Elektrotehnički fakultet
Univerzitet u Sarajevu
Odsjek za AiE/Odsjek za RI
Zimski semestar, a.g. 2018/2019.
Predmet: Multimedijalni sistemi

Vizualizacija Bellman-Ford algoritma

Sarajevo, Januar 2019.

Studenti: Lamija Hasanagić(1551/17544)
Tin Vidović (1544/17545)
Edna Fazlagić (1531/17554)
Mirza Hodović(1493/17493)

1. Definicija zadatka

Ovaj rad opisuje implementaciju vizualizacije Bellman-Fordovog algoritma kao algoritma za pronalaženje najkraćeg puta u grafu. Zadatak se sastoji od implementacije Bellman-Ford algoritma u nekom programskom jeziku, u ovom slučaju, Python, kao i vizualizacije rada algoritma kroz grafovski kao i tabelarni prikaz. Grafovski prikaz služi za vizualizaciju izbora grana pri radu algoritma, dok tabelarni prikaz za cilj ima vizualizirati promjene težinskih varijabli algoritma koje isti koristi za pronalazak najkraćeg puta između čvorova u grafu. Vizualizacija rada algoritma se izvodi sa ciljem lakše analize te učenja principa na kojima se rad algoritma zasniva.

2. IZJAVA O DOPRINOSU ČLANOVA TIMA PRI IZRADI SEMINARSKOG RADA IZ PREDMETA MMS

U tabeli ispod navesti imena svih članova tima, te aktivnosti u kojima je učestvovao svaki pojedini član tima. Pri navođenju aktivnosti za svakog člana tima koristite bilo koju kombinaciju sljedećih opcija:

- Analiza problema seminarskog rada;
- Osmišljavanje rješenja;
- Praktična implementacija seminarskog rada;
- Pisanje teksta seminarskog rada;
- Ostalo (navesti koje su to ostale eventualne aktivnosti).

Prezime i ime člana tima	**Aktivnosti**
Fazlagić Edna	- Analiza problema seminarskog rada - Osmišljavanje rješenja - Praktična implementacija seminarskog rada (tabelarni prikaz) - Pisanje teksta seminarskog rada
Hasanagić Lamija	- Analiza problema seminarskog rada - Osmišljavanje rješenja - Praktična implementacija seminarskog rada (implementacija algoritma u Python-u) - Pisanje teksta seminarskog rada
Hodović Mirza	- Analiza problema seminarskog rada - Osmišljavanje rješenja - Praktična implementacija seminarskog rada (prikaz u formi grafa) - Pisanje teksta seminarskog rada
Vidović Tin	- Analiza problema seminarskog rada - Osmišljavanje rješenja - Praktična implementacija seminarskog rada (implementacija algoritma i Python-u) - Pisanje teksta seminarskog rada

3. Uvod

3.1. Elementi teorije grafova

Kako bi se mogao opisati način rada Bellman-Fordovog algoritma na ovom mjestu će biti definisani neki od osnovnih pojmova teorije grafova i to:

- Graf

Graf G predstavlja uređeni par G = (E, V), gdje je E neki neprazan skup, a V binarna relacija unutar skupa E.

- Čvor

Elementi skupa E se nazivaju čvorovima grafa G = (E, V).

- Grana

Elementi relacije V se nazivaju granama grafa G = (E, V).

Graf G = (E, V) u kojem je relacija R simetrična relacija naziva se simetrični odnosno neusmjereni graf dok se u suprotnom govori o nesimetričnom odnosno usmjerenom grafu.

- Put

Pod putem u nekom grafu G =(E, V) smatramo svaki niz grana (tj. elemenata iz V) $g_1,g_2,...,g_k$ pri čemu svaka grana g_i , i = 2,3,...,k počinje u onom čvoru u kojem završava grana g_{i-1}. Početni čvor grane g_1 i krajnji čvor grane g_k nazivaju se početak i kraj puta. Putevi mogu prolaziti više puta istom granom ili kroz isti čvor. [1]

3.2. Historija i značaj Bellman-Ford algoritma

Bellman-Fordov algoritam je algoritam koji pronalazi najkraći put od jednog čvora koji predstavlja izvor do svih ostalih čvorova u grafu. Jedna od najpoznatijih optimizacija Bellman-Ford algoritma je Dijkstrin algoritam, međutim bitno je naglasiti da taj algoritam ne radi u slučaju da neke grane u grafu imaju negativne težine. Bellman-Fordov algoritam pravilno pronalazi najkraće puteve i u ovom slučaju. [2]

Bellman-Fordov algoritam je prvi predložio Alfonso Shimbel 1955. Godine, ali je ime dobio po Richard Bellmanu i Lester Ford Jr. koji su objavili algoritam 1958., i 1956. godine respektivno.

Edward F. Moore je objavio isti algoritam 1957. Godine, pa se ovaj algoritam često naziva i Bellman-Ford-Moore-ov algoritam. [2]

Kao što je već spomenuto Bellman-Fordov algoritam pravilno pronalazi najkraće puteve i za one grafove koji sadrže grane negativnih težina, što je čest slučaj u praktičnoj primjeni. Također, ukoliko graf sadrži negativne cikluse, tada ne postoji najkraći put, jer se sa još jednim dodatnim prolaskom kroz ciklus negativne težine ukupna cijena puta smanjuje. U ovom slučaju, Bellman-Fordov algoritam je u stanju pronaći ovakve cikluse i upozoriti na njihovo prisustvo. [2]

3.3. Opis algoritma

Bellman-Fordov algoritam je u osnovi veoma jednostavan i zasniva se na ažuriranju pomoćnih veličina pridruženih čvorovima, koje se mogu označiti sa λ_i = 1, 2,…,n i čija je uloga prikazivanje težine najkraćeg puta od čvora koji predstavlja izvor do svih čvorova koji su od tog čvora udaljeni za k grana gdje je k broj iteracija algoritma. Dalje, ovaj algoritam pravilno radi i ukoliko se graf posmatra kao neusmjereni graf odnosno kada se svaka neusmjerena grana grafa posmatra kao par usmjerenih grana. [1]

U nastavku je ukratko opisan način rada Bellman-Fordovog algoritma. Početni čvor se može označiti sa x_1 . Na početku algoritma se postavlja $\lambda_1 = 0$, dok se svim ostalim čvorovima pridružuje $\lambda_i = \infty$, ii = 2, 3,…,n . Sada se redom, za svaku granu grafa oblika (x_i, x_j) računaju veličine $\lambda_i' = \lambda_i + w_{i,j}$, pri čemu je $w_{i,j}$ težina grane (x_i, x_j). Ukoliko je $\lambda_j' < \lambda_j$, korigira se vrijednost tako da postane $\lambda_j' = \lambda_j$. Pri tome se kaže da je korekcija izvršena u odnosu na čvor x_i. Ovim je završena prva iteracija algoritma. U narednim iteracijama ponavlja se isti postupak za svaku granu, uz ponovne eventualne korekcije veličina λ_i , i = 1,2,…,n . Zapravo, algoritam se može značajno ubrzati ukoliko se u narednim iteracijama ne razmatraju sve grane nego samo one grane koje izlaze iz onih čvorova x_i u odnosu na koje je u prethodnoj iteraciji došlo do promjene vrijednosti λ_i , s obzirom da samo take grane mogu dovesti do novih promjena. Postupak se završava onda kada se u nekoj od iteracija ne izvrši nijedna korekcija. Može se dokazati da broj iteracija ne može biti veči od n, a tipično je znatno manji. [1]

Sam najkraći put prema izvornom Bellman-Fordovom algoritmu određuje se na sljedeći način. Neka je krajnji čvor do kojeg se traži najkraći put x_k. Njegov prethodnik u traženom putu je onaj čvor x_p za koji vrijedi $w_{p,k} = \lambda_k - \lambda_p$. Sada se na isti način određuje i prethodnik čvora x_p dok se ne dođe do čvora koji je odabran kao izvor. [1]

Bitno je naglasiti da se Bellman-Fordov algoritam ne smije prekinuti prije nego što se završe sve iteracije, čak i ukoliko se traži najkraći put do nekog određenog čvora x_k. Naime, sve do okončanja postupka, ne postoji nikakva garancija da se vrijednost λ_k neće dalje mijenjati. [1]

3.4 Pregled dostupne literature

Bellman-Fordov algoritam je veoma značajan algoritam za pronalaženje najkraćeg puta od izvora do ostalih čvorova unutar grafa te je zbog ovoga kao i zbog osobine da pravilno zaključuje o najkraćim putevima čak i u grafovima sa negativnim težinama grana u širokoj upotrebi. S obzirom na široku upotrebu ovog algoritma postoje mnogi radovi na temu Bellman-Fordovog algoritma i njegovih varijacija kao u [3] i [1]. Različite implementacije vizualizacije Bellman-Fordovog algoritma su također dostupne kao u npr. [4] i [5].

4. Opis praktičnog dijela

Praktični dio ovog rada sastoji se od nekoliko dijelova i to:

- Implementacija Bellman-Fordovog algoritma
- Implementacija vizualizacije u tabelarnom obliku i
- Implementacija vizualizacije u grafovskom obliku

4.1 Implementacija Bellman-Fordovog algoritma

Implementacija algoritma se sastoji od implementacije klase Graf sa atributima koji predstavljaju skup cvorova i skup relacija između čvorova, odnosno grana grafa. Te su implentirane metode daLiJeNeusmjereniGraf koja vraća tip grafa te dodajGranu koja se koristi za dodavanje grana u grafu specificirajući polazni dolazni čvor. Njihova implementacija u python programskom jeziku je data u nastavku:

```
class Graf:
    def __init__(self, cvorovi, daLiJeNeusmjereni):
        self._V = cvorovi
        self._graf = []
        self.G = nx.Graph()
        self.valjda_valja = dict()
        self._daLiJeNeusmjereni = daLiJeNeusmjereni
        self.prviput = True
        self.pos = None

    def getDaLiJeNeusmjereni(self):
        return self._daLiJeNeusmjereni


    def dodajGranu(self, u, v, w):
        self._graf.append((u, v, w))
        self.G.add_edge(u, v, weight=w)
        self.valjda_valja[(u, v)] = w
        if self.getDaLiJeNeusmjereni():
            self._graf.append((v, u, w))
            self.valjda_valja[(v, u)] = w
```

Implementirane su i metode ispisiMinimalneUdaljenosti, ispisiPuteveDoCvorova, pronadjiPutDoCvora koje predstavljaju pomoćne metode za ispisivanje parametara λ_i koji su objašnjeni u 3. poglavlju. Njihova implementacija je data u nastavku:

```
def ispisiMinimalneUdaljenosti(self, udaljenosti):
        print("Cvor        Udaljenost od izvora")
        for i in range(self._V):
            if udaljenosti[i] == float("Inf"):
                print("%d\t\t\tinf" % (i))
            else:
                print("%d\t\t\t%d" % (i, udaljenosti[i]))
        return

    def ispisiPuteveDoCvorova(self, izvor, prethodnici):
        print("Cvor        Put do cvora")
        for i in range(self._V):
            if prethodnici[i] == float("Inf"):
                print("%d\t\tPut do cvora ne postoji!" % (i))
            else:
                print("%d\t\t" % (i), end = '')
                print(self.pronadjiPutDoCvora(i, izvor, prethodnici))
        return
```

Sam algoritam implementiran je unutar metode Bellman_Ford čija implementacija je data u nastavku: Algoritam je implementiran tako da radi samo sa onim granama koje izlaze iz onih čvorova x_i u odnosu na koje je u prethodnoj iteraciji došlo do promjene vrijednosti λ_i, u cilju optimizacije koja je opisana u 3.3.

```
def Bellman_Ford(self, izvor):

        print(len(self._graf))

        udaljenosti = []
        for i in range(self._V):
            udaljenosti.append(float("Inf"))
        udaljenosti[izvor] = 0

        prethodnici = []
        for i in range(self._V):
            prethodnici.append(float("Inf"))
        prethodnici[izvor] = izvor

        korigiraneUdaljenosti = self._graf

        self.ispisiMinimalneUdaljenosti(udaljenosti)

        for i in range(self._V-1):

            dosloDoKorekcije = False ## NIJE POTREBNO POSMATRATI SVE GRANE

            for u, v, w in korigiraneUdaljenosti:
```

```
                    if udaljenosti[u] != float("Inf") and udaljenosti[u] + w <
udaljenosti[v]:
                    udaljenosti[v] = udaljenosti[u] + w
                    prethodnici[v] = u
                    dosloDoKorekcije = True

                    korigiraneUdaljenosti.append((u, v, w))
                    if self.getDaLiJeNeusmjereni():
                        korigiraneUdaljenosti.append((v, u, w))

            if dosloDoKorekcije == False:
                break

            self.ispisiMinimalneUdaljenosti(udaljenosti)


        if not self.getDaLiJeNeusmjereni():
            for u, v, w in self._graf:
                    if udaljenosti[u] != float("Inf") and udaljenosti[u] + w <
udaljenosti[v]:
                    raise TypeError("Graf sadrzi ciklus negativne tezine!")
                    return

        self.ispisiPuteveDoCvorova(izvor, prethodnici)
```

4.2 Implementacija vizualizacije u tabelarnom obliku

Za potrebe implementacije tabele korištene su biblioteke Matplotlib i Pandas.
Biblioteka pandas omogućava efikasno i jednostavno upravljanje za programski jezik Python. Za potrebe realizacije ovog dijela projekta korištena je naredba koja formatira podatke koji se upisuju u tabelu. [6]
Matplotlib je Pythonova biblioteka koja omogućava kreiranje različitih dvodimenzionalnih figura, između ostalog i tabela.

Crtanje tabele izvršava se pozivom metode klase *Graf*, koja se zove *praviTabelu*. Ova metoda prima sljedeće parametre:

- *niz* - dvodimenzionalna matrica podataka koji se upisuju u tabelu, dimenzija (n+1)x3, gdje je n broj čvorova u grafu.
- *fig* - figura na kojoj se prikazuje tabela
- *ax* - pripadajuća osa na figuri u kojoj se prikazuje tabela (potrebno zbog istovremenog prikazivanja tabele i grafa na istoj figuri)
- *br_redova* - broj čvorova grafa

- *indeks_promjene* - i razloga što se tabela mijenja u vremenu, potrebno je na neki način označiti ćeliju tabele u kojoj je vrijednost promijenjena, radi lakšeg praćenja. Ova varijabla čuva podatak u kojem redu se nalazi ta ćelija.
- *kolona_promjene* - čuva podatak u kojoj se koloni nalazi ćelija koja treba biti označena.

Metoda *praviTabelu* je data u nastavku.

```
def praviTabelu(niz, fig, ax, br_redova, indeks_promjene, kolona_promjene=1):
    df = pd.DataFrame(niz)
    boje_kolona = ["#AB0101", "#AB0101", "#AB0101"]
    boje_redova = [["#2C7BEE"] * 3] * br_redova
    t = ax.table(cellText=df.values,
                 cellColours=boje_redova,
                 cellLoc='center',
                 colWidths=[0.1, 0.1, 0.4],
                 colLabels=df.columns,
                 colColours=boje_kolona,
                 loc='center')
    t.scale(2, 2)
    if indeks_promjene != -1:
        t._cells[(indeks_promjene + 1, kolona_promjene)].set_facecolor("#D32C2C")
```

Ova metoda radi na sljedećem principu. Na početku rada programa se generiše početna tabela, kao na slici. Graf ima 11 čvorova, tako da tabela ima 12 redova. Postoje 3 kolone, ‘Cilj’, ‘Duzina’ i ‘Put’. U prvoj koloni su u rastućem poretku sortirani svi čvorovi. U drugoj koloni su trenutno određene dužine između početne tačke i svakog čvora. Inicijalno su sve dužine postavljene na beskonačno. Konačno, u trećoj koloni se nalaze najkraći putevi od početne tačke do svake druge. Ove vrijednosti se upisuju u tabelu tek nakon što algoritam odredi najbolju. Na sljedećim slikama je prikazana opisana procedura.

Cilj	Duzina	Put
0	inf	0
1	inf	0
2	inf	0
3	inf	0
4	inf	0
5	inf	0
6	inf	0
7	inf	0
8	inf	0
9	inf	0
10	inf	0
11	inf	0

Slika 4.1. Početna generisana tabela

Cilj	Duzina	Put
0	0	0
1	10	0
2	12	0
3	14	0
4	2	0
5	7	0
6	10	0
7	20	0
8	3	0
9	5	0
10	7	0
11	8	0

Slika 4.2. Prikaz tabele dok algoritam pronalazi nove najkraće puteve

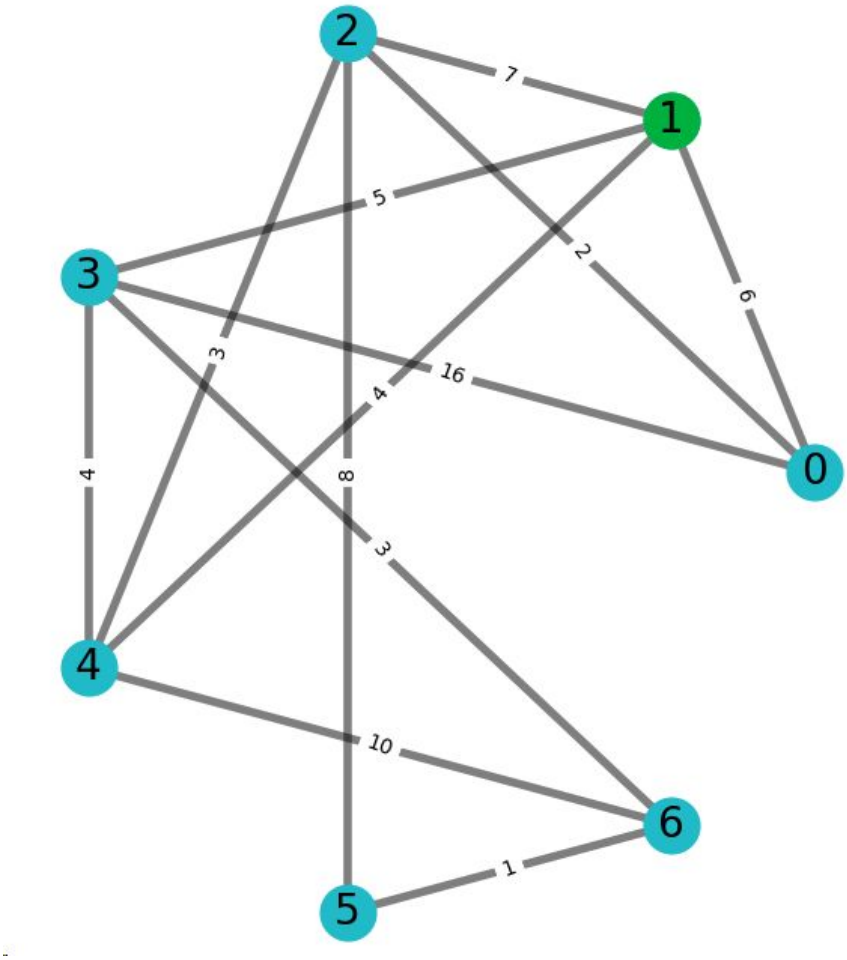
Cilj	Duzina	Put
0	0	0
1	10	0 - 1
2	11	0 - 4 - 9 - 5 - 6 - 2
3	12	0 - 4 - 9 - 5 - 6 - 3
4	2	0 - 4
5	7	0 - 4 - 9 - 5
6	10	0 - 4 - 9 - 5 - 6
7	13	0 - 4 - 9 - 10 - 11 - 7
8	3	0
9	5	0
10	7	0
11	8	0

Slika 4.3. Prikaz tabele nakon što je algoritam završen, te se u nju upisuju najkraći putevi

4.3 Implementacija vizualizacije u grafovskom obliku

Za implementaciju vizualizacije u grafovskom obliku korištene su biblioteke Networkx i Matplotlib. [7] Postoje dvije različite metode za crtanje grafa i to su *Crtaj* i *CrtajPut*. Funkcija Crtaj je korištena tokom traženja i optimizacije ruta, dok je funkcija *CrtajPut* korištena za ispisivanje nađenog najkraćeg puta, te se poziva nakon što je algoritam završen. Prilikom vizualizacije bilo je nužno omogućiti da i tabela i graf budu u istom prozoru, te da se podudaraju trenutne informacije na oba vida vizualizacije.

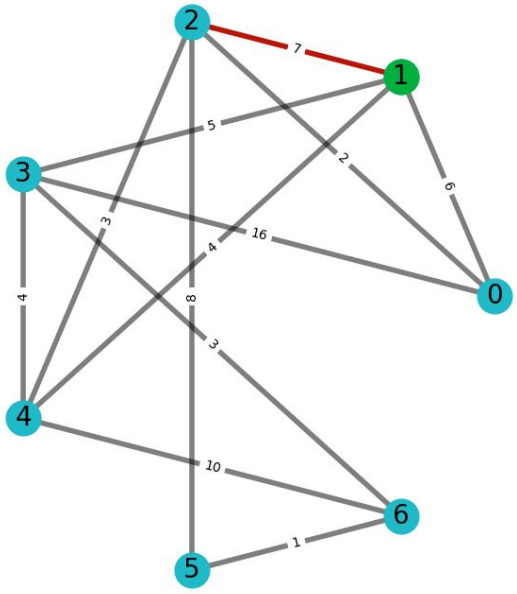
Funkcija *Crtaj* je metoda klase Graf i od parametara prima samo varijablu *par*, koja predstavlja par čvorova koji čine granu koja se trenutno razmatra u optimizaciji. Tako da početni dijagram predstavlja graf sa svim granama iste boje, jer se trenutno algoritam nije počeo razmatrati. Početni čvor je prikazan drugačijom bojom kao što se vidi na slici 4.4.



Slika 4.4: Početni graf

Tokom izvršavanja algoritma na grafu će se prikazivati grana koja se trenutno optimizuje. Na slici 4.5 vidimo čitavu i tabelu i graf i kako izgleda za razmatranje određene grane. Na pomenutoj slici vidimo da je dužina za neke ciljeve inf (beskonačno), to je zato što ti ciljevi još nisu razmatrani.

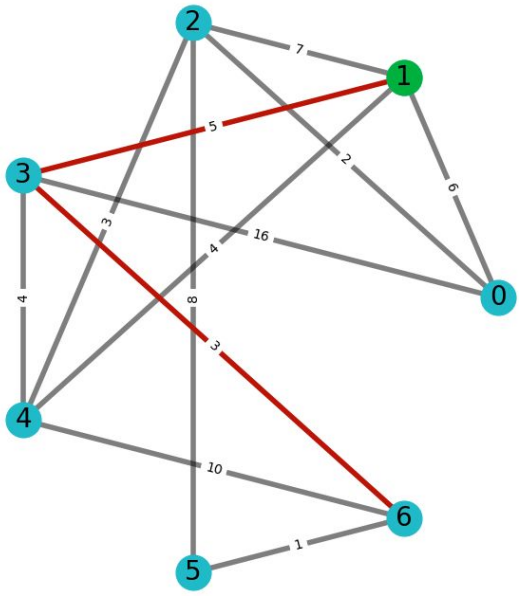
Cilj	Duzina	Put
0	6	0
1	0	1
2	7	0
3	5	0
4	4	0
5	inf	0
6	inf	0



Slika 4.5. Vizualizacija razmatranja određene grane

Nakon završetka algoritma u tabeli će se prikazivati najkraći put od odredišta do cilja, te će se taj cilj nacrtati i na grafu drugom bojom. Za ovo je korištena funkcije CrtajPut koja prima varijable put i zadnji, koje nam govore koji je najkraći put za određeni cilj i da li je taj cilj zadnji koji se razmatra. Rezultat izvršavanja je prikazan na slici 4.6. Ova druga varijabla zadnji nam služi samo kako bi zadržali našu vizualizaciju određeni duži period, umjesto da nam se ona ugasi.

Cilj	Duzina	Put
0	6	1 - 0
1	0	1
2	7	1 - 2
3	5	1 - 3
4	4	1 - 4
5	9	1 - 3 - 6 - 5
6	8	1 - 3 - 6



Slika 4.6. Vizualizacija najkraćeg puta

Zaključak

Kroz ovaj rad su predstavljeni osnovni pojmovi potrebni za razumijevanja rada sa grafovima, a kako bi se opisao Bellman-Fordov algoritam koji je i bio tema rada. Iz iznesenog o ovom algoritmu može se zaključiti da je Bellman-Fordov algoritam veoma koristan za pronaleženje najkraćeg puta između jednog čvora i ostalih dokučivih čvora jednog grafa, a pogotovo u slučajevima gdje su cijene grana grafa predstavljene negativnim vrijednostima za razliku od nekih sličnih algoritama.

Poznata je činjenica da ljudi o općem slučaju brže usvajaju materiju ukoliko je popraćena nekom vizualnom informacijom odnosno grafičkim sadržajem dakle vizualizacija rada algoritma može pomoći u procesu savladavanja Bellman-Fordovog algoritma i njegovog načina rada osobama koje se prvi put susreću sa ovim algoritmom. Kroz ovaj rad je predložen jedan način vizualizacije rada Bellman-Fordovog algoritma sa ciljem da se ista koristi za savladavanje načina rada algoritma.

Literatura

[1] Željko Jurić, *Predavanja na predmetu: Diskretna matematika*, Elektrotehnički fakultet, Univerzitet u Sarajevu, Š.G. 2018./2019.

[2] Wikipedia contributors, *Bellman–Ford algorithm*, Wikipedia, The Free Encyclopedia, 12.12.2018., dostupno na: https://en.wikipedia.org/w/index.php?title=Bellman%E2%80%93Ford_algorithm&oldid=873252848

[3] Abdullah Al Mamun, *Bellman-Ford algorithm*, Kanada, 14.2.2012., dostupno na: http://www.cs.mun.ca/~kol/courses/6783-w12/bellman-ford-scribe.pdf

[4] Richard Stotz, *The Bellman-Ford Algorithm*, München, 2013., dostupno na: https://www-m9.ma.tum.de/graph-algorithms/spp-bellman-ford/index_en.html

[5] Steven Halim, *VISUALGO-Visualising data structures and algorithms through animation*, National University of Singapore, Singapur, dostupno na: https://visualgo.net/en/sssp?slide=1

[6] Aric Hagberg, Dan Schult, Pieter Swart, *NetworkX Reference,* 13.1.2019., dostupno na: https://networkx.github.io/documentation/latest/_downloads/networkx_reference.pdf

[7] Wes McKinney, PyData Development Team, *pandas: powerful Python data analysis toolkit*, 3.2.2019., dostupno na: https://pandas.pydata.org/pandas-docs/stable/pandas.pdf