta. Na slici 36 putovanje započinje u 8:00:00h tijekom „rush-hour-a“, kada je gustoća prometa najveća u gradu Zagrebu. Da bi se prilagodio promjenama na prometnoj mreži, program računa novu rutu koja iznosi 7823 metra, s vremenom putovanja od 12.5 minuta. Na slici 37 putovanje započinje u 15:30:00h, kada je gustoća prometa velika. Kako bi smanjio ukupno vrijeme putovanja, program ponovo prilagođava rutu. Nova ruta se drastično razlikuje od prve dvije. Program ovaj put ne sugerira prolazak kroz uži centar grada, već preporučuje korištenje avenija kako bi se izbjegla gužva. Vrijeme putovanja se smanjuje na 11.75 minuta, dok se duljina rute povećava na 9108 metara. Vremensko rutiranje omogućava analizu prometnih uvjeta i pronalazak optimalne rute ovisno o vremenu početka putovanja. Program se prilagođava gustoći prometa i uvjetima na cesti kako bi pronašao najbolju rutu.



Slika 36. Izgled rute (08:00:00h)

Izvor: Izradio autor



Slika 37. Izgled rute (15:30:00h)

Izvor: Izradio autor

5. ZAKLJUČAK

S povećanjem broja osobnih i teretnih vozila u gradu Zagrebu i u Republici Hrvatskoj, prometna zagušenja i gužve postaju svakodnevna pojava. Kako bi se riješio problem prometnog zagušenja važno je pravilno usmjeravati i voditi promet u gradovima, a optimizacija ruta vozila postaje ključna. U ovom završnom radu se rješava problem vremenski najkraćeg puta kako bismo pokušali pronaći optimalne rute za vozila. Jedan od ključnih dijelova ovog rada su GPS uređaji koji su korišteni u osobnim vozilima da bi se prikupili potrebni podatci za izračun vremenski najkraćeg puta u gradu Zagrebu. Prikupljeni GPS podatci su obrađeni i preneseni u oblik grafa, što nam omogućuje primjenu različitih algoritama za izračun vremenski najkraćeg puta.

Glavni algoritmi koji su korišteni u ovoj aplikaciji za optimizaciju rute u gradu Zagrebu su Dijkstrin algoritam i Fibonaccijeva hrpa. Primjena Fibonaccijeve hrpe u Dijsktirnom algoritmu omogućila je izračun vremenski najkraćeg puta od polazne do završne točke u vrlo kratkom vremenu, vremenska složenost takvog algoritma je $O(E + V * logV)$, to jest u svega par stotina milisekundi dobiva se vremenski najkraća ruta za graf razmatran u radu koji ima 87984 vrhova i 43992 brida. Rezultati tih algoritama su jako vrijedni vozačima jer im omogućuje efikasnije putovanje za vrijeme vršnih sati. Dobiveni rezultati se prikazuju u grafičkom korisničkom sučelju koje je razvijeno u Visual Studio-u u Windows Forms aplikaciji. Ovo sučelje pruža vizualni prikaz dobivene rute na OpenStreetMap-i, što olakšava vozačima da slijede optimalnu rutu. Korištenjem Dijkstrinog alogritma i Fibonaccijeve hrpe i općenito ove aplikacije može dovesti do smanjenja vremena putovanja, poboljšanja prometnog toka i smanjenja zagađenja okoliša.

Na kraju ovi rezultati i ova aplikacija nisu u rangu s nekim boljim programima kao što su Google karte, ali se mogu dodatno proširiti da bi se unaprijedile performanse. Može se proširiti i unaprijediti grafičko korisničko sučelje te se sama aplikacija može primijeniti na drugim gradovima sa sličnim prometnim problemima. Na kraju ovaj završni rad pruža temelje za daljnji napredak u području optimizacije ruta u urbanim sredinama.

LITERATURA

[1] M. Gendreau, G. Ghiani i E. Guerriero, Time-dependent routing problems, Amsterdam: Elsevier ltd, 2015.

[2] T. Paoletti, »MAA,« 2011. [Mrežno]. Available: https://www.maa.org/press/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem. [Pokušaj pristupa svibanj 2023].

[3] L. Rožić, T. Carić, M. Matulin, M. Ravlić, J. Fosin i A. Milošević, »Tehnički izvještaj rezultata eksperimentalnog razvoja projekta SORDITO,« Zagreb, Fakultet Prometnih Znanosti Sveučilište u Zagrebu, 2016.

[4] »Riteh fest,« 2021. [Mrežno]. Available: https://fest.riteh.hr/tvrtke-2021/mireo/. [Pokušaj pristupa svibanj 2023].

[5] T. Erdelić, T. Carić, M. Erdelić, L. Tišljarić, A. Turković i N. Jelušić, »Estimating congestion zones and travel time indexes based on the floating car data,« *Computers, Environment and Urban Systems,* pp. https://pdf.sciencedirectassets.com/271803/1-s2.0-S0198971521X00026/1-s2.0-S0198971521000119/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEEAaCXVzLWVhc3QtMSJHMEUCIDe87y%2FJz0JnI10Xm8vIdf5J6lxBzApP0ASDiyTFwY7PAiEA44kcfqdmxRqIsI2fZOfXEO06jFG%2FXRvZz4QhkdfA, 2021.

[6] S. Tawde, »Educba,« 2023. [Mrežno]. Available: https://www.educba.com/c-sharp-system-dot-io/ . [Pokušaj pristupa svibanj 2023].

[7] Knowledgehut, »Knowledgehut,« [Mrežno]. Available: https://www.knowledgehut.com/tutorials/csharp/csharp-classes-objects. [Pokušaj pristupa svibanj 2023].

[8] M. Radanović, »ManuelRadanovic,« 11 prosinac 2015. [Mrežno]. Available: https://www.manuelradovanovic.com/2015/12/klase-u-c-programskom-jeziku.html. [Pokušaj pristupa svibanj 2023].

[9] TutorialTeacher, »TutorialTeacher,« 2023. [Mrežno]. Available: https://www.tutorialsteacher.com/csharp/csharp-dictionary. [Pokušaj pristupa svibanj 2023].

[10] P. Jingyoung, R. DSouza i M. Zhang, »SpringerOpen,« 2022. [Mrežno]. Available: https://appliednetsci.springeropen.com/articles/10.1007/s41109-022-00521-8. [Pokušaj pristupa svibanj 2023].

[11] F. Karimi, »Researchgate,« veljača 2015. [Mrežno]. Available: https://www.researchgate.net/figure/Koenigsberg-bridges-Euler-formulated-the-problem-of-finding-a-way-to-walk-through-the_fig1_282099701. [Pokušaj pristupa svibanj 2023].

[12] Geeksforgeeks, »Geeksforgeeks,« 2023. [Mrežno]. Available: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/. [Pokušaj pristupa svibanj 2023].

[13] T. Carić i J. Fosin, »Osnovni pojmovi teorije grafova,« u *Optimizacija prometnih procesa*, Zagreb, Fakultet Prometnih Znanosti Sveučilište u Zagrebu, 2023.

[14] Wikipedia, »Wikipedia,« 2023. [Mrežno]. Available: https://en.wikipedia.org/wiki/Directed_graph. [Pokušaj pristupa svibanj 2023].

[15] Study, »Study,« 2023. [Mrežno]. Available: https://study.com/academy/lesson/weighted-graphs-implementation-dijkstra-algorithm.html. [Pokušaj pristupa svibanj 2023].

[16] G. Cox, »Baeldung,« 2022. [Mrežno]. Available: https://www.baeldung.com/cs/dijkstra. [Pokušaj pristupa svibanj 2023].

[17] JavaTpoint, »JavaTpoint,« 2023. [Mrežno]. Available: https://www.javatpoint.com/dijkstras-algorithm. [Pokušaj pristupa svibanj 2023].

[18] Wikipedia, »Wikipedia,« 2023. [Mrežno]. Available: https://en.wikipedia.org/wiki/Fibonacci_heap. [Pokušaj pristupa svibanj 2023].

[19] Geeksforgeeks, »Geeksforgeeks,« 2023. [Mrežno]. Available: https://www.geeksforgeeks.org/fibonacci-heap-set-1-introduction/. [Pokušaj pristupa svibanj 2023].

[20] M. Mahajan, »Geeksforgeeks,« [Mrežno]. Available: https://www.geeksforgeeks.org/fibonacci-heap-insertion-and-union/. [Pokušaj pristupa svibanj 2023].

[21] M. Mahajan, »Geeksforgeeks,« 2023. [Mrežno]. Available: https://www.geeksforgeeks.org/fibonacci-heap-deletion-extract-min-and-decrease-key/. [Pokušaj pristupa svibanj 2023].

[22] K. Bailey, »Github,« 2023. [Mrežno]. Available: https://kbaile03.github.io/projects/fibo_dijk/fibo_dijk.html. [Pokušaj pristupa svibanj 2023].

[23] J. Juviler, »Hubspot,« 2022. [Mrežno]. Available: https://blog.hubspot.com/website/what-is-gui. [Pokušaj pristupa svibanj 2023].

[24] Microsoft, »Microsoft,« 2023. [Mrežno]. Available: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-7.0. [Pokušaj pristupa svibanj 2023].

[25] Syncfusion, »Syncfusion,« 2023. [Mrežno]. Available: https://help.syncfusion.com/windowsforms/installation/install-nuget-packages. [Pokušaj pristupa svibanj 2023].

[26] A. Saini, »Geeksforgeeks,« 2019. [Mrežno]. Available: https://www.geeksforgeeks.org/label-in-c-sharp/. [Pokušaj pristupa svibanj 2023].

[27] CSharpNetInformations, »CSharpNetInformations,« 2023. [Mrežno]. Available: http://csharp.net-informations.com/gui/button.htm. [Pokušaj pristupa svibanj 2023].

[28] M. Priyadharshini, »C#Corner,« 2018. [Mrežno]. Available: https://www.c-sharpcorner.com/article/working-with-textbox-control-in-windows-forms-application-using-visual-stuudio-2/. [Pokušaj pristupa svibanj 2023].

[29] M. Chand, »C#Corner,« 2018. [Mrežno]. Available: https://www.c-sharpcorner.com/uploadfile/mahesh/groupbox-in-C-Sharp/. [Pokušaj pristupa svibanj 2023].

[30] CodeProject, »CodeProject,« 2013. [Mrežno]. Available: https://www.codeproject.com/Articles/32643/GMap-NET-Great-Maps-for-Windows-Forms-and-Presenta. [Pokušaj pristupa svibanj 2023].

[31] Microsoft, »Microsoft,« 2022. [Mrežno]. Available: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022. [Pokušaj pristupa svibanj 2023].

[32] Microsoft, »Microsoft,« 2023. [Mrežno]. Available: https://learn.microsoft.com/en-us/visualstudio/ide/class-designer/designing-and-viewing-classes-and-types?view=vs-2022. [Pokušaj pristupa svibanj 2023].

[33] IndependentSoftware, »IndependentSoftware,« 2013. [Mrežno]. Available: http://www.independent-software.com/gmap-net-tutorial-routes.html. [Pokušaj pristupa svibanj 2023].

[34] IndependentSoftware, »IndependentSoftware,« 2016. [Mrežno]. Available: http://www.independent-software.com/gmap-net-beginners-tutorial-maps-markers-polygons-routes-updated-for-vs2015-and-gmap1-7.html. [Pokušaj pristupa svibanj 2023].

POPIS SLIKA

POPIS TABLICA

Sveučilište u Zagrebu
Fakultet prometnih znanosti
Vukelićeva 4, 10000 Zagreb

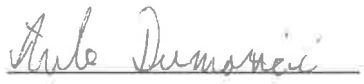
IZJAVA O AKADEMSKOJ ČESTITOSTI I SUGLASNOSTI

Izjavljujem i svojim potpisom potvrđujem da je završni rad
(*vrsta rada*)
isključivo rezultat mojega vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju upotrijebljene bilješke i bibliografija. Izjavljujem da nijedan dio rada nije napisan na nedopušten način, odnosno da je prepisan iz necitiranog rada te da nijedan dio rada ne krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Svojim potpisom potvrđujem i dajem suglasnost za javnu objavu završnog/diplomskog rada pod naslovom **Rješavanje problema vremenski najkraćeg puta**, u Nacionalni repozitorij završnih i diplomskih radova ZIR.

Student/ica: Ante Dumančić

U Zagrebu, 12.6.2023



(ime i prezime, *potpis*)