Algoritmi za pronalaženje najkraćih puteva u mreži i njihova programska implementacija

Maligec, Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA

I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

ALGORITMI ZA PRONALAŽENJE NAJKRAĆIH PUTEVA U MREŽI I NJIHOVA PROGRAMSKA IMPLEMENTACIJA

Diplomski rad

Matej Maligec

Osijek, 2017.

Sadržaj

1. UVOD

U današnjem dobu visoko razvijenih komunikacijskih sustava i mreža, jedan od ključnih zadataka je pronaći najkraći put u mreži od početne točke do krajnje točke. Ovom problematikom bave se mnoga istraživanja, te su ona rezultirala pronalaskom nekoliko različitih rješenja, odnosno algoritama za rješavanje pitanja najkrećeg puta u mreži. Prema [1], razvoj, testiranje i učinkovita implementacija algoritama za pronalaženje najkraćih puteva u mreži su važni pojmovi kako u računarstvu, tako i u geografiji, menadžmentu, transportu itd. Kada je u pitanju odabir algoritma kojim se pokušava riješiti problem najkraćeg puta u mreži, prigodan algoritam će se odabrati ovisno o zadanim parametrima mreže te samom vremenu izvođenja algoritma. U ovom radu biti će opisana tri algoritma za pronalaženje najkraćih puteva: Djikstrin, Bellman-Fordov te Floyd-Warshallov algoritam. Svaki od navedenih algoritama biti će obrađen uz primjer na zadanoj mreži, te će isto tako biti implementiran u softversku aplikaciju kojom će korisnik, za zadane parametre pojedine mreže, moći pronaći najkraći put od početne do krajnje točke (ukoliko su zadovoljeni uvjeti). Nadalje, ovaj rad opisuje prednosti i nedostatke navedenih algoritama u svrhu usporedbe njihove efikasnosti.

2. NAJKRAĆI PUTEVI JEDNOG IZVORA

Prije detaljne obrade algoritama za traženje najkraćih puteva u mreži, potrebno je definirati nekoliko pojmova kojima opisujemo mrežu, odnosno graf na kojima će se najkraći put tražiti. Prema [2], usmjereni ili direktni graf $G=(V,E)$ sastoji se od konačnog nepraznog skupa čvorova V i skupa usmjerenih grana E, odnosno uređenih parova različitih čvorova iz V. Šetnja je slijed čvorova (v_1, v_2, ..., v_k), gdje parovi (v_1, v_2), (v_2, v_3), ..., (v_{1-k}, v_k) predstavljaju grane grafa. Put je šetnja u kojoj se ne ponavljaju čvorovi. Ciklus ili petlja je šetnja (v_1, v_2, ..., v_k), u kojoj je početni čvor jednak odredišnom čvoru, odnosno $v_1 = v_k$. Također je vrlo bitno spomenuti pojam „relaksacije“ grane, odnosno korištenja čvorova sa manjom težinskom vrijednosti kako bi se smanjio najkraći put do određenog čvora. Prema [5], kontekst „relaksacije“ proizlazi iz analogije između predviđene vrijednosti najkraćeg puta te opruge. U inicijalnoj fazi izvođenja algoritama, vrijednost najkraćeg puta je previsoko procjenjena, što možemo usporediti sa rastegnutom oprugom. Kako se pronalaze novi putevi u mreži, ta vrijednost se može smanjivati, što je usporedivo sa opuštanjem opruge. Konačno, ako postoji najkraći put, on je analogan sa potpuno opuštenom oprugom. Prethodno navedeni algoritmi koji se koriste za pronalaženje najkraćeg puta u mreži koriste se na usmjerenim ili direktnim grafovima, odnosno digrafima. Problem najkraćeg puta jednog izvora svodi se na pronalazak najkraćeg puta od zadanog izvorišnog čvora $s \in V$ do bilo kojeg čvora u mreži $v \in V$. Prema [3], algoritam za rješavanje ovog problema također rješava i sljedeće probleme:

- Najkraći put jednog odredišta: pronaći najkraći put do odredišnog čvora t od svakog čvora v. Ovaj problem se svodi na problem najkraćeg puta jednog izvora na način da se okrenu smjerovi svake grane u mreži.
- Najkraći put pojedinog para: pronaći najkraći put od čvora u do čvora v za zadane čvorove u i v. Rješavanjem problema najkraćeg puta jednog izvora također rješavamo i ovaj problem.
- Najkraći putevi svih parova: pronaći najkraći put od čvora u do čvora v za svaki par čvorova u i v. Ovaj problem se može riješiti algoritmom za pronalaženje najkraćih puteva pojedinog izvora, no postoje i algoritmi koji to rješavaju na brži i efikasniji način.

Grafovi u nekim slučajevima mogu imati negativne težinske vrijednosti grana. Ako graf $G=(V,E)$ ne sadrži negativne cikluse ili petlje dostupne iz izvorišnog čvora, tada za $v \in V$ vrijedi da je težinska vrijednost najkraćeg puta $\delta(s,v)$ dobro definirana, te može poprimiti negativnu vrijednost. No ako postoji negativni ciklus koji počinje u čvoru s, tada težinske vrijednosti najkraćeg puta nisu dobro definirane; uvijek možemo pronaći kraći put sa manjom težinskom vrijednosti te ulazimo u negativni ciklus. Tada se težinska vrijednost najkraćeg puta definira kao: $\delta(s,v) = -\infty$. Neki algoritmi rade samo sa nenegativnim težinskim vrijednostima grana (Dijkstrin algoritam), dok drugi algoritmi funkcioniraju sa negativnim težinskim vrijednostima grana sve dok u grafu ne postoji negativni ciklus (Bellman-Fordov algoritam).

3. ALGORITMI ZA TRAŽENJE NAJKRAĆEG PUTA

3.1 Dijkstrin Algoritam

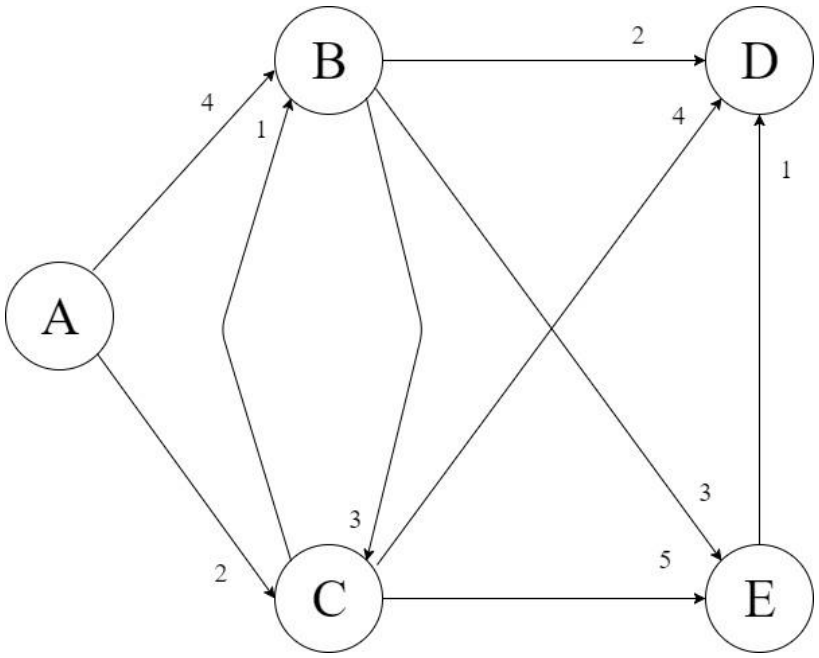
Prema [3] i [4], Dijkstrin algoritam je postupak kojim se pronalazi najkraći put pojedinog izvora u usmjerenom grafu $G=(V,E)$ u kojem sve grane imaju nenegativnu vrijednost. Pseudokod Dijkstrinog algoritma je prikazan na sljedeći način:

1	*distance*[*s*] = 0
2	*for all* $v \in V - \{s\}$
3	*do dist*[*v*] = ∞
4	*S* = 0
5	*Q* = *V*
6	*while* $Q \neq 0$
7	*do u = min(Q,distance)*
8	$S = S \cup \{u\}$
9	*for all v* ∈ *neighbour*[*u*]
10	*do if distance*[v] > *dist*[u] + *w*(*u*,*v*)
11	*then dist*[*v*] = *dist*[*u*] + *w*(*u*, *v*)
12	*return*(*dist*[*v*])

Tab 2.1. Pseudokod Dijsktrinog dijagrama

Na početku samog izvršavanja algoritma, udaljenost do izvorišnog čvora je postavljena na nulu, dok su vrijednosti najkraćeg puta do svih ostalih čvorova postavljene u beskonačnost. Varijabla *S* predstavlja skup svih „posjećenih“ čvorova u mreži na putu do odredišta, dok varijabla *Q* predstavlja „red čekanja“ u kojem se inicijalno nalaze svi čvorovi u mreži. Nakon toga se u *while* petlji odvija sljedeće: dok god skup *Q* nije prazan petlja će odabrati čvor *u* iz skupa *Q* koji ima najmanju udaljenost od čvora u kojem se petlja trenutno nalazi, te da dodati u skup posjećenih čvorova *S*. Nakon toga će provjeriti postoji li novi najkraći put, te ukoliko postoji, pohraniti njegovu vrijednost u varijablu *dist*[*v*]. Nakon što petlja završi, pseudokod će vratiti konačnu vrijednost varijable *dist*[*v*] koja označava najkraći put do odredišnog čvora *v*.

Na sljedećem primjeru biti će demonstrirano kako primjeniti Dijkstrin algoritam na zadanu mrežu. Zadan je sljedeći graf:



Sl 2.1. Primjer usmjerenog grafa s pozitivnim vrijednostima grana

Prvi korak je odabrati početni, odnosno izvorišni čvor. U ovom slučaju to će biti čvor A. Nakon toga udaljenost do čvora A postavlja se u nulu, dok će sve ostale udaljenosti biti beskonačno velike. Isto tako je potrebno voditi računa o „neposjećenim“ čvorovima. Ovaj korak reprezentiramo tablicom 2.2.

Korak	A	B	C	D	E
1	0	∞	∞	∞	∞
Neposjećeni čvorovi: A, B,C,D,E					

Tab 2.2. Prvi korak Dijkstrinog algoritma za primjer

Sljedeći korak je proučiti grane koje izlaze iz čvora A. Kako je prikazano na slici 2.1, iz čvora A moguće je doći do čvorova B i C, pa je potrebno nadopuniti tablicu 2.2 težinskim vrijednostima grana koje vode do čvorova B i C, te dobijamo tablicu 2.3.

Korak	A	B	C	D	E
1	0	∞	∞	∞	∞
2	0	4	2	∞	∞
Neposjećeni čvorovi: B,C,D,E					

Tab 2.3. Drugi korak Dijkstrinog algoritma za primjer

Nakon toga se odabire najmanja vrijednost grane koja odlazi do čvora koji još nije posjećen. U ovom slučaju je to čvor C. Čvor C se uklanja sa liste neposjećenih čvorova, te se tablica nadopunjava vrijednostima grana koje odlaze iz čvora C. Vidljivo je da je čvor B sada dostupan čvoru A uz vrijednost 3, prolaskom kroz čvor C. Također, čvorovi D i E postaju dostupni po prvi puta.

Korak	A	B	C	D	E
1	0	∞	∞	∞	∞
2	0	4	2	∞	∞
3	0	3	2	6	7
Neposjećeni čvorovi: B,D,E					

Tab 2.4 Treći korak Dijkstrinog algoritma za primjer

Proces se ponavlja za svaki neposjećeni čvor grafa do kojeg dolazimo iz trenutnog čvora sa najmanjom vrijednošću grane, te mijenjamo vrijednosti u tablici samo ukoliko postoje nove i/ili kraće vrijednosti najkraćeg puta u odnosu na početni čvor. Nakon što su posjećeni svi čvorovi grafa, tablica vrijednosti će izgledati sljedeće:

Korak	A	B	C	D	E
1	0	∞	∞	∞	∞
2	0	4	2	∞	∞
3	0	3	2	6	7
4	0	3	2	5	6
5	0	3	2	5	6
6	0	3	2	5	6
Neposjećeni čvorovi: /					

Tab 2.5. Konačna tablica Dijkstrinog algoritma za primjer

Iz tablice 2.5 možemo vidjeti vrijednosti najkraćeg puta od čvora A do bilo kojeg čvora u grafu. Također je vidljivo da algoritam ima $V - 1$ koraka, gdje je V broj čvorova u mreži, odnosno da će se *while* petlja pseudokoda izvršavati V puta (korak 1 je početni korak pseudokoda te se nalazi prije petlje). Iz pseudokoda algoritma, te iz koda programa koji se nalazi u Prilogu, može se zaključiti da se vremenska kompleksnost Dijkstrinog algoritma O može izvesti kao V^2. To je vidljivo iz linija 9-11, gdje se prilikom „posjete“ svakog čvora ažurira udaljenost do njemu susjednih čvorova u grafu te lista koja sadrži udaljenost najkraćih puteva od izvorišnog čvora do svih ostalih čvovora.

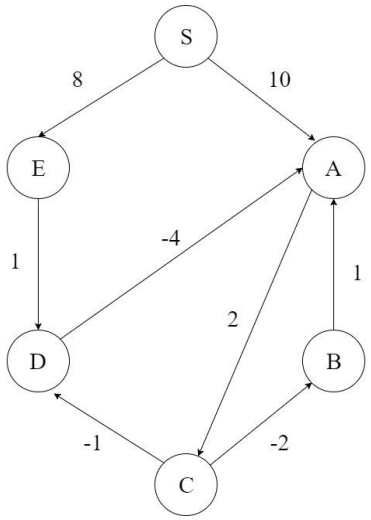
3.2 Bellman-Fordov algoritam

Prema [3], Bellman-Fordov algoritam služi za rješavanje problema najkraćeg puta jednog izvora u slučaju kada težinske vrijednosti grana u grafu, za razliku kod Dijkstrinog algoritma, mogu poprimiti negativnu vrijednost. Ovaj algoritam vraća *boolean* vrijednost koja kazuje postoji li u grafu negativni ciklus koji je dostupan izvorišnom čvoru, te u tom slučaju algoritam ne može riješiti problem najkraćeg puta. Ako takav ciklus ne postoji, algoritam pronalazi najkraće puteve te njihove težinske vrijednosti. Pseudokod Bellman-Fordovog algoritma prikazan je na sljedeći način:

1	*Inicijalizacija grafa i izvorišnog čvora*
2	*for i =1 to* $|G.V| - 1$
3	*for each edge* (*u*,*v*)
4	*relax*(*u*, *v*, *w*)
5	*for each edge* (*u*,*v*)
6	*if v.d > u.d + w*(*u*,*v*)
7	*return FALSE*
8	*return TRUE*

Tab 2.6. Pseudokod Bellman-Fordovog algoritma

Algoritam u svojoj inicijalnoj fazi postavlja udaljenost od izvorišnog čvora u nulu prema samome sebi, te beskonačnost prema ostalim čvorovima u grafu. Zatim, u koracima 2-4, prolazi kroz svaki čvor u grafu te ažurira tablicu najkraćeg puta od izvorišnog čvora do ostalih čvorova u mreži, odnosno „relaksira“ vrijednosti puta, ukoliko je to moguće. Nadalje, u koracima 5-8 vrši provjeru u svrhu pronalaska negativnog ciklusa, ukoliko on postoji, te vraća pripadnu *boolean* vrijednost. Proces izvođenja Bellman-Fordovog algoritma prikazati ćemo na primjeru grafa koji se nalazi na slici 2.2.



Sl 2.2. Primjer usmjerenog grafa s negativnim vrijednostima grana

Iz 2. linije pseudokoda Bellman-Fordovog algoritma možemo zaključiti da će algoritam za graf proći kroz najviše 5 iteracija, jer sadrži 6 čvorova. Ukoliko se vrijednosti najkraćih puteva ne promijene u dvije uzastopne iteracije koje se odvijaju prije maksimalnog mogućeg broja iteracija, algoritam izlazi iz petlje opisane linijama 2-4 pseudokoda. Kao izvorišni čvor odabran je čvor S. Zatim je potrebno napraviti jednak prvi korak kao u Dijkstrinom algoritmu, odnosno postaviti udaljenost do izvorišnoga čvora u 0, dok vrijednost udaljenosti do svih odredišnih čvorova postavljamo u beskonačnost. U svakoj iteraciji potrebno je proučiti sve grane grafa, odnosno posjetiti svaki čvor, te ovisno o tome postoji li novi najkraći put do čvora koji je sa trenutnim posjećenim čvorom povezan izlaznom granom iz trenutnog čvora, ažurirati tablicu najkraće udaljenosti čvorova. U tablici 2.7 prikazana je prva iteracija Bellman-Fordovog algoritma za graf na slici 2.2.

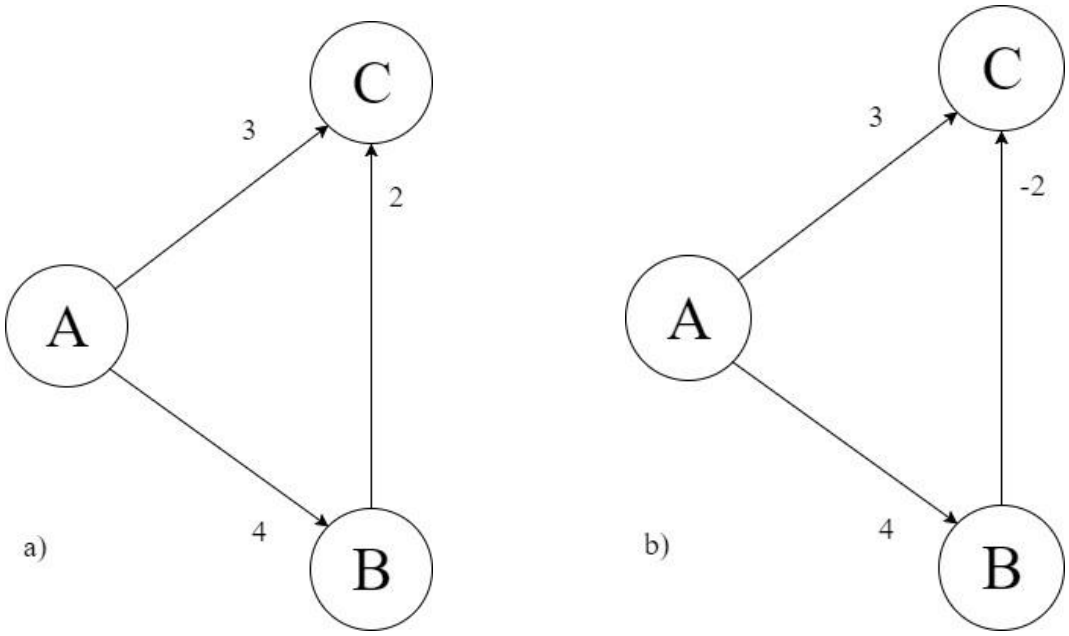
Korak	S	A	B	C	D	E
1	0	∞	∞	∞	∞	∞
2	0	10	∞	∞	∞	8
3	0	10	∞	12	∞	8
4	0	10	∞	12	∞	8
5	0	10	10	12	∞	8
6	0	10	10	12	∞	8
7	0	10	10	12	9	8
Neposjećeni čvorovi: /						

Tab 2.7. Prva iteracija Bellman-Fordovog Algoritma za navedeni primjer

Ovakvu iteraciju potrebno je napraviti $V - 1$ puta, gdje je V broj čvorova na grafu, te time izlazimo iz petlje opisane linijama 2-4 pseudokoda. Linije 5-7 u principu rade V – tu iteraciju, te se njome provjerava postoji li promjena u tablici udaljenosti najkraćih puteva. Ukoliko takva promjena postoji, algoritam će zaključiti da na grafu postoji negativni ciklus te je nemoguće odrediti vrijednost najkraćeg puta zbog toga što možemo koristiti takav negativan ciklus nebrojeno mnogo puta i smanjivati vrijednost najkraćeg puta. No ako nema promjene u vrijednosti najkraćeg puta, algoritam će zaključiti da nema negativnog ciklusa te ispisati vrijednosti najkraćih puteva na grafu. Vremenska kompleksnost Bellman-Fordovog algoritma O se može izraziti kao $V \times E$, gdje je V broj svih čvorova u grafu, a E broj svih grana u grafu.

3.3 Usporedba Dijkstrinog i Bellman-Fordovog algoritma

Dijkstrin i Bellman-Fordov algoritam imaju isti rezultat, odnosno oba algoritma daju najkraći put od izvorišnog čvora do svih ostalih čvorova u grafu. Kako je ranije spomenuto, Dijkstrin dijagram ne može raditi sa negativnim težinskim vrijednostima grana, dok Bellman-Fordov algoritam može, no oba algoritma ne mogu dati rezultat ukoliko u grafu postoji negativan ciklus. Razlika između ova dva algoritma, te razlog zašto Dijkstrin algoritam ne radi sa negativnim vrijednostima je u tome što je Dijkstrin algoritam tzv. „pohlepan“ algoritam. Takav tip algoritma je pobliže opisan na sljedećem primjeru.



Slika 2.3. a) graf bez negativnih vrijednosti grana; b) graf s negativnom vrijednošću

U grafu koji je prikazan na slici 2.3. na primjeru a), Dijkstrin algoritam može točno zaključiti da je najkraći put od čvora A do čvora C ima vrijednost 3. No ako vrijednost grane od čvora B do čvora C promijenimo iz 2 u -2, Dijkstrin algoritam će u prvoj iteraciji ponovno upisati podatak da najkraći put od čvora A do čvora C ima vrijednost 3, te zatvoriti iteraciju tako da se više ne može izvršiti

ažuriranje vrijednosti najkraćeg puta. Ali kako je vidljivo na slici 2.3 iz primjera b), najkraći put od čvora A do čvora C zapravo ima vrijednost 2 te prolazi kroz čvor B. Da bi se izbjegle ovakve situacije, za računanje grafova s negativnim težinskim vrijednostima grana koristi se Bellman-Fordov algoritam.

3.4 Floyd-Warshallov algoritam

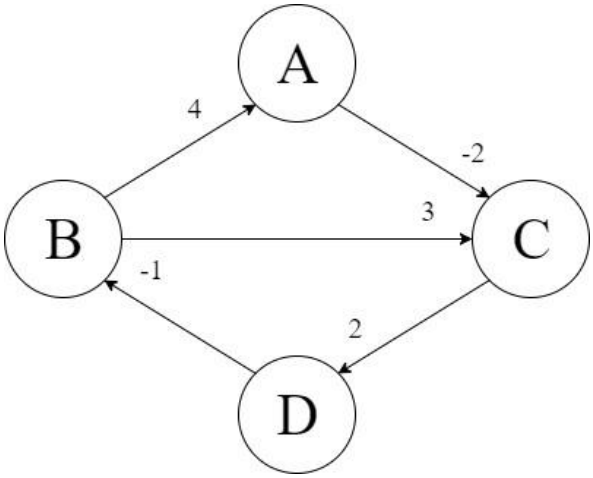
Prema [3], Floyd-Warshallov algoritam rješava problem najkraćeg puta svih parova čvorova na usmjerenom grafu *G=(V,E)*. Grane grafa mogu poprimiti negativne vrijednosti, no vrijedi pretpostavka kao i kod Bellman-Fordovog algoritma, a to je da na grafu ne postoje negativni ciklusi. Prema [3] i [6], ovaj algoritam kao svoj unos koristi matricu udaljenosti *W* koja je dimenzija $n \times n$, gdje je *n* broj čvorova u usmjerenom grafu. Algoritam će kao svoj rezultati ispisati matricu najkraćih puteva $D^{(n)}$. Pseudokod Floyd-Warshallovog algoritma prikazan je u tablici 2.8.

1	*n = W.rows*
2	$D^{(n)} = W$
3	*for k = 1 to n*
4	*let* $D^{(k)} = d_{ij}^{(k)}$ *be a new n × n matrix*
5	*for i* = 1 *to n*
6	*for j* = 1 *to n*
7	$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
8	*return* $D^{(n)}$

Tab 2.8. Pseudokod Floyd-Warshallovog algoritma

U početne dvije linije pseudokoda definira se matrica $D^{(n)}$. Ona sadrži težinske udaljenosti između čvorova definirane usmjerenim grafom, te te iste vrijednosti stavlja u odgovarajuća polja u matrici (npr. ukoliko su čvorovi 2 i 3 spojeni granom težine 5, broj 5 se postavlja na mjesto u matrici koje odgovara redu 2 i stupcu 3). Zatim slijedi niz ugnježđenih *for* petlji čijim se izvođenjem popunjavaju preostali prazni elementi, te ažuriraju već postojeće vrijednosti najkraćih puteva

između pojedinih parova čvorova. Floyd-Warshallov algoritam prikazan je na primjeru u daljnjem tekstu.



Sl 2.4. Primjer grafa na kojemu je izvršen Floyd-Warshallov algoritam

Na graf koji je prikazan na slici 2.4. primjenjen je Floyd-Warshallov algoritam. Kako je ranije navedeno, rezultat Floyd-Warshallovog algoritma je matrica. Kako Floyd-Warshallov algoritam ima vremensku kompleksnost *O* vrijednosti V^3, što je vidljivo iz pseudokoda, to bi značilo da konačna matrica vrijednosti najkraćih puteva ima 48 mogućih međurješenja, ovisno o potrebi za ažuriranjem vrijednosti najkraćih puteva. Radi jednostavnosti, prikazana je samo konačna matrica najkraćih puteva.

	A	B	C	D
A	0	-1	-2	0
B	4	0	2	4
C	5	1	0	2
D	3	-1	1	0

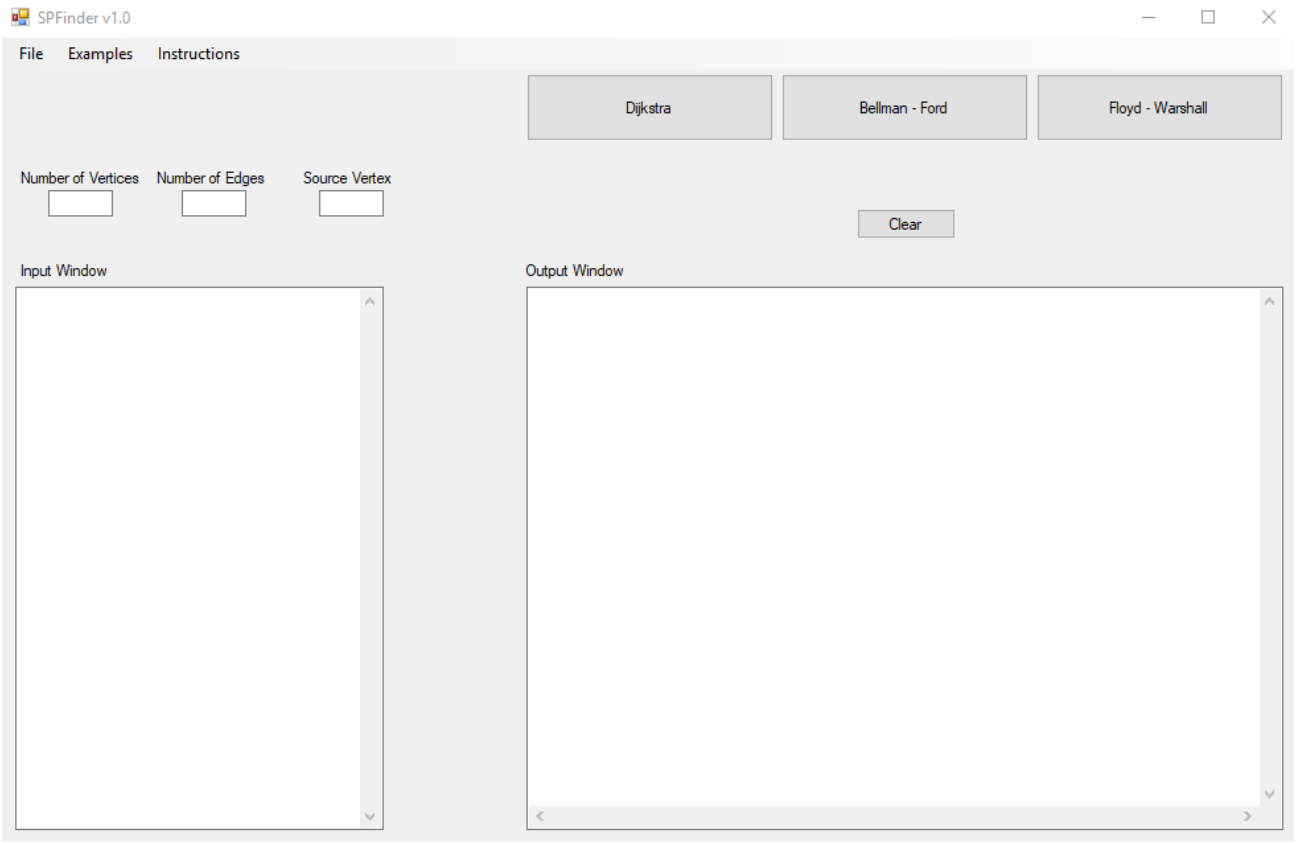
Tab 2.9. Prikaz konačnih vrijednosti najkraćih puteva za Floyd-Warshallov algoritam

Kako je ranije spomenuto, rezultantna matrica koja prikazuje vrijednosti najkraćih puteva ima V^3 međurješenja, gdje je V broj čvorova u grafu. To označava maksimalno vrijeme koje bi Floyd-Warshallovom algoritmu bilo potrebno da izračuna vrijednosti najkraćih puteva u matrici. Međutim, ukoliko ažuriranje vrijednosti najkraćih puteva nije potrebno u svakom međukoraku, to će smanjiti vrijeme potrebno za izvršavanje Floyd-Warshallovog algoritma. Isto tako je bitno primjetiti da su vrijednosti najkraćih puteva na glavnoj dijagonali rezultantne matrice jednake nuli. To proizlazi iz zaključka da je najkraća udaljenost od izvorišnog do odredišnog čvora, ukoliko se radi o istom čvoru, jednaka nuli.

4. PROGRAMSKA IMPLEMENTACIJA ALGORITAMA

4.1 Opis programa

Kao dio ovog diplomskog rada razvijen je program nazvan SPFinder (Shortest Path Finder) , koji objedinjuje ranije navedena 3 algoritma na jednom mjestu te omogućuje korisniku da za zadani graf izvrši prikladan algoritam te pronađe najkraće puteve u grafu. Program je razvijen u Microsoft Visual Studiu 2013 u programskom jeziku C#, te u obliku Windows Form-e, koja je vrlo prepoznatljiva bilo kojem korisniku koji se susretne s ovim programom.

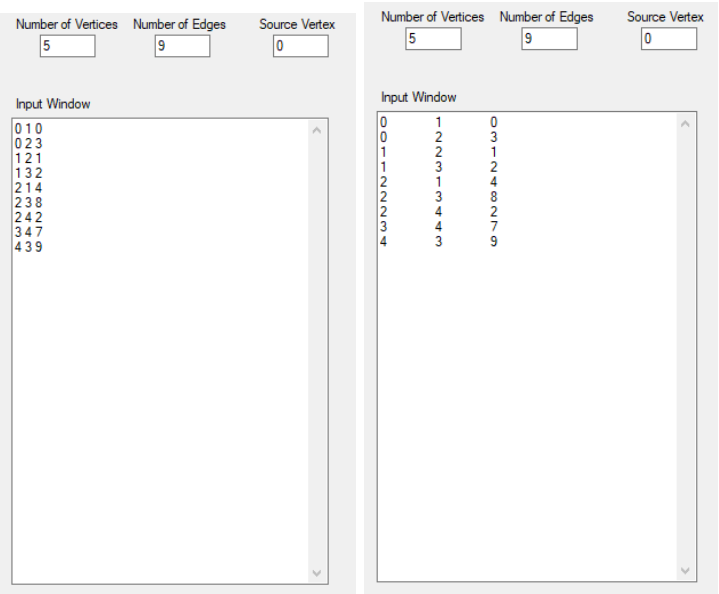


Sl 4.1. Prikaz sučelja programa SPFinder

Sučelje programa se sastoji od nekoliko elemenata: trake izbornika u kojoj se nalaze razne opcije (spremanje, učitavanje, pokazni primjeri, upute za korištenje), prozora za unos relevantnih podataka o grafu, gumba za odabir izvođenja željenog algoritma, prozora za ispis izlaznih podataka, te gumba za čišćenje podataka. U ovoj verziji program radi samo sa cjelobrojnim vrijednostima. Detaljan prikaz koda sa komentarima pojedinih dijelova nalazi se u Prilogu.

4.2 Upute za korištenje

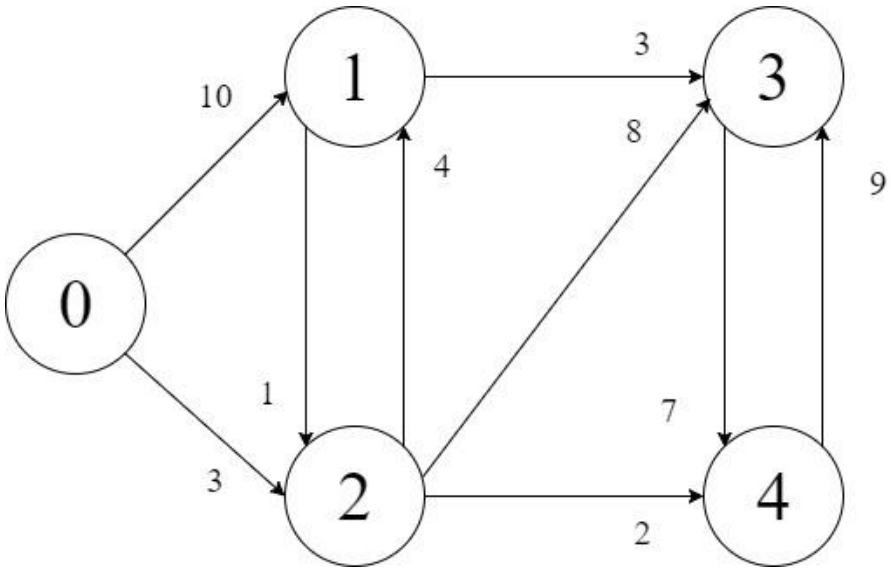
U tri prozora za unos u gornjem lijevom kutu potrebno je unijeti redom: broj čvorova u grafu, broj grana u grafu, te početni čvor (početni čvor je potrebno unijeti samo za izvođenje Dijkstrinog i Bellman-Fordovog algoritma). Čvorovi se numeriraju cjelobrojnim vrijednostima počevši od nule (prvi čvor grafa ima oznaku 0). Program radi na principu obrade matričnih podataka, te je stoga informacije o grafu potrebno prikazati u obliku matrice. To se izvodi na način da se u *Input Window* unose tri vrijednosti u jedan red. Vrijednosti mogu biti odvojene jednostrukim razmakom ili tabulatorom, te je nakon unosa posljednje vrijednosti u jednom redu potrebno prijeći u novi red. Tako formatirane vrijednosti su u biti informacije o granama grafa, odnosno predstavljaju redom: izvorišni čvor grane, odredišni čvor grane i težinsku vrijednost grane. Ove vrijednosti je potrebno unijeti za svaku granu na grafu. Te vrijednosti će biti spremljene u matricu veličine *3 x e*, gdje je *e* broj grana na grafu. Informacije o grafu se također mogu unijeti u program iz bilo koje datoteke koja je u *.txt* formatu, te u kojoj su informacije o granama oblikovane na ranije naveden način. Ovaj korak se izvodi tako da se u traci izbornika odabere opcija *File*, te zatim opcija *Load*. Nakon toga će se prikazati dijaloški okvir u kojem korisnik odabire datoteku u kojoj se nalaze podatci o grafu.

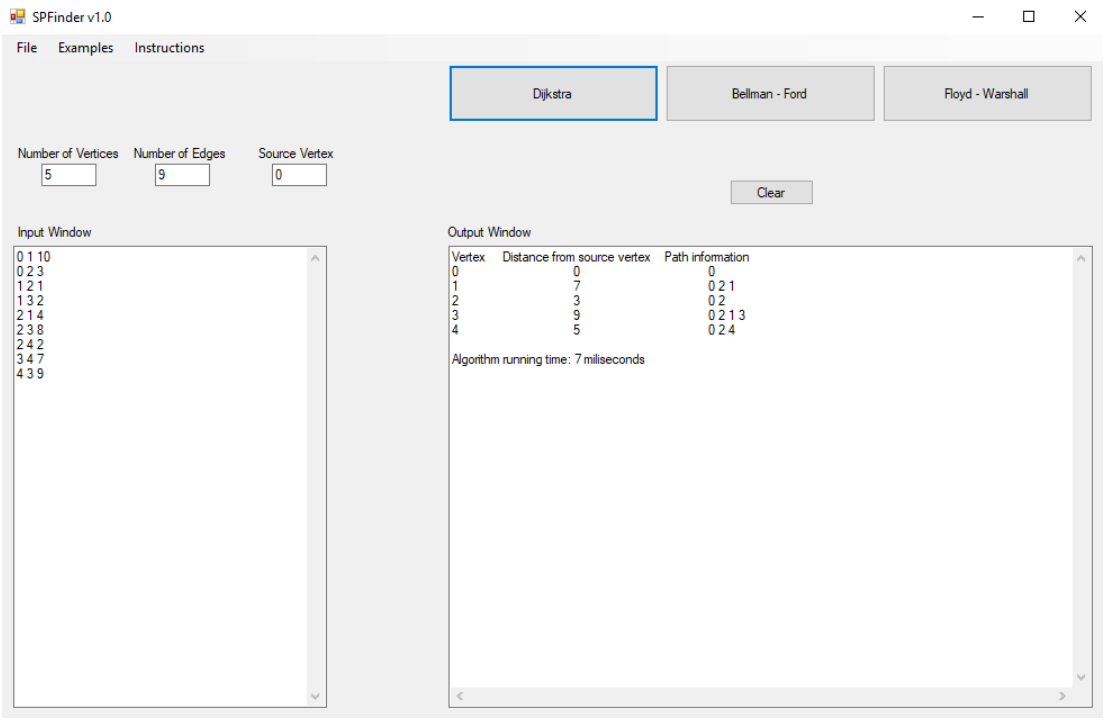


Sl 4.2. Prikaz mogućnosti unosa podataka u Input Window

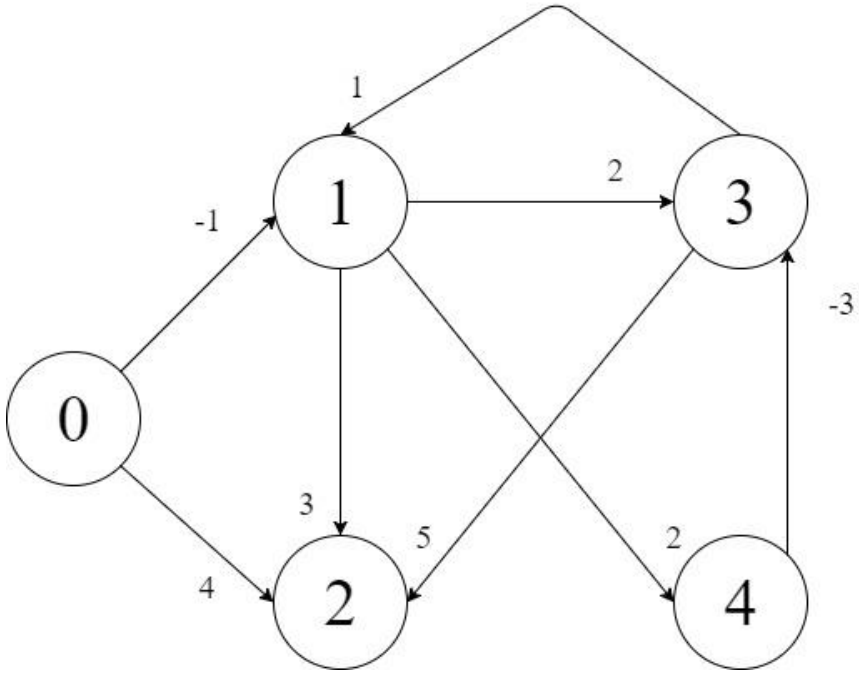
Nakon što su ulazni podatci pravilno oblikovani, korisnik odabire prikladan algoritam za traženje najkraćeg puta u grafu pritiskom na jedan od tri gumba smještenih u gornjem desnom dijelu sučelja. Program zatim izvršava odabrani algoritam temeljen na pseudokodu te rezultate prikazuje u *Output Window*-u. Rezultati se sastoje od podataka relevantnih za određeni algoritam, odnosno informacija o najkraćim putevima u mreži te vremena koje je potrebno da se određeni algoritam izvrši. Izlazne vrijednosti Dijkstrinog i Bellman-Fordovog algoritma su oblikovane na način da prikazuju sve čvorove u mreži, udaljenosti tih čvorova od ranije zadanog početnog čvora i informacije o samom putu do odredišnog čvora, dok su rezultati Floyd-Warshallovog algoritma prikazani u matričnom obliku. Ukoliko je neki od čvorova nedostupan iz odabranog početnog čvora, prikazat će se simbol *INF*. U izlaznoj matrici Floyd-Warshallovog algoritma svaka vrijednost predstavlja najkraću udaljenost između čvorova koji određuju položaj te vrijednosti u matrici (npr. vrijednost na poziciji [0,2] predstavlja najkraću udaljenost između čvorova 0 i 2, odnosno, slijedom pravila o označavanju čvorova u ovom programu, prvog i trećeg čvora na grafu). Izlazni podatci Floyd-Warshallovog algoritma ne sadrže informacije o pojedinim putevima između čvorova, zbog činjenice da čak i sa relativno malim brojem čvorova, primjerice 20, matrica će imati 400 vrijednosti najkraćih puteva

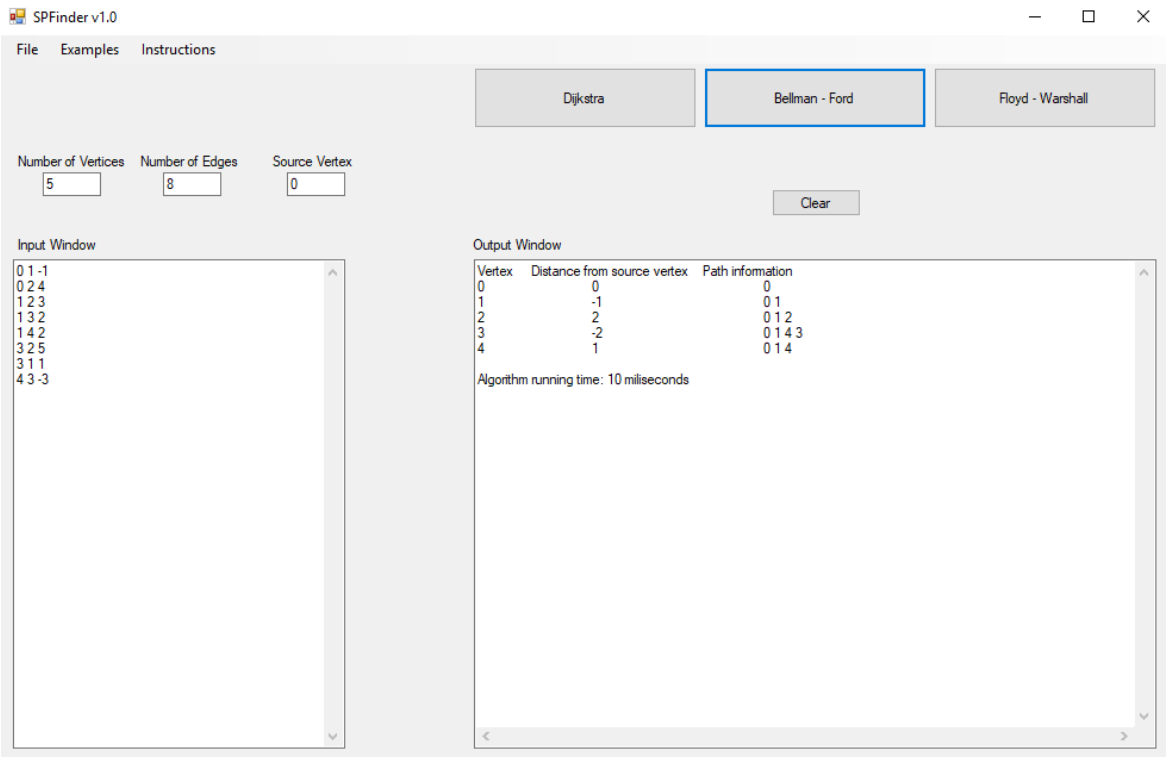
(kako je ranije navedeno, izlazna matrica je oblika *v* x *v*, gdje *v* broj čvorova). Te podatke zbog toga ne bi bilo moguće prikazati u obliku koji bi korisniku bio pregledan.



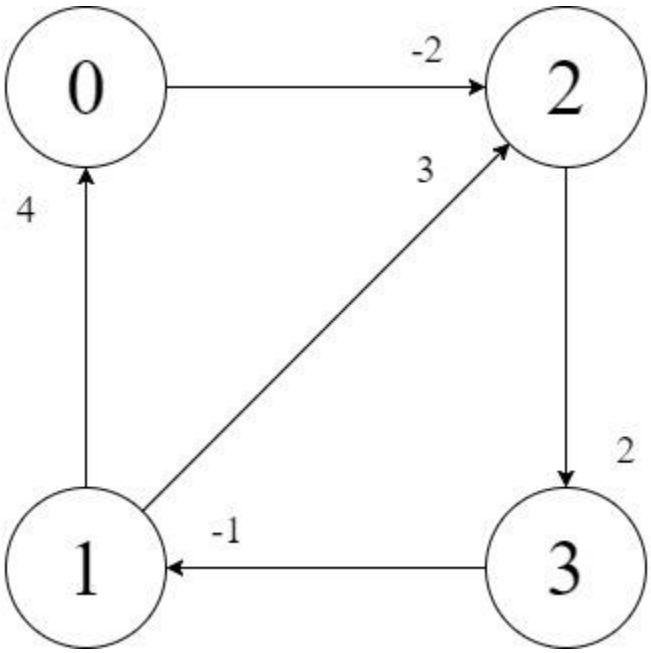


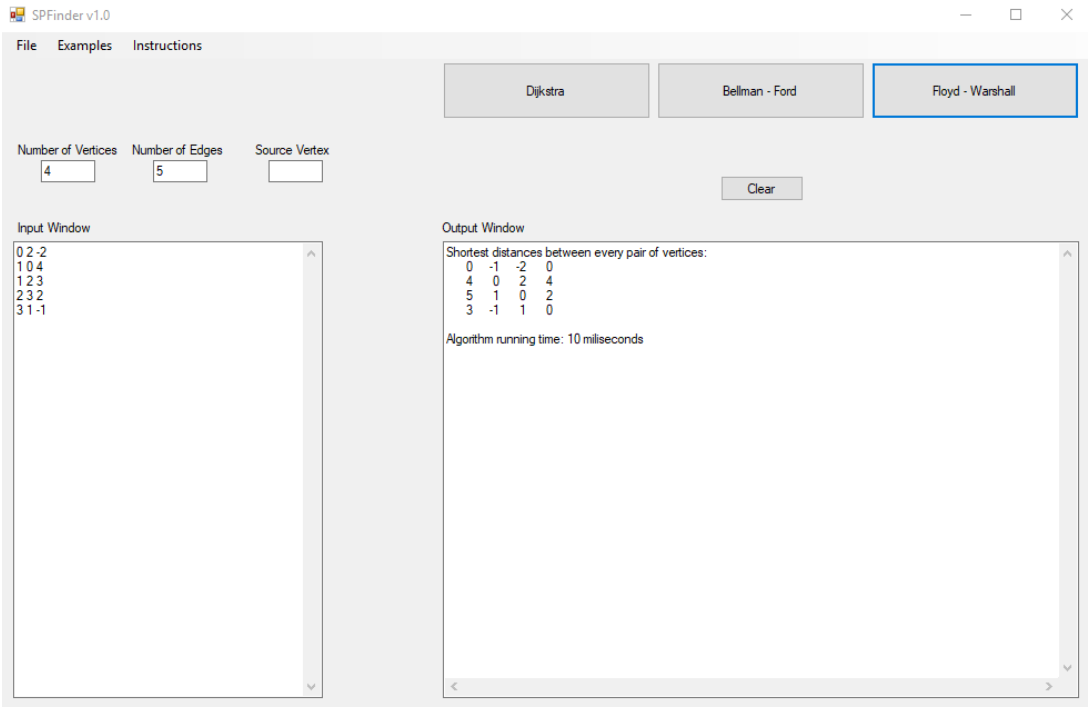
Sl 4.3. Prikaz ulaznih i izlaznih vrijednosti u Dijkstrinom algoritmu za zadani graf





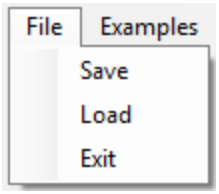
Sl 4.4. Prikaz ulaznih i izlaznih vrijednosti u Bellman-Fordovom algoritmu za zadani graf





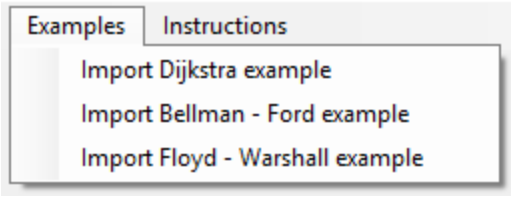
Sl 4.5. Prikaz ulaznih i izlaznih vrijednosti u Floyd-Warshallovom algoritmu za zadani graf

Nakon što je algoritam izvršen i izlazne vrijednosti prikazane, korisnik ima mogućnost spremiti sadržaj izlaznog prozora u datoteku *.txt* formata. To se izvršava klikom na *File* u traci izbornika, te zatim klikom na *Save* opciju. Nakon toga otvara se dijaloški okvir u kojem korisnik može odabrati lokaciju na koju želi spremiti datoteku s izlaznim podatcima, te imenovati samu datoteku. Na istom mjestu u programu nalazi se ranije navedena opcija za uvoz podataka iz datoteke, te opcija *Exit* kojom korisnik izlazi iz samog programa.



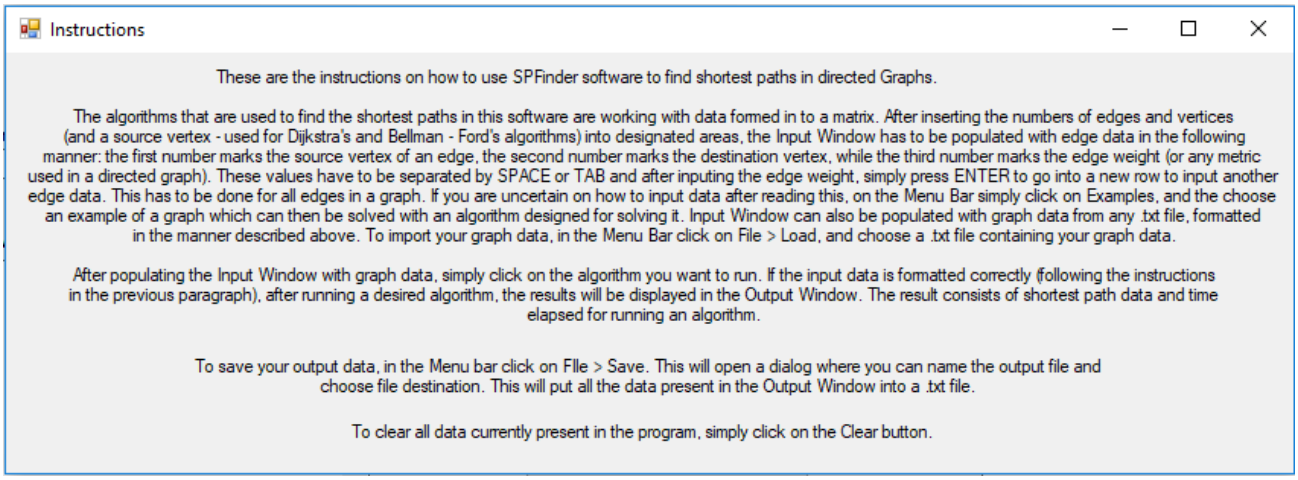
Sl 4.6. Prikaz opcija u File grupi trake izbornika

U programu se također nalaze i predefinirani primjeri za rješavanje određenim algoritmom koje korisnik može odabrati u svrhu vizualne demonstracije formatiranja ulaznih podataka. Ti primjeri su dostupni u traci izbornika, odabirom opcije *Examples*, gdje je zatim moguće odabrati jedan od primjera za svaki algoritam koji ovaj program može izvršiti.



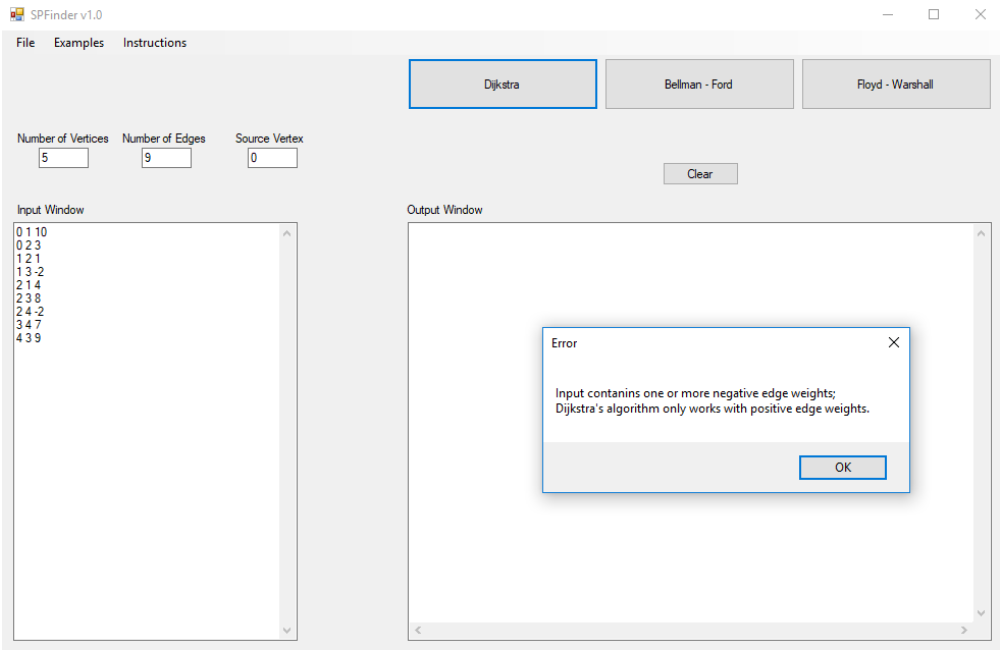
Sl 4.7. Prikaz opcija u Examples grupi trake izbornika

Sve ranije navedene upute za korištenje dostupne su i unutar samog programa. Upute su na engleskom jeziku, te su dostupne odabirom opcije *Instructions* na traci izbornika.



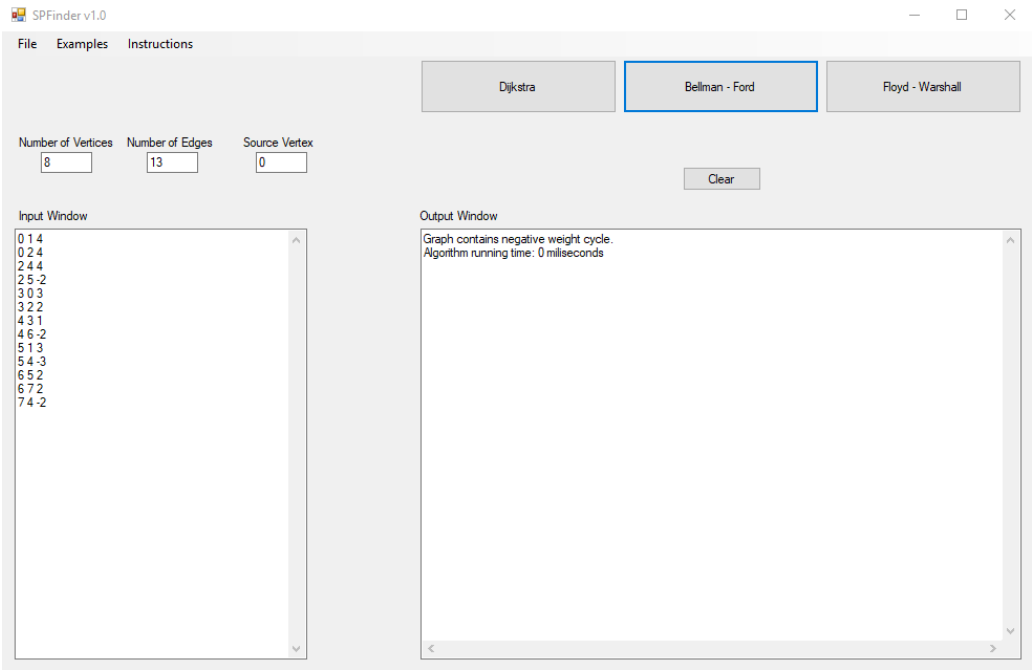
Sl 4.8. Prikaz uputa za korištenje unutar programa

Ukoliko korisnik pokuša izvršiti Dijkstrin algoritam na grafu na kojem postoje negativne težinske vrijednosti grana, prikazat će se prozor pogreške koji će korisnika upozoriti da u ulaznim podatcima postoje negativne vrijednosti grana.



Sl 4.9. Prikaz prozora pogreške kod negativnih vrijednosti u Dijkstrinom algoritmu

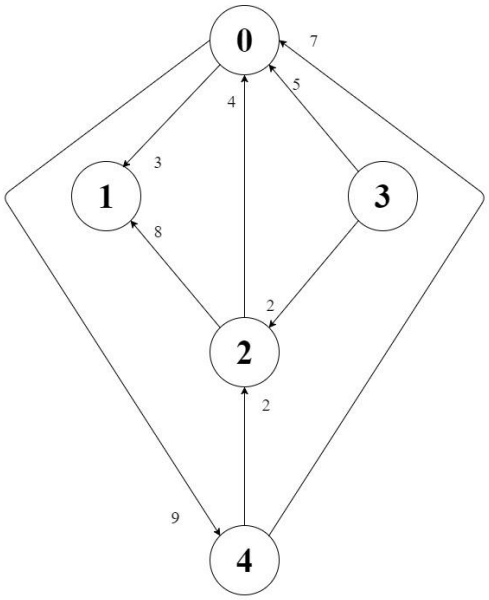
Isto tako, ukoliko pri izvršavanju Bellman-Fordovog algoritma dođe do detekcije negativnog ciklusa, takav rezultat će se prikazati u izlaznom prozoru te će algoritam prestati sa izvršavanjem.



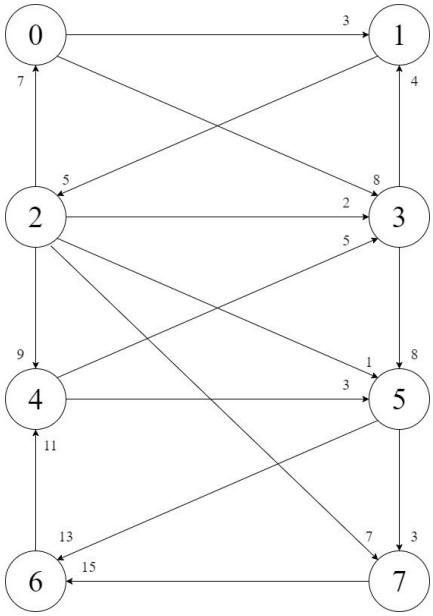
Sl 4.10. Prikaz detekcije prisutnosti negativnog ciklusa kod Bellman- Fordovog algoritma

4.3 Vremenska analiza algoritama

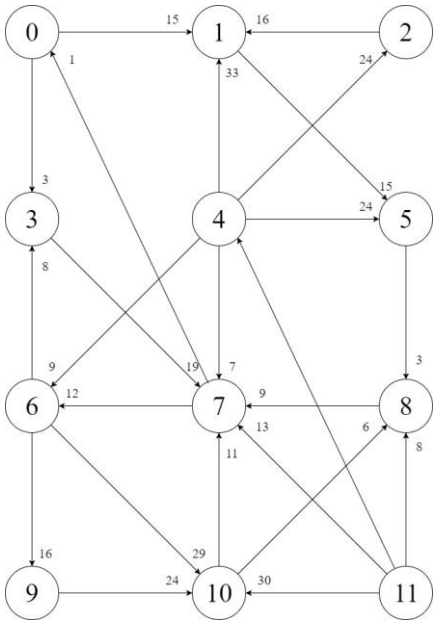
Kako bi se analiziralo vrijeme potrebno svakom od algoritama da pronađe vrijednosti najkraćih puteva, kreirana su tri usmjerena grafa, različitih brojeva čvorova i grana, na kojima će se izvršiti svaki od algoritama. Isto tako, analizirati će se ovisnost vremena trajanja algoritma o broju čvorova i grana u grafu. Prvi graf se sastoji od 5 čvora i 8 grana, drugi graf od 8 čvorova i 16 grana, a treći graf od 12 čvorova i 24 grane.



Sl 4.11. Testni graf 1: 5 čvorova i 8 grana

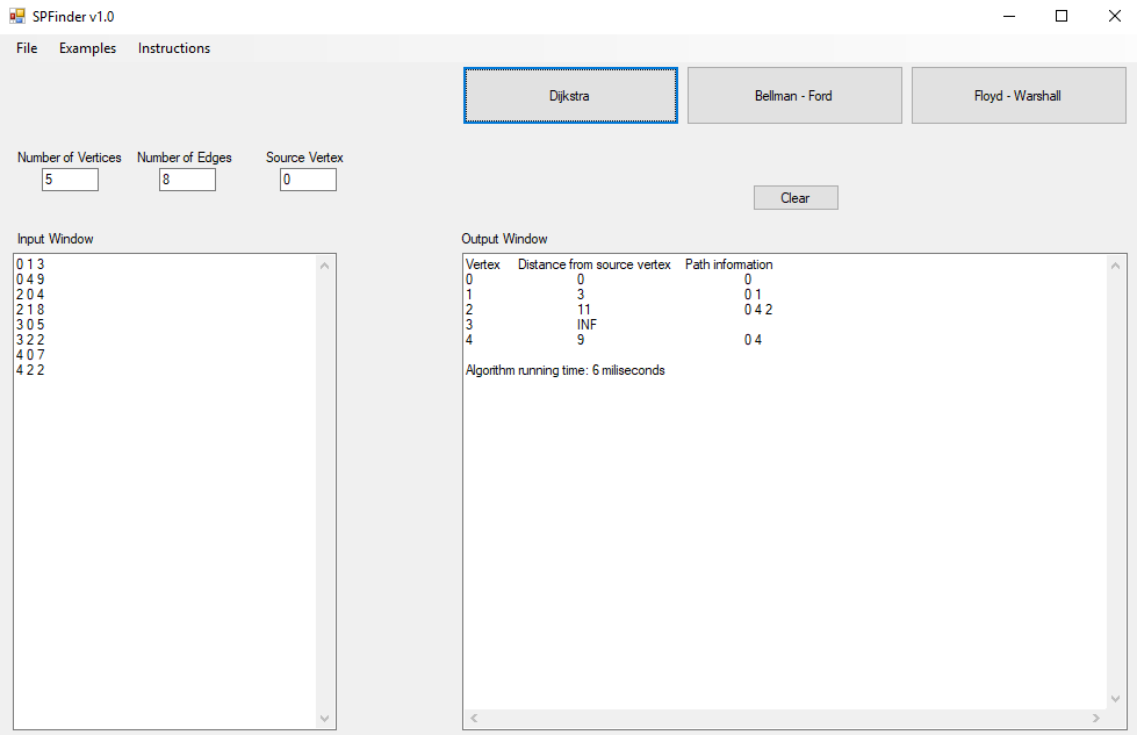


Sl 4.12. Testni graf 2: 8 čvorova i 16 grana

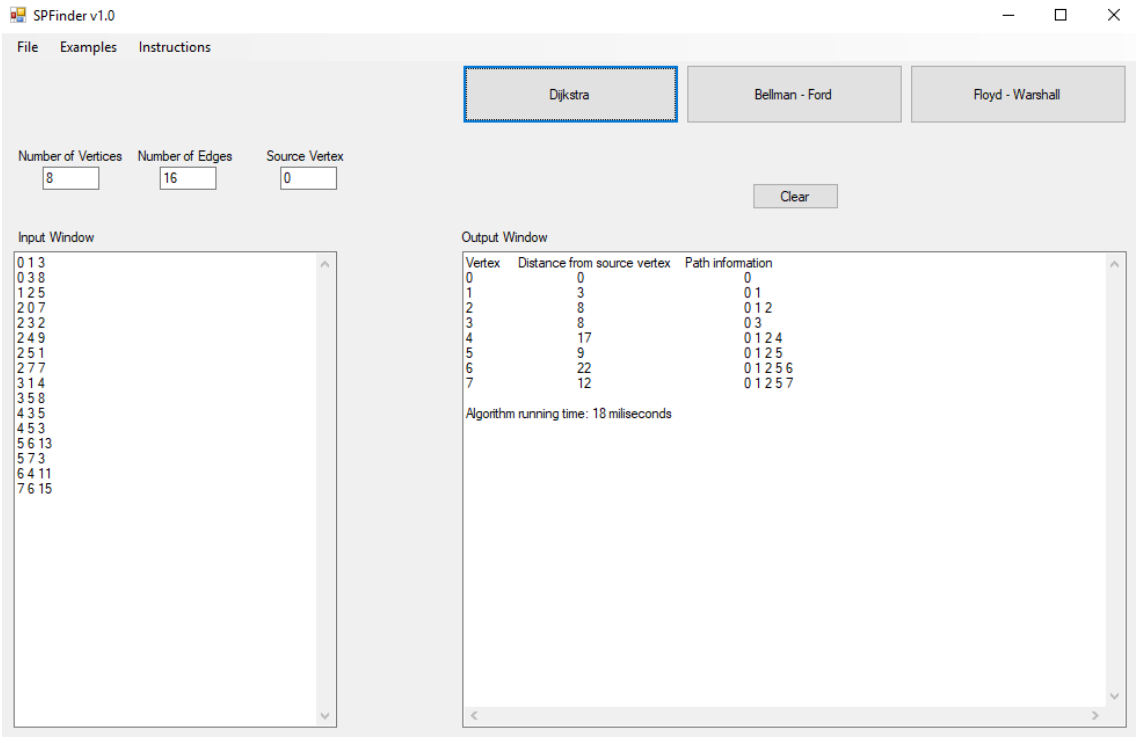


Slika 4.13. Testni graf 3: 12 čvorova i 24 grane

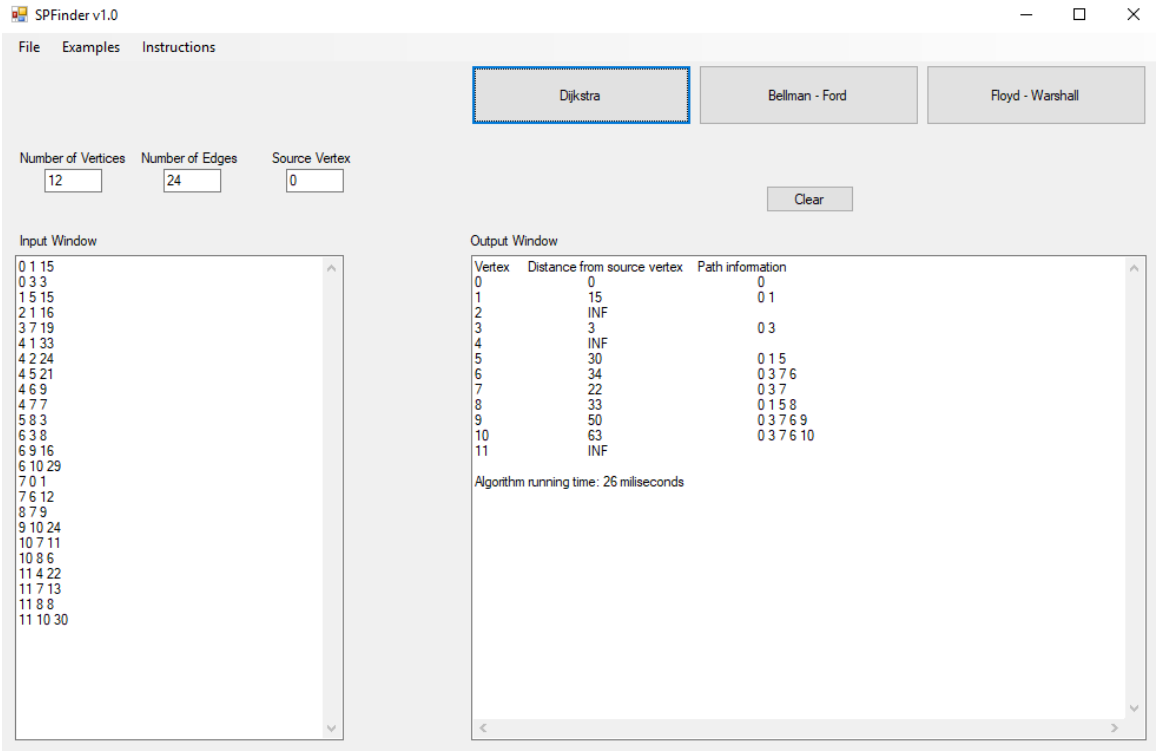
Prethodno prikazani grafovi imaju sve pozitivne težinske vrijednosti grana zbog činjenice da Dijkstrin algoritam ne radi sa negativnim vrijednostima. Ukoliko bi jedan od grafova imao negativnu težinsku vrijednost grane, Dijkstrin algoritam ne bi mogao biti primjenjen, te bi provedena vremenska analiza bila nepotpuna. Isto tako, ukoliko bi neke vrijednosti bile negativne, postoji mogućnost postojanja negativnog ciklusa kod Bellman-Fordovog algoritma, te sam rezultat algoritma ne bi bio pouzdan. Kao izvorišni čvor u Dijkstrinom i Bellman-Fordovom algoritmu odabran je čvor 0 u svakom od grafova. Podatci o grafovima su prikladno oblikovani i unešeni u SPFinder, te su dobiveni sljedeći rezultati:



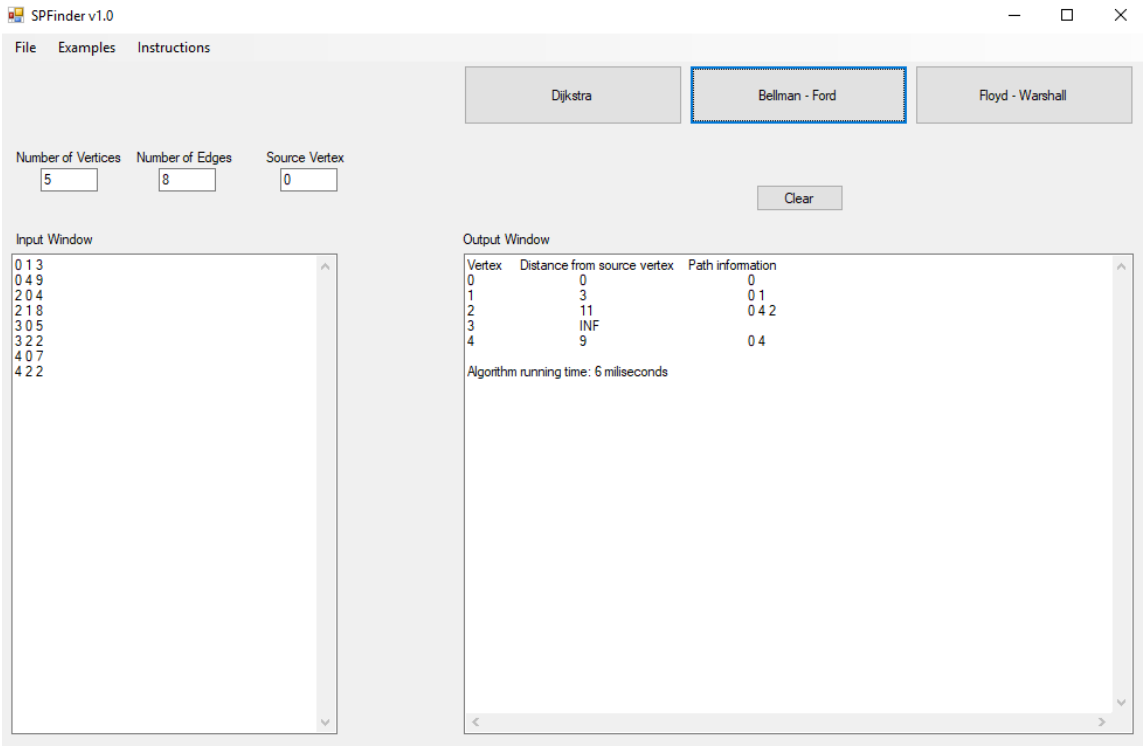
Slika 4.14. Rezultati Dijkstrinog algoritma za testni graf 1



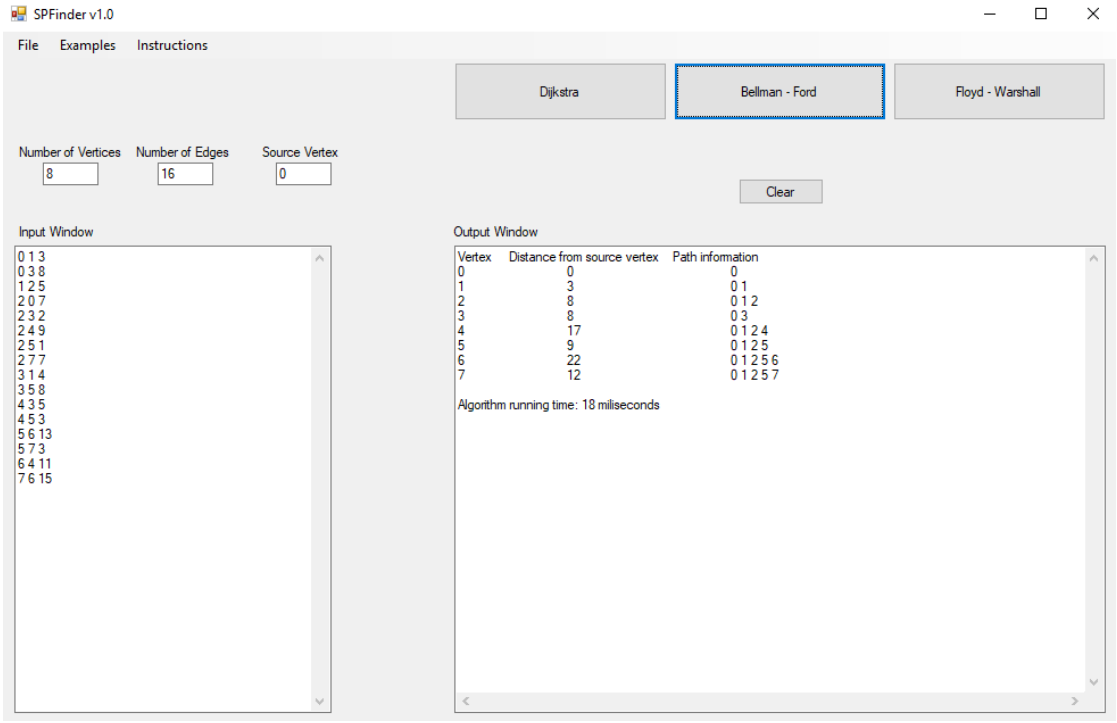
Slika 4.15. Rezultati Dijkstrinog algoritma za testni graf 2



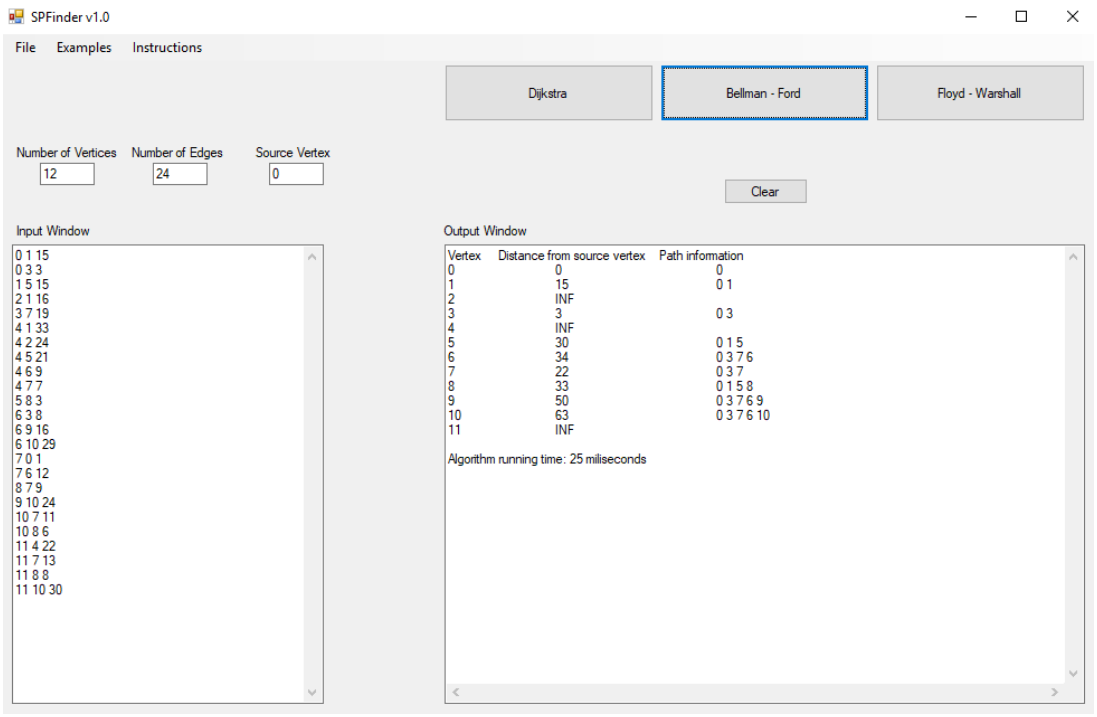
Slika 4.16. Rezultati Dijkstrinog algoritma za testni graf 3



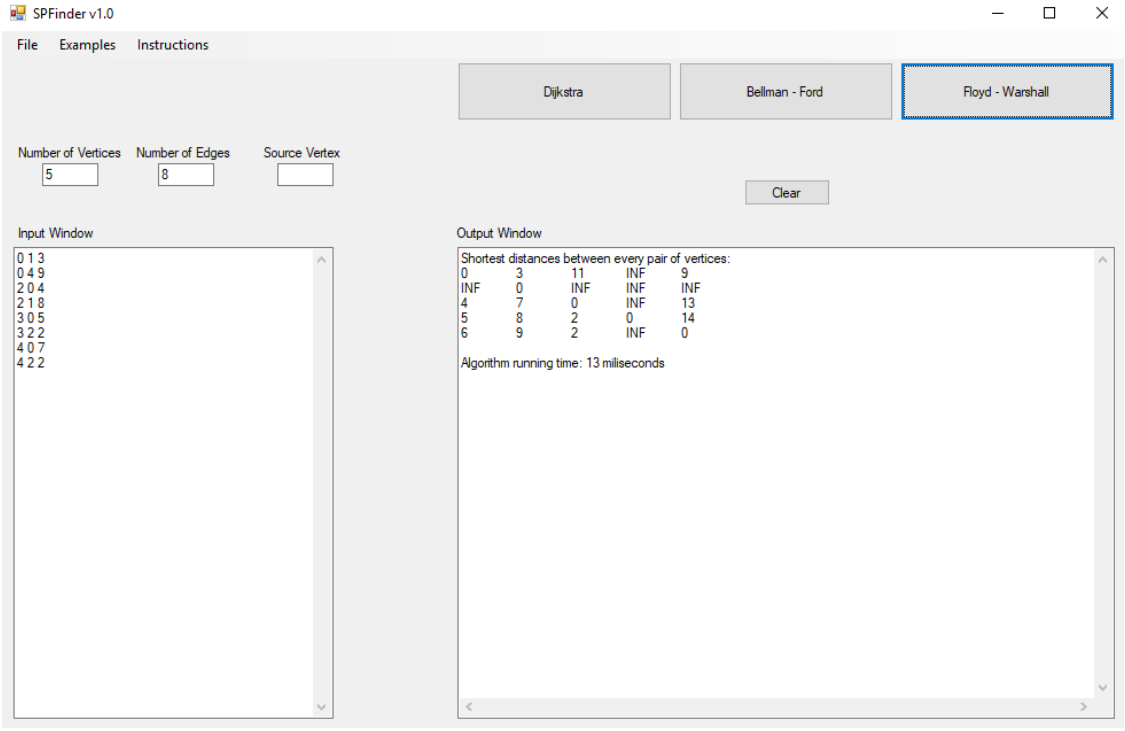
Slika 4.17. Rezultati Bellman-Fordovog algoritma za testni graf 1



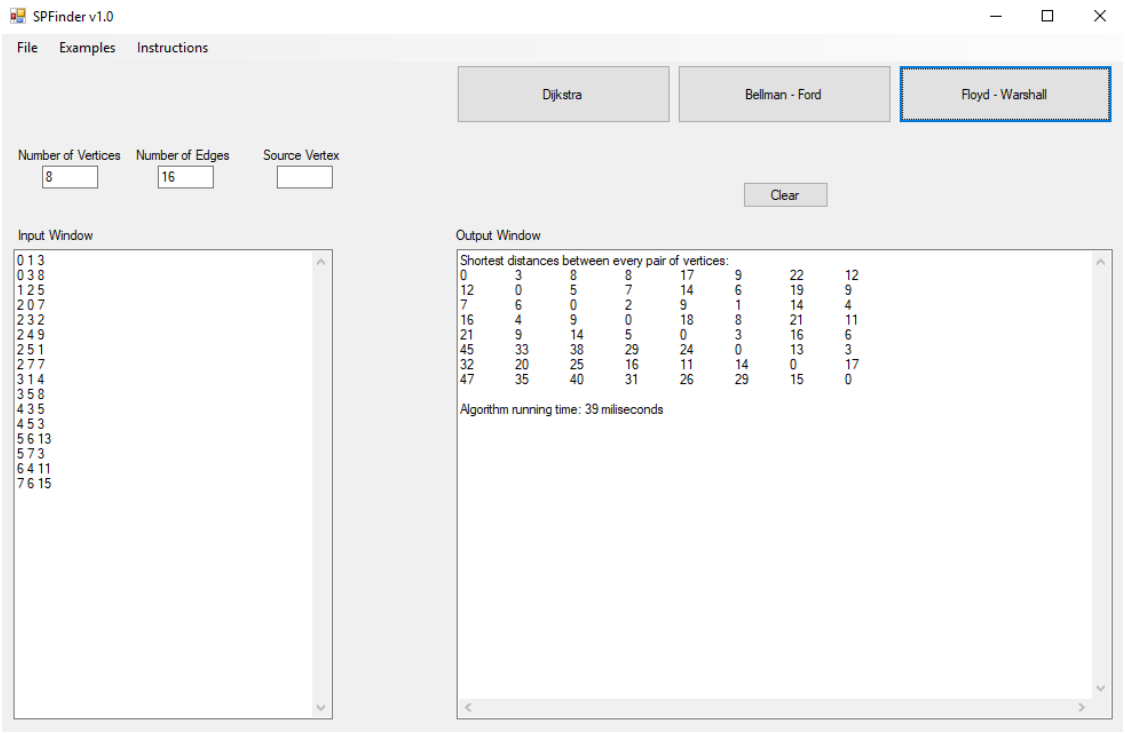
Slika 4.18. Rezultati Bellman-Fordovog algoritma za testni graf 2



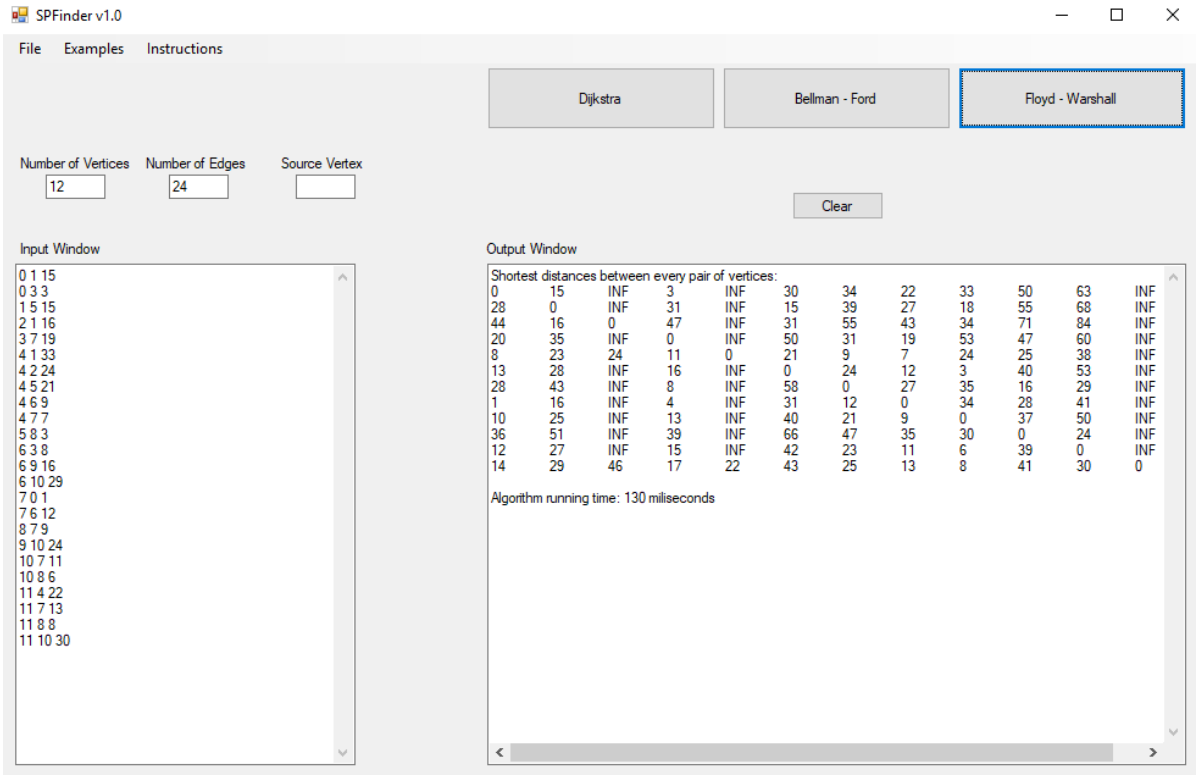
Slika 4.19. Rezultati Bellman-Fordovog algoritma za testni graf 3



Slika 4.20. Rezultati Floyd-Warshallovog algoritma za testni graf 1



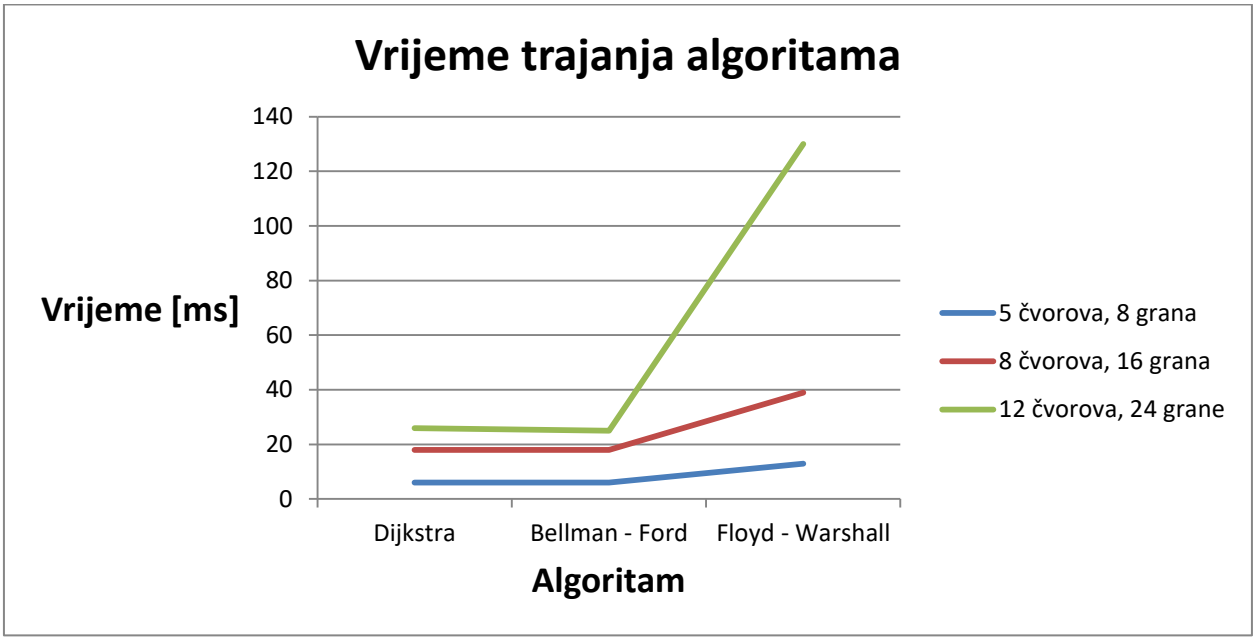
Slika 4.21. Rezultati Floyd-Warshallovog algoritma za testni graf 2



Slika 4.22. Rezultati Floyd-Warshallovog algoritma za testni graf 3

	Dijkstra	Bellman-Ford	Floyd-Warshall
Testni graf 1 (5 čvorova, 8 grana)	6 ms	6 ms	13 ms
Tesntni graf 2 (8 čvorova, 16 grana)	18 ms	18 ms	39 ms
Testni graf 3 (12 čvorova, 24 grana)	26 ms	25 ms	130 ms

Tab 4.1. Tablica ovisnosti vremena trajanja algoritma o veličini grafa



Sl 4.23. Graf ovisnosti vremena trajanja algoritma o veličini grafa

Iz rezultata vremenske analize algoritama možemo zaključiti da Dijkstrin i Bellman-Fordov algoritam imaju slično vremensko trajanje za zadane testne grafove. Razlog tome je što u testnim grafovima nema negativnih težinskih vrijednosti grafa, te se provjera negativnih ciklusa u grafovima preskače. Isto tako, razlog sličnom vremenskom trajanju ta dva algoritma je u tome da će Bellman-Fordov algoritam imati samo jednu iteraciju, jer zbog svih pozitivnih težinskih vrijednosti grana u grafu nema potrebne za daljnjim ažuriranjem vrijednosti najkraćeg puta koje bi se provodilo u slučaju postojanja negativnih težinskih vrijednosti grana na grafovima. Vrijeme trajanja Floyd-Warshallovog algoritma drastično povećava sa povećanjem broja čvorova grafa. Razlog tome je što algoritam, kako je ranije navedeno, kao izlaznu strukturu podataka ima matricu vrijednosti koja je dimenzija $v \times v$, gdje je v broj čvorova u grafu. Isto tako, svi ovi rezultati ovise o samoj procesorskoj moći računala na kojem se izvode. Dijkstrin algoritam je pogodan za pronalazak najkraćeg puta u mreži sa manjim brojem grana pozitivnih težinskih vrijednosti, dok je za one grafove sa manjim brojem grana, ali sa negativnim vrijednostima (bez negativnih ciklusa) pogodan Bellman-Fordov algoritam. Floyd-Warshallov algoritam je pogodan za traženje najkraćih puteva u onim grafovima koji imaju velik broj grana, odnosno na grafovima svaki čvor ima neposredan pristup gotovo svakom čvoru u mreži. To je zato što jezgru Floyd-Warshallovog algoritma čine 3 ugnijeđžene *for* petlje, gdje vanjska petlja predstavlja „prethodnika“, odnosno bazu koji će sljedeće dvije unutarnje petlje koristiti kako bi ažurirale matricu vrijednosti najkraćih puteva. Svaka petlja se izvršava $v - 1$ puta, gdje je v broj čvorova u grafu. To znači da vremenska kompleksnost Floyd-Warshallovog algoritma raste sa trećom potencijom broja čvorova u grafu. Petlje se uvijek izvršavaju v^3 puta, no ukoliko su grane čvorova povezane velikim brojem grana (u idealnom slučaju, svaki čvor bi bio neposredno povezan sa svakim od ostalih čvorova u mreži), preskočiti će se velik broj koraka u kojem se ažuriraju vrijednosti u izlaznoj matrici. To za posljedicu ima uvelike smanjeno vrijeme trajanja samog algoritma.

5. ZAKLJUČAK

Zadatak ovog rada bio je proučiti, analizirati te softverski implementirati neke od algoritama za traženje najkraćeg puta u mreži: Dijkstrin, Bellman - Fordov te Floyd - Warshallov. Bilo je potrebno detaljno analizirati teorijsku pozadinu svakoga od algoritama kako bi se oni mogli što efikasnije implementirati u softver, odnosno kako bi sam kod algoritama bio optimiziran. Svaki od ovih algoritama ima različite uvjete u kojima se može izvoditi, te različito vrijeme izvođenja, stoga se kao takvi ne mogu upotrebljavati u svim situacijama. U konstrukciji programa bilo je potrebno pronaći način kako informacije o grafu pretvoriti u oblik prikladan za obradu zadanim algoritmima, što je u ovom slučaju izvedeno na način da se informacije o grafu pretvore u matricu podataka. Također je bilo potrebno pronaći metode kako bi se u softveru izbjegle pogreške pri unosu podataka, izvršavanju algoritma koji nije predviđen za obradu specifičnih ulaznih podataka, te anomalija unutar samih grafova. Nakon što je program kreiran, prikazani su i neki od primjera izvršavanja samih algoritama, te je izvedena vremenska analiza na testnim primjerima kako bi se prikazala vremenska ovisnost izvršavanja algoritma o veličini grafova.

6. LITERATURA

[1] F.B. Zahn: Shortest Path Algorithms: An evaluation using Real Road Networks, Texas, 1998.

[2] Mladen Kos: 6. predavanje: Mrežni algoritmi 1, Ak.god. 2013./2014. https://www.fer.unizg.hr/_download/repository/Predavanje_6-Mrezni_algoritmi.pdf (lipanj 2017.)

[3] T.H. Cormen, C.E. Leirserson, R.L. Rivest, C. Stein: Introduction to Algorithms, Massachusets, 2009.

[4]M. Yan: Dijkstra's Algorithm http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf (lipanj 2017.)

[5] http://www.egr.unlv.edu/~larmore/Courses/CSC269/pathing (lipanj 2017.)

[6] http://www.programming-algorithms.net/article/45708/Floyd-Warshall-algorithm (lipanj 2017.)

[7] https://www.programmingalgorithms.com/algorithm/bellman%E2%80%93ford-algorithm (rujan 2017.)

[8] https://www.programmingalgorithms.com/algorithm/floyd%E2%80%93warshall-algorithm (rujan 2017.)

[9] https://www.programmingalgorithms.com/algorithm/dijkstra's-algorithm (rujan 2017.)

7. SAŽETAK

Algoritmi za pronalaženje najkraćih puteva u mreži vrlo su bitni za brz i pouzdan prijenos podataka od početne to krajnje točke u mreži. Neki od tih algoritama su Dijkstrin, Bellman-Fordov i Floyd-Warshallov algoritam. Dijkstrin algoritam radi samo sa pozitivnim težinskim vrijednostima grana, dok Bellman-Fordov i Floyd-Warshallov algoritam mogu raditi i sa negativnim vrijednostima. Kod Bellman-Fordovog algoritma može doći do detekcije negativnog ciklusa, što znači da se vrijednost najkraćeg puta do određenog čvora može beskonačno smanjivati. Ukoliko negativan ciklus ne postoji, Dijkstrin i Bellman-Fordov algoritam imaju slično vrijeme izvođenja, dok Floyd-Warshallov algoritam ima najveće vrijeme izvođenja zbog prikaza izlaznih vrijednosti u obliku matrice te velikog broja mogućih međurješenja.

Ključne riječi: algoritam, graf, Dijkstra, Bellman-Ford, Floyd-Warshall

ABSTRACT

Shortest path algorithms are very important for fast and reliable data transportation from a starting to an ending point in a network. Some of these algorithms are Dijkstra's, Bellman-Ford's and Floyd-Warshall's algorithm. Dijkstra's algorithm works only with positive edge weights, whereas Bellman-Ford's and Floyd-Warshall's algorithms are able to work with negative edge weights aswell. Bellman-Ford's algorithm is capable of negative cycle detection, which means that the shortest path value to a specific vertex can be infinitely reduced. If there is no such cycle, Dijkstra's and Bellman-Ford's algorithms have a similar runtime, while Floyd-Warshall's algorithm has the greatest runtime of the three, because of the format of the output values being a data matrix and a large amount of possible intermediate solutions.

Keywords: algorithm, graph, Dijkstra, Bellman-Ford, Floyd-Warshall

8. ŽIVOTOPIS

Matej Maligec rođen je 12. studenoga 1993. godine u Osijeku. 2000. godine upisuje OŠ „Josipovac“ u Josipovcu. Tijekom svoga osnovnog školovanja svih osam razreda postizao je odličan uspjeh s prosjekom 5.0 uz uzorno vladanje, te je sudjelovao na mnogim natjecanjima u matematici i hrvatskom jeziku. Nakon završene osnovne škole, 2008. godine upisuje srednju Elektrotehničku i prometnu školu Osijek. Sve četiri godine prolazi s odličnim uspjehom i uzornim vladanjem te nastavlja sa sudjelovanjem na natjecanjima iz matematike i fizike. Kao jedan od najboljih učenika generacije, nakon završene srednje škole ostvaruje pravo na izravan upis na Elektrotehnički fakultet Osijek. 2015. godine uspješno završava preddiplomski studij elektrotehnike, smjer Komunikacije i informatika, stječe zvanje inženjera prvostupnika elektrotehnike, te se iste godine upisuje na diplomski studij elektrotehnike, smjer Komunikacije i informatika, podsmjer Mrežne tehnologije.

_____

Matej Maligec

9. PRILOG

Glavna forma programa:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;
namespace SPFinder
{
    public partial class formHome : Form
    {

        public formHome()
        {
            InitializeComponent();
        }

        //Inicijalizacija varijabli potrebnih za funkcije štoperice, potrebnih za vremensku
analizu algoritama.

        public int INF = 99999;
        public Stopwatch stopWatch = new Stopwatch();
        long elapsedTime;
        private void btnDijkstra_Click(object sender, EventArgs e)
        {

            /*Postavljanje vrijednosti unešenih na sučelju u prikladne varijable
             te inicijalizacija matrica koje će algoritmi obrađivati. Pošto
             logički operator ili ne radi sa stringovima, potrebno je napraviti
             4 if petlje koje provjeravaju jesu li upisani svi potrebni podatci.*/

            if(tbSource.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            if (tbVertices.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            if (tbEdges.Text == "")
            {
```

```
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            if (tbInput.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            int curPos = 0;
            int source = Convert.ToInt32(tbSource.Text);
            int vertexCount = Convert.ToInt32(tbVertices.Text);
            int edgeCount = Convert.ToInt32(tbEdges.Text);
            int[,] matrix = new int[3, edgeCount];
            int[,] dijkstraInput = new int[vertexCount, vertexCount];
            string[] input = tbInput.Text.Split(' ', '\t', '\n');
            Graph graph = CreateGraph(vertexCount, edgeCount);
            stopWatch.Start();

  /*Postavljanje vrijednosti iz Input Window-a u strukturu Graph, te zatim u matricu koju će
algoritam obrađivati. Varijabla curPos koristi se za pomicanje položaja u matrici, odnosno
svaki puta nakon što se jedna pozicija popuni nekom od vrijednosti iz Input Window-a,
položaj u matrici se pomiče i omogućava popunjavanje nove pozicije. Vrijednosti su redom:
izvorišni čvor, odredišni čvor, težinska vrijednost grane.*/

            for (int i = 0; i < 3 && curPos < input.Length; i++)
            {
                for (int j = 0; j < edgeCount && curPos < input.Length; j++)
                {
                    graph.edge[j].source = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].source;
                    curPos++;
                    graph.edge[j].destination = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].destination;
                    curPos++;
                    graph.edge[j].weight = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].weight;
                    curPos++;
                    dijkstraInput[graph.edge[j].source, graph.edge[j].destination] =
graph.edge[j].weight;
                }

            }

 /*Provjera negativnih vrijednosti prije izvršavanja Dijkstrinog algoritma. U slučaju
   prisutnosti negativnih vrijednosti, prikazuje se prozor greške.*/

            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < edgeCount; j++)
                {
                    if (matrix[i, j] < 0)
                    {
                        MessageBox.Show("Input contanins one or more negative edge
weights;\nDijkstra's algorithm only works with positive edge weights.", "Error");
                        return;
```

```
                    }
                }
            }

        /*Resetiranje varijable curPos, izvršavanje Dijkstrinog algoritma te prikaz vremena
         koje je bilo potrebno za izvršavanje algoritma. */

            curPos = 0;
            Dijkstra(dijkstraInput, source, vertexCount);
            stopWatch.Stop();
            elapsedTime = stopWatch.ElapsedMilliseconds;
            tbOutput.Text += "Algorithm running time: " + elapsedTime + " miliseconds" +
Environment.NewLine + Environment.NewLine;
            stopWatch.Reset();

        }

        //Struktura u kojoj se nalaze podatci o granama grafa.

        public struct Edge
        {
            public int source;
            public int destination;
            public int weight;
        }

        //Nadstruktura koja uz prethodnu strukturu, sadrži i broj čvorova i grana u grafu.

        public struct Graph
        {
            public int verticesCount;
            public int edgesCount;
            public Edge[] edge;
        }

        //Metoda koja u koju se spremaju vrijednosti iz Input Window-a te zatim u matricu.

        public static Graph CreateGraph(int verticesCount, int edgesCount)
        {
            Graph graph = new Graph();
            graph.verticesCount = verticesCount;
            graph.edgesCount = edgesCount;
            graph.edge = new Edge[graph.edgesCount];
            return graph;
        }

        /*Metoda za ispis rezultata Dijkstrinog i Bellman - Fordovog algoritma.
         Ispisuje udaljenost od izvorišnog čvora do svih čvorova u grafu, te
         informacije o najkraćim putevima*/

        public void Print (int[] distance, int count, int[] parent)
        {
            tbOutput.Text += "Vertex     Distance from source vertex    Path information";
            tbOutput.Text += Environment.NewLine;
            int src = 0;
            for ( int i = 0; i < count; i++)
            {
```

```
        if (distance[i] == 2147483647)
        {
            tbOutput.Text += i + "\t\t" + "INF";
            tbOutput.Text += Environment.NewLine;
        }
        else
        {
            tbOutput.Text += i + "\t\t" + distance[i] + "\t\t\t"+ src + " ";
            printPath(parent, i);
            tbOutput.Text += Environment.NewLine;
        }

    }
    tbOutput.Text += Environment.NewLine;
}

/*Metoda za ispis najkraćeg puta od izvorišnog do pojedinog čvora u grafu.
  Ukoliko je u pitanju izvorišni čvor, funkcija se prekida.*/

public void printPath(int[] parent, int j)
{
    if (parent[j] == -1)
        return;

    printPath(parent, parent[j]);
    tbOutput.Text += j +" ";
}

/*Bellman - Fordov algoritam baziran na pseudokodu. Sadrži prazan cjelobrojni
 niz koji reprezentira udaljenosti od izvorišnog čvora do bilo kojeg čvora
 u mreži. On se popunjava u ugniježđenoj petlji koja provjerava trenutnu
 udaljenost, te ukoliko je ona veća od nove udaljenosti dodavanjem težine
 grane, ažurira. Ako graf sadrži negativan ciklus, algoritam se prekida.*/

public void BellmanFord(Graph graph, int source)
{
    int verticesCount = graph.verticesCount;
    int edgesCount = graph.edgesCount;
    int[] distance = new int[verticesCount];
    int[] parent = new int[verticesCount];
    for (int i = 0; i < verticesCount; i++)
    {
        parent[i] = -1;
        distance[i] = int.MaxValue;
    }

    distance[source] = 0;
    for (int i = 1; i <= verticesCount - 1; ++i)
    {
        for (int j = 0; j < edgesCount; ++j)
        {
            int u = graph.edge[j].source;
            int v = graph.edge[j].destination;
            int weight = graph.edge[j].weight;
```

```
                    /*Ukoliko je trenutna vrijednost najkraćeg puta od izvorišnog čvora
                     različita od maksimalne vrijednosti koja predstavlja beskonačnost,
                     te manja od sume te udaljenosti s težinom grane, ažurira se iznos
                     najkraćeg puta.*/

                    if (distance[u] != int.MaxValue && distance[u] + weight < distance[v])
                    {
                        parent[v] = u;
                        distance[v] = distance[u] + weight;
                    }
                }
            }
            /*Metoda za provjeru negativnog ciklusa u Bellman-Fordovom algoritmu.
              Ukoliko u grafu postoji negativan ciklus, algoritam prekida s radom
              te se u izlaznom prozoru ispisuje poruka.*/

            for (int i = 0; i < edgesCount; ++i)
            {
                int u = graph.edge[i].source;
                int v = graph.edge[i].destination;
                int weight = graph.edge[i].weight;

                if (distance[u] != int.MaxValue && distance[u] + weight < distance[v])
                {
                    tbOutput.Text += "Graph contains negative weight cycle.";
                    tbOutput.Text += Environment.NewLine;
                    return;
                }
            }

            Print(distance, verticesCount, parent);
        }


            /*Postavljanje vrijednosti unešenih na sučelju u prikladne varijable
             te inicijalizacija matrica koje će algoritmi obrađivati. Pošto
             logički operator ili ne radi sa stringovima, potrebno je napraviti
             4 if petlje koje provjeravaju jesu li upisani svi potrebni podatci.*/

        private void btnBellmanFord_Click(object sender, EventArgs e)
        {
            if(tbSource.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            if (tbVertices.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            if (tbEdges.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
```

```
                return;
            }
            if (tbInput.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            int curPos = 0;
            int source = Convert.ToInt32(tbSource.Text);
            int vertexCount = Convert.ToInt32(tbVertices.Text);
            int edgeCount = Convert.ToInt32(tbEdges.Text);
            Graph graph = CreateGraph(vertexCount, edgeCount);
            int[,] matrix = new int[3, edgeCount];
            string[] input = tbInput.Text.Split(' ', '\n','\t');
            stopWatch.Start();
            for (int i = 0; i < 3 && curPos < input.Length; i++)
            {
                for (int j = 0; j < edgeCount && curPos < input.Length; j++)
                {
                    graph.edge[j].source = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].source;
                    curPos++;
                    graph.edge[j].destination = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].destination;
                    curPos++;
                    graph.edge[j].weight = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].weight;
                    curPos++;
                }
            }

            /*Resetiranje varijabli, izvršavanje Bellman - Fordovog algoritma
             te prikaz vremena potrebnog za izvršavanje algoritma.*/

            curPos = 0;
            BellmanFord(graph, source);
            stopWatch.Stop();
            elapsedTime = stopWatch.ElapsedMilliseconds;
            tbOutput.Text += "Algorithm running time: " + elapsedTime + " miliseconds" +
Environment.NewLine + Environment.NewLine;
            stopWatch.Reset();

        }
```

```
/*Metoda koja se koristi u Dijkstrinom algoritmu. Ona manipulira "tablicom
  posjećenosti čvorova" opisanoj u radu, odnosno nizom vrijednosti najkraćih puteva
  u grafu. Posljednji niz brojeva koji se nalaiz u nizu distance predstavlja
  vrijednosti najkraćih puteva od izvorišnog čvora do svih ostalih čvorova u grafu.*/

        public int MinimumDistance(int[] distance, bool [] shortestPathTreeSet, int
verticesCount)
        {
            int min = int.MaxValue;
            int minIndex = 0;

            for (int v = 0; v < verticesCount; ++v)
            {
                if (shortestPathTreeSet[v] == false && distance[v] <= min)
                {
                    min = distance[v];
                    minIndex = v;
                }
            }

            return minIndex;
        }

 /*Metoda koja izvodi dijkstrin algoritam. Koristi prethodnu metodu kako bi provjerila
trenutne vrijednosti najkraćih puteva u grafu, te na osnovu unesene matrice, u for petlji
ažurira podatke u nizu distance, koji se na kraju prosljeđuje u metodu za ispis.*/

        public void Dijkstra(int[,] graph, int source, int verticesCount)
        {
            int[] distance = new int[verticesCount];
            bool[] shortestPathTreeSet = new bool[verticesCount];
            int[] parent = new int [verticesCount];

            for (int i = 0; i < verticesCount; ++i)
            {
                parent[i] = -1;
                distance[i] = int.MaxValue;
                shortestPathTreeSet[i] = false;
            }

            distance[source] = 0;

            for (int count = 0; count < verticesCount - 1; ++count)
            {
                int u = MinimumDistance(distance, shortestPathTreeSet, verticesCount);
                shortestPathTreeSet[u] = true;
```

```
/*Uvjeti za ažuriranje vrijednosti: neposjećeni svi čvorovi, postoji grana između čvorova
čija vrijednost umanjuje trenutnu vrijednost najkraćeg puta.*/

                for (int v = 0; v < verticesCount; ++v)
                    if (!shortestPathTreeSet[v] && Convert.ToBoolean(graph[u, v]) &&
distance[u] != int.MaxValue && distance[u] + graph[u, v] < distance[v])
                    {
                        parent[v] = u;
                        distance[v] = distance[u] + graph[u, v];
                    }
            }

            Print(distance, verticesCount, parent);
        }

        /*Inicijalizacija potrebnih tipova podataka za Floyd - Warshallov algoritam,
         slična kao kod prethodna dvs algoritma, s iznimkom inicijalizacije
         završne matrice za ispis koja ima sve vrijednosti INF, koje se kasnije
         po potrebi ažuriraju u samom algoritmu*/

        private void btnFloydWarshall_Click(object sender, EventArgs e)
        {
            if (tbVertices.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            if (tbEdges.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            if (tbInput.Text == "")
            {
                MessageBox.Show("There are one or more empty fields that require data.",
"Error");
                return;
            }
            int curPos = 0;
            int INF = 99999;
            int vertexCount = Convert.ToInt32(tbVertices.Text);
            int edgeCount = Convert.ToInt32(tbEdges.Text);
            int[,] matrix = new int[3, edgeCount];
            int[,] floydWarshallInput = new int[vertexCount, vertexCount];
            stopWatch.Start();
            for (int i = 0; i < vertexCount; i++)
            {
                for (int j = 0; j < vertexCount; j++)
                {
                    floydWarshallInput[i, j] = INF;
                }
            }
            string[] input = tbInput.Text.Split(' ', '\n', '\t');
            Graph graph = CreateGraph(vertexCount, edgeCount);
            for (int i = 0; i < 3 && curPos < input.Length; i++)
```

```
            {
                for (int j = 0; j < edgeCount && curPos < input.Length; j++)
                {
                    graph.edge[j].source = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].source;
                    curPos++;
                    graph.edge[j].destination = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].destination;
                    curPos++;
                    graph.edge[j].weight = Convert.ToInt32(input[curPos]);
                    matrix[i, j] = graph.edge[j].weight;
                    curPos++;
                    floydWarshallInput[graph.edge[j].source, graph.edge[j].destination] =
graph.edge[j].weight;
                }

            }

            /*Resetiranje varijabli, izvođenje Floyd - Warshallovog algoritma te prikaz
             vremena potrebnog za izvršavanje*/

            curPos = 0;
            FloydWarshall(floydWarshallInput, vertexCount);
            stopWatch.Stop();
            elapsedTime = stopWatch.ElapsedMilliseconds;
            tbOutput.Text += "Algorithm running time: " + elapsedTime + " miliseconds" +
Environment.NewLine + Environment.NewLine;
            stopWatch.Reset();
        }

        /*Metoda za ispis Floyd - Warshallovog algoritma.*/

        public void PrintFloydWarshall(int[,] distance, int verticesCount)
        {
            tbOutput.Text += "Shortest distances between every pair of vertices:";
            tbOutput.Text += Environment.NewLine;
            for (int i = 0; i < verticesCount; ++i)
            {
                for (int j = 0; j < verticesCount; ++j)
                {
                    if (distance[i, j] == INF)
                        tbOutput.Text += "INF\t";
                    else
                        tbOutput.Text += distance[i, j].ToString() + "\t";
                }
                tbOutput.Text += Environment.NewLine;
            }
            tbOutput.Text += Environment.NewLine;

        }
```

```
/*Metoda za izvršavanje Floyd - Warshallovog algoritma. Sastoji se od trostruke petlje
  koja služi za ažuriranje vrijednosti najkraćih puteva. Treća, odnosno vanjska petlja
  definirana varijablom k predstavlja prethodnika na koji se unutarnje petlje definirane
  varijablama i i k referiraju kako bi, ukoliko je potrebno, ažurirali vrijednosti u
  matrici najkraćih puteva*/

        public void FloydWarshall(int[,] graph, int verticesCount)
        {
            int[,] distance = new int[verticesCount, verticesCount];

            for (int i = 0; i < verticesCount; ++i)
                for (int j = 0; j < verticesCount; ++j)
                    distance[i, j] = graph[i, j];
            for (int k = 0; k < verticesCount; ++k)
            {
                for (int i = 0; i < verticesCount; ++i)
                {
                    for (int j = 0; j < verticesCount; ++j)
                    {
                        if (distance[i, k] + distance[k, j] < distance[i, j])
                            distance[i, j] = distance[i, k] + distance[k, j];
                    }
                }
            }

            /*Postavljanje vrijednosti na glavnoj dijagonali matrice u nulu. Najkraći put od
             jednog čvora to tog istog čvora je nula.*/
            for (int i = 0; i < verticesCount; ++i)
            {
                distance[i, i] = 0;
            }
            PrintFloydWarshall(distance, verticesCount);
        }

        //Inicijalizacija dijaloga za spremanje podataka u tekstualnu datoteku.

        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                    {
                        File.WriteAllText(saveFileDialog1.FileName, tbOutput.Text);
                    }
        }

        //Inicijalizacija dijaloga za unos podataka iz tekstualne datoteke.

        private void loadToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                tbInput.Text = File.ReadAllText(openFileDialog1.FileName);
            }

        }
```

```
        //Postavljanje primjera ulaznih podataka za Dijkstrin algoritam

         private void importDijkstraExampleToolStripMenuItem_Click(object sender, EventArgs
e)
         {
             tbVertices.Text = "5";
             tbEdges.Text = "9";
             tbSource.Text = "0";
             tbInput.Text = "0 1 10\r\n0 2 3\r\n1 2 1\r\n1 3 2\r\n2 1 4\r\n2 3 8\r\n2 4
2\r\n3 4 7\r\n4 3 9";
         }

         //Postavljanje primjera ulaznih podataka za Floyd - Warshallov algoritam.

         private void ImportFloydWarshallExampleToolStripMenuItem_Click(object sender,
EventArgs e)
         {
             tbVertices.Text = "4";
             tbEdges.Text = "5";
             tbInput.Text = "0 2 -2\r\n1 0 4\r\n1 2 3\r\n2 3 2\r\n3 1 -1";
         }

         //Postavljanje primjera ulaznih podataka za Bellman - Fordov algoritam.

         private void importBellmanFordExampleToolStripMenuItem_Click(object sender,
EventArgs e)
         {
             tbVertices.Text = "5";
             tbEdges.Text = "8";
             tbSource.Text = "0";
             tbInput.Text = "0 1 -1\r\n0 2 4\r\n1 2 3\r\n1 3 2\r\n1 4 2\r\n3 2 5\r\n3 1
1\r\n4 3 -3";
         }

         //Metoda za izlaz iz programa

         private void exitToolStripMenuItem_Click(object sender, EventArgs e)
         {
             this.Close();
         }

         //Metoda za čišćenje podatkovnih prozora u programu.

         private void btnClear_Click(object sender, EventArgs e)
         {
             tbVertices.Text = "";
             tbEdges.Text = "";
             tbSource.Text = "";
             tbInput.Text = "";
             tbOutput.Text = "";
         }
```

```
//Metoda za prikaz uputa za korištenje.

        private void instructionsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Form2 frmInstructions = new Form2();
            frmInstructions.Show();
        }
    }
}
```

Forma za prikaz uputa za korištenje:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SPFinder
{
    public partial class Form2 : Form
    {

        //Prozor kojemu je naslov Instructions te prikazuje upute za korištenje.

        public Form2()
        {
            InitializeComponent();
            this.Text = "Instructions";
        }
    }


}
```