UDC: 519.1:004.421.2 Info M: str. 26-33

IMPLEMENTACIJA ALGORITAMA IZ TEORIJE GRAFOVA
IMPLEMENTATION OF GRAPH THEORY ALGORITHMS

Zlatko Dejanović, Milorad Božić

REZIME: U ovom radu je opisana jednostavna desktop aplikacija koja izvršava osnovne operacije na grafovima. Osnovni motiv za realizaciju aplikacije leži u velikoj primjeni principa iz teorije grafova u oblasti računarstva i drugim disciplinama. U radu je dat kratak pregled grafova tipa stabla, te Ojlerovih i Hamiltonovih grafova kao i ideje za praktičnu realizaciju problema u navedenim podoblastima. Sve tehnike u aplikaciji su realizovane grafički. Kao alat je korišćeno razvojno okruženje Visual Studio 2012 i programski jezik C#.

KLJUČNE REČI: teorija grafova, softver

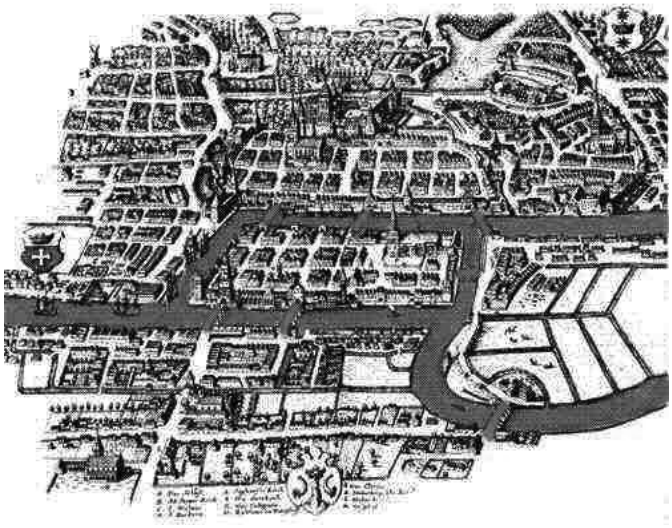
ABSTRACT: This paper describes the implementation of a simple desktop application that performs basic operations on graphs. The motive for the implementation lies in the significant usage of graph theory principles in the field of computing. The paper gives a brief overview of the trees, Euler and Hamilton graphs as well as the ideas for practical implementation problems in these subfields. All of the techniques in the application are implemented in a graphical way. Visual Studio 2012 and C# are used as tools for programming.

KEY WORDS: graph theory, software

1. UVOD

Teorija grafova predstavlja granu matematike koja ima vrlo veliku primjenu u različitim oblastima računarstva i informatike, počevši od računarskih mreža preko operativnih sistema do struktura podataka i računarskih algoritama.

Istorija teorije grafova počinje u XVIII vijeku sa Lenardom Ojlerom [1] [2] [3] i njegovim čuvenim problemom Keningsberških mostova koji su predstavljeni na Slici 1. Ojler je pokazao da se prilikom šetanja ne mogu obići svi mostovi tako da se preko svakog od njih pređe samo jednom.



Slika 1: *Problem Keninsberških mostova [4]*

Teorija grafova evoluira u nauku u prvoj polovini XX vijeka i jedna je od oblasti matematike u kojoj se generišu značajni teoretski i praktični rezultati [1]. Preplitanje teorije grafova i računarskih nauka proizilazi iz činjenice da grafovi u mnogim slučajevima predstavljaju podesan alat za modelovanje i rješavanje problema koji se mogu susresti u računarskim sistemima.

Uprkos širokoj primjeni teorije grafova, ne postoji veliki broj softverskih rješenja koja se bave vizuelizacijom njenih osnovnih principa. Neka od njih su Mathematica [5], Graphviz [6] i NetworkX [7]. Nedostaci navedenih alata, kao što su cijena ili lakoća upotrebe, bili su glavna motivacija za realizaciju jednostavne desktop aplikacije u kojoj su vizuelno realizovani neki od najčešće korišćenih koncepata.

U realizovanom softverskom rješenju korisniku je omogućeno da vizuelno predstavi jednostavan graf[1], a nakon toga da se nad tim grafom obave neke od sljedećih radnji:

1. crtanje šetnji u Ojlerovim i polu-Ojlerovim grafovima po Flurijevom algoritmu [1] [2];
2. crtanje Dajkstrinog stabla za svako tjeme grafa [1] [2];
3. izračunavanje minimalnog razapinjućeg stabla grafa prema Prim-Jarnikovom algoritmu [1] [3];
4. računanje broja različitih razapinjućih stabala prema Kirhofovoj matričnoj teoremi [2] [3] ;
5. traženje mostova u grafu prema Tardžanovom algoritmu [8] [9];
6. traženje Hamiltonovog puta prema Oreovom kriterijumu i Palmerovom algoritmu[3],
7. računanje Pruferovog koda za označeno stablo [2].

Kao programski jezik je korišten C#, a kao razvojno okruženje korišten je *Visual Studio* 2012.

2. PREGLED OSNOVNIH PRINCIPA IZ TEORIJE GRAFOVA

U ovom dijelu rada je dat pregled osnovnih pojmova i algoritama iz teorije grafova. Naglasak je dat na objašnjenje koncepata i algoritama koji su iskorišćeni prilikom programiranja, a svjesno su izostavljeni dokazi teorema koji se mogu pronaći u odgovarajućoj matematičkoj literaturi [1] [2] [3]. Takođe, matematička izvođenja su svedena na minimum.

1 Jednostavan graf je graf bez petlji i sa najviše jednom ivicom između dva tjemena.

Softverska rješenja u razmatranoj aplikaciji se odnose na:

- Grafove tipa stabla;
- Ojlerove grafove;
- Hamiltonove grafove.

2.1 *Grafovi tipa stabla*

Jedan od problema iz teorije grafova koji je našao veliku primjenu u računarskim naukama je svakako problem pronalaženja minimalnog razapinjućeg stabla. Ideja iza minimalnog razapinjućeg stabla može se uočiti na osnovu sljedećeg primjera. Ako su v_1, ..., v_n gradovi u nekoj državi, a c_{ij} cijena da se v_i i v_j povežu autoputem, treba napraviti mrežu autoputeva tako da se iz svakog grada v_i može doći do v_j i to na način da cijena bude minimalna. Dva najpoznatija algoritma za rješavanje navedenog problema su Kruškalov[2] i Prim – Jarnikov[3] algoritam. Za programsku implementaciju je pogodniji Prim – Jarnikov algoritam pošto kod njega u svakom koraku algoritma postoji povezan graf. Kod Kruškalovog algoritma je potrebno u svakom koraku ispitivati da li nova ivica kada se nadoveže na postojeće stablo čini ciklus, a to nije računarski jednostavan problem [10].

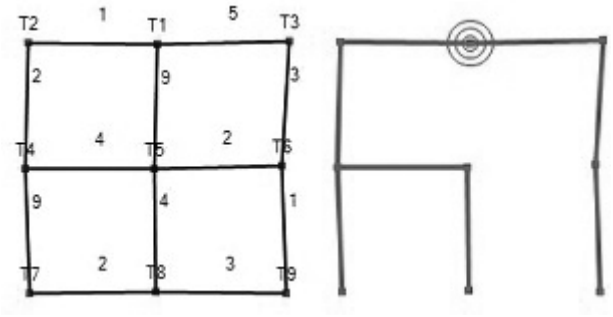
Prim – Jarnikov algoritam se sastoji iz sljedećih koraka:

1. Bira se proizvoljno tjeme iz originalnog grafa;
2. Bira se ivica koja polazi iz izabranog tjemena takva da je njena težina manja ili jednaka težini bilo koje druge ivice koja ide iz tog tjemena;
3. Ako su već odabrana određena tjemena i određene ivice, bira se nova ivica koja nije izabrana tako da budu ispunjena sljedeća dva uslova:
   a. Jedno njeno tjeme je već izabrano, a drugo nije;
   b. Težina izabrane ivice je minimalna u skupu svih mogućih ivica koje idu iz jednog od izabranih tjemena do jednog od onih koja još nisu izabrana.

Dajkstrin algoritam[4] predstavlja popularan algoritam kojem je cilj da pronađe put minimalne cijene od jednog tjemena grafa do svih ostalih. Kao što će se vidjeti u implementaciji, ideja je vrlo slična Prim – Jarnikovom algoritmu. Koraci Dajkstrinog algoritma za minimalno stablo za unaprijed određenim tjemenom kao korijenom su sljedeći:

1. Dodijeliti svakom tjemenu početne udaljenosti od korijena – korijenu vrijednost nula, a ostalim tjemenima vrijednost beskonačno;
2. Označiti sva tjemena kao neodabrana, a korijen kao odabran;
3. Dodati u stablo i označiti prethodno neoznačeno tjeme koje je direktno povezano sa jednim od prethodno označenih tjemena takvo da mu je udaljenost od korijena stabla najmanja;
4. Ponavljati korak 3 dok se ne označe sva tjemena.

Primjer primjene Dajkstrinog algoritma dat je na Slikama 2 i 3. Kao tjeme tipa korijena je odabrano tjeme T1. Svaki novi red u tabeli sa Slike 3 treba posmatrati kao sljedeći korak algoritma.



Slika 2: *Primjer izračunavanja Dajkstrinog algoritma*

	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1	0	∞	∞	∞	∞	∞	∞	∞	∞
T2	0	1	5	∞	9	∞	∞	∞	∞
T4	0	1	5	3	9	∞	∞	∞	∞
T3	0	1	5	3	7	∞	12	∞	∞
T5	0	1	5	3	7	8	12	∞	∞
T6	0	1	5	3	7	8	12	11	∞
T9	0	1	5	3	7	8	12	11	9
T8	0	1	5	3	7	8	12	11	9
T7									

Slika 3: *Koraci Dajkstrinog algoritma za graf sa Slike 2*

Jedan od problema koji se mogu susresti je i određivanje ukupnog broja razapinjućih stabala u jednom grafu. Odgovor na to pitanje nudi Kirhofova matrična teorema. Ideja je da se graf sa n tjemena predstavi pomoću matrice dimenzija $n \times n$, tako da su ispunjeni sljedeći uslovi:

1. Za elemente matrice za koje je $i = j$, staviti da su jednaki stepenu[5] tjemena i;
2. Za ostale elemente matrice staviti vrijednost $-d_{ij}$, gdje d_{ij} predstavlja broj ivica između tjemena i i j. Očigledno, u slučaju jednostavnih grafova, odgovarajuća polja za elemente matrice mogu da sadrže samo vrijednosti -1 i 0.

Nakon toga, broj ukupnih razapinjućih stabala u grafu predstavlja determinanta matrice koja se dobije kada se iz matrice formirane na opisani način izostave proizvoljna vrsta i proizvoljna kolona.

U aplikaciji je realizovano i određivanje Pruferovog koda[6] za svako stablo dobijeno Dajkstrinim algoritmom. (Više o Pruferovom kodu može se naći u [2]).

2 Joseph Kruskal, 1956.

3 Vojtech Jarnik, 1930. – Robert C. Prim, 1957.

4 Edsger Dijkstra, 1959.

5 Stepen tjemena u grafu predstavlja broj drugih tjemena sa kojima dato tjeme formira ivice.

6 Heinz Prufer, 1918.

2.2 *Ojlerovi grafovi*

Ciklus generalno predstavlja zatvorenu putanju u grafu. Ojlerov ciklus sadrži sve ivice grafa. Graf je Ojlerov ako i samo ako ima Ojlerov ciklus. Sa druge strane, graf je polu-Ojlerov ako i samo ako sadrži Ojlerovu stazu [1].

Postoji jasna karakterizacija kada je graf Ojlerov, a kada polu-Ojlerov. Graf je Ojlerov ako i samo ako je povezan i ima sva tjemena parnog stepena[7]. Graf je polu-Ojlerov ako i samo ako je povezan i ima najviše dva tjemena neparnog stepena. Ojlerov ciklus može da se započne iz bilo kojeg tjemena u grafu, dok se šetnja u polu-Ojlerovom grafu mora započeti iz jednog tjemena neparnog stepena i završiti u drugom tjemenu neparnog stepena.

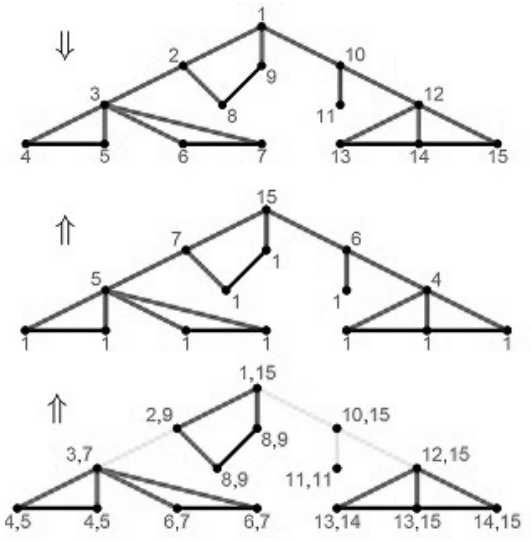
Dok je programska realizacija ispitivanja da li je graf Ojlerov ili polu-Ojlerov trivijalna, samo iscrtavanje šetnje kroz graf je prilično komplikovano. U programu je korišten Flurijev algoritam[8] koji se za Ojlerov graf sastoji iz sljedećih koraka [3]:

1. Izabrati proizvoljno tjeme iz grafa;
2. Bira se ivica koja još nije izabrana kao incidentna sa izabranim tjemenom takva da nije most u grafu dobijenom izostavljanjem izabranih ivica iz originalnog grafa. Ako ne postoji takva ivica, izabrati most;
3. Ponavljati korak 2.

U koraku 2 je sadržana složenost algoritma. Most u grafu je ivica čijim se izostavljanjem smanjuje povezanost grafa. Utvrđivanje da li je ivica most obavlja se primjenom Tardžanovog algoritma[9]. Tardžanov algoritam je prilično složen, a njegovi koraci su:

1. Pronaći bilo koje razapinjuće stablo. Kao tjeme korijena može biti postavljeno bilo koje tjeme. Nakon toga je bitno definisati odnose prethodnik (predak) – nasljednik (potomak) u dobijenom stablu. Kao i u svakom stablu, sva tjemena osim korijenog imaju tačno jednog prethodnika;
2. Stablo treba numerisati u *preorder* redoslijedu. Po *preorder* redoslijedu svaki nasljednik će imati *preorder* vrijednost veću od svog prethodnika;
3. Pronaći ND vrijednosti za svako tjeme stabla. ND vrijednost predstavlja ukupan broj nasljednika datog tjemena uvećan za 1;
4. Izračunati *low* i *high* vrijednosti za svako tjeme stabla. Prilikom traženja ovih vrijednosti prvo ih inicijalizovati sa *preorder* vrijednostima. Nakon toga, treba provjeriti da li direktno povezano tjeme preko ivice iz originalnog grafa koja nije u stablu ima manju ili veću *preorder* vrijednost. Ako ima manju, dodijeliti je kao *low* vrijednost tjemenu za koje se vrši provjera. Analogno tome, uraditi i za high vrijednost ako je *preorder* vrijednost veća. Nakon toga, proći kroz sva tjemena tipa nasljednika u stablu i provjeriti da li neko od njih ima manju ili veću *preorder* vrijednost. Najmanju i najveću dodijeliti kao *low* i *high* vrijednost datom tjemenu;
5. Ako je za neko tjeme *low* vrijednost jednaka *preorder* vrijednosti, a zbir *preorder* i ND vrijednosti veći od *high* vrijednosti, onda ivica incidentna sa datim tjemenom i njegovim prethodnikom u stablu predstavlja most u originalnom grafu.

Primjer grafa sa izračunatim vrijednostima koje su naprijed opisane dat je na Slici 4.



Slika 4: *Tardžanov algoritam: a) preorder vrijednosti tjemena, b) ND vrijednosti tjemena, c) low i high vrijednosti tjemena.*

2.3 *Hamiltonovi grafovi*

Hamiltonov ciklus u grafu G predstavlja ciklus u grafu koji sadrži sva tjemena grafa G, ali tako da se tjemena ne ponavljaju. Graf je Hamiltonov ako i samo ako ima Hamiltonov ciklus[10].

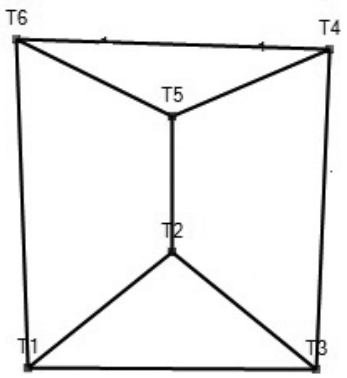
Hamiltonovi ciklusi predstavljaju jednu od najvećih enigmi u polju teorije grafova pošto ne postoji ekvivalencija koja se može upotrebiti za provjeru da li je graf Hamiltonov. Postoje samo parcijalne karakterizacije, međutim sve one nude dokaz samo u jednom smjeru. Jedna od takvih karakterizacija je i Oreova teorema[11] koja kaže da ako je zbir stepeni bilo koja dva nesusjedna tjemena u grafu veći ili jednak od broja tjemena, onda je graf Hamiltonov. Nažalost, obrnuto ne važi. Na primjer, svaka kontura je sigurno Hamiltonov graf, međutim za konturu sa pet tjemena gornja polazna pretpostavka očigledno ne važi (stepen svakog tjemena je 2, pa je zbir dva stepena očigledno manji od 5). U programu je realizovan Oreov kriterijum, međutim jasno je naglašeno da njegov neprolazak ne znači nužno da graf nije Hamiltonov.

Na osnovu Oreove teoreme napravljen je Palmerov algoritam[12] za formiranje Hamiltonovog puta u slučaju da je ispunjen Oreov kriterijum. Njegovi koraci su sljedeći [3]:

7 Leonard Euler, 1736.
8 Fleury, 1883.
9 Robert Tarjan, 1976.
10 William Rowan Hamilton, 1857.
11 Øystein Ore, 1960.
12 E. M. Palmer, 1997.

1. Aranžirati tjemena grafa u konturu slučajno;
2. Tražiti kroz konturu *gap*. *Gap* predstavlja dva tjemena koja su u aranžiranoj konturi jedno do drugog, a nisu susjedna u grafu. Nakon toga, naći par uzastopnih tjemena u konturi (njihova međusobna povezanost u originalnom grafu nije bitna) takav da je prvo tjeme iz para povezano sa prvim tjemenom iz *gap*-a, a drugo tjeme iz para povezano sa drugim tjemenom iz *gap*-a;
3. Ako takav par postoji, rearanžirati cirkularni redoslijed tjemena u konturi od drugog u *gap*-u do prvog u nađenom paru (uključujući i njih) u obrnutom redoslijedu.

Primjer dobijanja Palmerove sekvence ilustrovan je na grafu sa Slike 5.



Slika 5: *Graf za računanje Palmerove sekvence za Hamiltonov ciklus*

Koraci za dobijanje Palmerove sekvence su dati u tabeli na Slici 6. Na početku su tjemena raspoređena ciklično na slučajan način. Nakon toga je svaki *gap* označen bold fontom, a par korespondirajućih tjemena označen crvenom bojom.

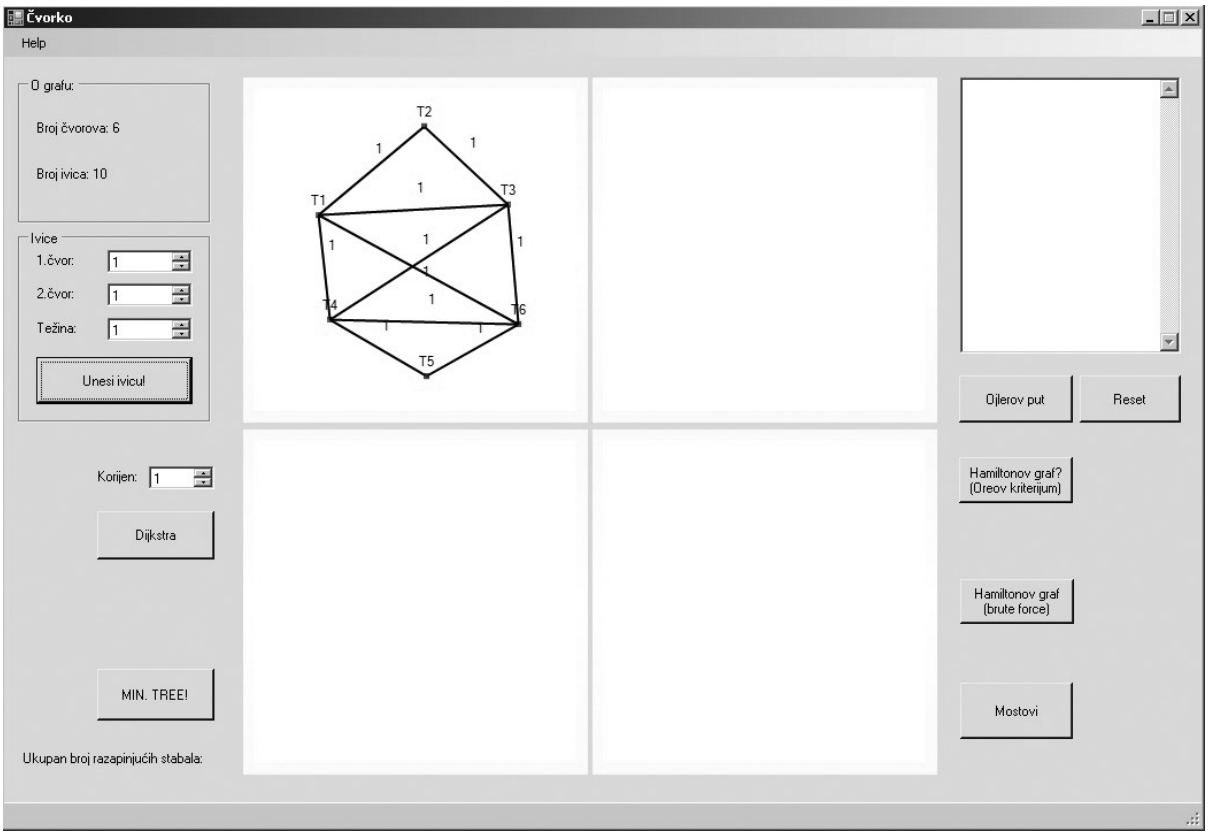
T1	**T4**	T2	T6	T3	T5	T1
T1	**T2**	**T4**	T6	T3	T5	T1
T1	T2	**T3**	**T6**	T4	T5	T1
T1	T2	T3	T4	T6	**T5**	**T1**
T2	T1	T3	T4	T6	T5	T2

Slika 6: *Koraci za računanje Palmerove sekvence za graf sa Slike 5*

U programu je osim Hamiltonovog puta po Palmerovom algoritmu, realizovan i *brute force* pristup, međutim jasno je da isti ne može biti efikasan za veći broj tjemena zbog ogromnog broja mogućih permutacija tjemena.

3. SOFTVERSKA IMPLEMENTACIJA

Za implementaciju softvera korišteno je *Microsoft*-ovo razvojno okruženje *Visual Studio 2012* i programski jezik C#. Za crtanje grafova korištena je komponenta *Chart* tipa *Point*. Pošto ova komponenta predstavlja *Point* dijagrame i ne nudi mogućnost povezivanja pojedinih tačaka, za njihovo povezivanje korištena je funkcija *DrawLine* kojoj su kao parametri šalju koordinate pojedinih tačaka na dijagramu. Izgled aplikacije je prikazan na Slici 7.



Slika 7: *Interfejs razvijene aplikacije*

Unos tjemena omogućen je proizvoljnim pritiskom tastera od miša na prvu oblast. Ivice grafa se unose pomoću forme u kojoj je potrebno unijeti incidentna tjemena sa ivicom. Takođe, omogućeni su unos i prikaz težina pojedinih ivica. Težinski grafovi nisu potrebni za sve aspekte aplikacije, ali su neophodni za izračunavanja kod Dajkstrinog i Prim – Jarnikovog algoritma.

Graf je programski predstavljen kao instanca istoimene klase. Glavni atribut klase je matrica susjedstva. Pošto su u programu omogućeni samo grafovi u kojima tjeme ne može biti povezano samo sa sobom, matrica susjedstva je malo modifikovana u odnosu na definiciju i realizovana je na sljedeći način:

- Ako je broj tjemena *n*, dimenzije matrice su *n x n*.
- Ako su dva tjemena sa rednim brojevima *x* i *y* povezana ivicom težine *t*, onda se na pozicije [*x*-1, *y*-1] i [*y*-1, *x*-1] smješta vrijednost *t*.
- Ako dva tjemena sa rednim brojevima *x* i *y* nisu direktno povezana, onda se na pozicije [*x*-1, *y*-1] i [*y*-1, *x*-1] smješta vrijednost 0.
- Na pozicije [*x*, *x*] se takođe smiješta vrijednost 0.

Drugi bitan atribut u klasi *Graf* je niz tjemena grafa. Ovaj atribut je bitan zbog propertija *broj* koji posjeduje svaka instanca klase *Cvor*, a koji predstavlja redni broj tjemena koji je i prikazan na grafičkoj reprezentaciji grafa.

3.1 *Stabla – implementacija*

Koraci za izračunavanje Dajkstrinog stabla za tjeme *x* su implementirani pomoću sljedećeg segmenta koda. Glavna petlja se izvršava dok ima nemarkiranih tjemena. U svakom koraku petlje markira se jedno tjeme i dodaje u stablo na označenu poziciju. Za maksimalnu težinu ivice je uzet broj 1000 i on u implementiranom algoritmu predstavlja beskonačno veliku vrijednost. Na kraju se poziva posebna rekurzivna funkcija za postavljanje prethodnika u stablu. Ovo se radi iz preventivnih razloga ako se stablo bude naknadno koristilo u neku drugu svrhu gdje je neophodno da postoji definisan prethodnik za svako tjeme.

```
public Graf dajkstra(int x)
{
    Graf stablo = copy();
    stablo.brojIvica = n-1;

    int[] marked = new int[30];
    int[] result = new int[30];

    for (int i = 0; i < n; i++)
result[i] = 1000;

    int root = x;
    marked[root] = 1;
    result[root] = 0;

    for (int i = 0; i < n - 1; i++)
    {
int min = 1000;
int redbr = 0;
int redbr2 = 0;
for (int j = 0; j < n; j++)
{
    for (int k = 0; k < n; k++)
    {
if (marked[j] > 0 && marked[k] == 0)
{
    if (matrica[j, k] > 0)
    {
if (min > (matrica[j, k] + result[j]))
{
    min = matrica[j, k] + result[j];
    redbr = k;
    redbr2 = j;
}
if (result[k] > matrica[j, k] + result[j])
    result[k] = matrica[j, k] + result[j];
    }
}
    }
}
marked[redbr] = 1;

stablo.matrica[redbr2, redbr] = matrica[redbr2, redbr];
stablo.matrica[redbr, redbr2] = matrica[redbr, redbr2];
    }
    stablo.postaviOceve(root, root);

    return stablo;
}

public void postaviOceve(int ind, int root)
{
    markedD[root] = 1;
    for (int i = 0; i < n; i++)
    {
if (matrica[ind, i] > 0 && markedD[i] == 0)
{
    cvorovi[i].father = ind + 1;
    markedD[i] = 1;
    postaviOceve(i, root);
        }
    }
}
```

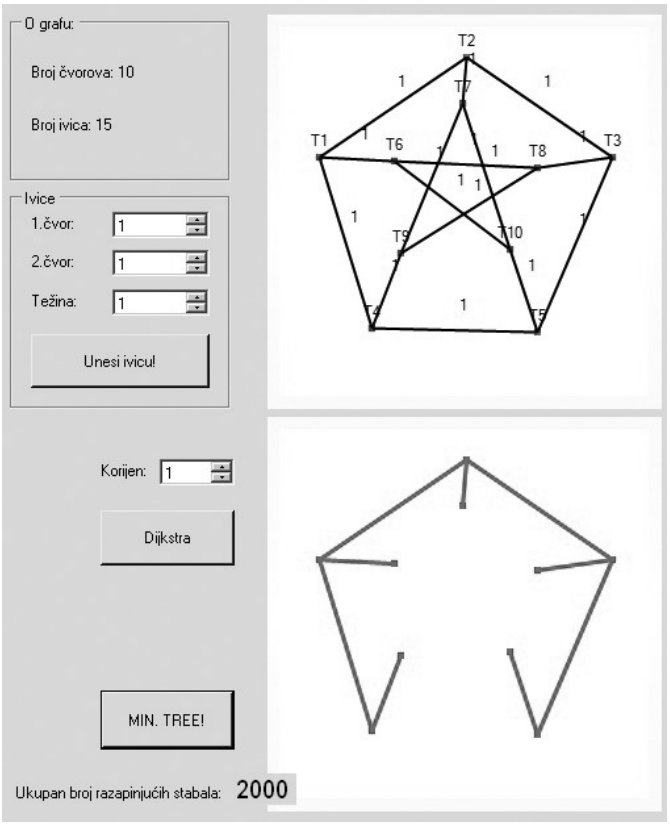
Kompletan kod za izračunavanje Prim – Jarnikovog minimalnog razapinjućeg stabla nije u potpunosti predstavljen u radu pošto je vrlo sličan navedenom Dajkstrinom algoritmu i vrlo lako ga je izvesti iz date implementacije. Jedina razlika je unutar dvostruke *for* petlje, gdje se u svakom koraku ne gleda udaljenost od *root*-a, već se za minimum postavlja sljedeća ivica sa najmanjom težinom koja se "lijepi" na stablo. U sljedećem segmentu koda predstavljena je opisana razlika.

```
if (marked[j] > 0 && marked[k] == 0)
{
    if (matrica[k, j] > 0)
    {
if (min > matrica[j, k])
{
    min = matrica[j, k];
    redbr = k;
    redbr2 = j;
}
    }
}
```

Izračunavanje ukupnog broja razapinjućih stabala je implementirano pomoću sljedećeg segmenta koda. Ovaj kod je dobijen na osnovu Kirhofove matrične teoreme koja se oslanja na izračunavanje determinante. Pošto je funkcija za izračunavanje determinante matrice reda *n* realizovana rekurzivno, ovdje ona predstavlja jedno od uskih grla sistema. Naime, ako u grafu postoji više od deset tjemena, izračunavanje determinante postaje računarski zahtjevno i ne preporučuje se iako je ostavljena mogućnost za pozivanje pomenute funkcije.

```
public int spanningTreeBroj()
{
    int[,] Kirhofovamatrica = new int[30, 30];

    for (int i = 0; i < n - 1; i++)
    {
for (int j = 0; j < n - 1; j++)
{
   if (i == j) Kirhofovamatrica[i, j] = stepenCvora(i);
    else
    {
if (matrica[i, j] == 0) Kirhofovamatrica[i, j] = 0;
else Kirhofovamatrica[i, j] = -1;
    }
}
    }

    return (Determinant(n - 1, Kirhofovamatrica));
}
```

Na Slici 8 je predstavljen način na koji je pomoću aplikacije izračunat ukupan broj razapinjućih stabala za popularni Petersenov graf koji sadrži deset tjemena. Za laptop prosječne konfiguracije je potrebno oko 20 sekundi da se izvrše odgovarajuća izračunavanja u ovom primjeru.



Slika 8: *Računanje ukupnog broja razapinjućih stabala za Petersonov graf*

3.2 *Ojlerovi grafovi - implementacija*

Programski kod za implementaciju Flurijevog algoritma nije realizovan kao posebna metoda klase *Graf*, već je direktno ugrađen u interfejs zbog svoje jednostavnosti, ali i zbog načina implementacije. Naime, u klasi je kreiran properti *ojlerovBrojac* koji se prilikom ispisivanja svake nove ivice inkrementira. Razlog za ovo je ideja da se prilikom svakog novog pozivanja događaja za ispisivanje Ojlerovog ciklusa dodaje sljedeća ivica. Na ovaj način je olakšano praćenje Ojlerove šetnje kroz graf.

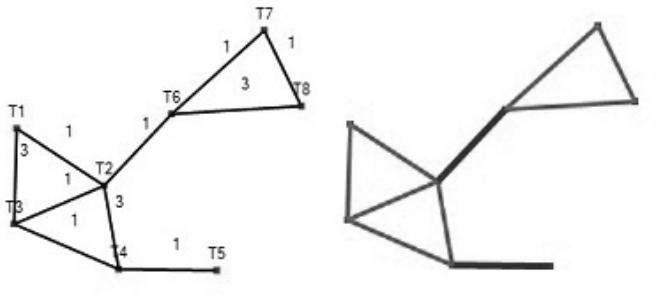
Kao što je objašnjeno u drugom odjeljku ovog rada, ključ prilikom biranja sljedeće ivice Ojlerove šetnje predstavlja ispitivanje da li je posmatrana ivica most. U sljedećem segmentu programskog koda data je implementacija Tardžanovog algoritma koji za dato tjeme provjerava da li ivica incidentna sa njim i njegovim prethodnikom predstavlja most.

```
public int isBridge(int a)
{
    Graf tempTree = dajkstra(0);

    tempTree.preorderuj(0);
    rBr = 1;

    for (int i = 0; i < tempTree.N; i++)
tempTree.cvorovi[i].ND = tempTree.nadjiND(i);
```

```
    for (int i = 0; i < tempTree.N; i++)
tempTree.cvorovi[i].low = tempTree.cvorovi[i].preorder;

    for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
{
   if (((matrica[i, j] - tempTree.matrica[i, j]) > 0))
    {
if (tempTree.cvorovi[i].preorder > tempTree.cvorovi[j].preorder)
    tempTree.cvorovi[i].low = tempTree.cvorovi[j].preorder;
    }
}

    int[] nizz = new int[n];
    for (int i = 0; i < n; i++)
nizz[i] = tempTree.nadjiLow(i);

    for (int i = 0; i < n; i++)
tempTree.cvorovi[i].low = nizz[i];

    for (int i = 0; i < n; i++)
tempTree.cvorovi[i].high = tempTree.cvorovi[i].preorder;

    for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
{
   if (((matrica[i, j] - tempTree.matrica[i, j]) > 0))
    {
if (tempTree.cvorovi[i].preorder < tempTree.cvorovi[j].preorder)
    tempTree.cvorovi[i].high = tempTree.cvorovi[j].preorder;
    }
}

    int[] nizzz = new int[n];
    for (int i = 0; i < n; i++)
nizzz[i] = tempTree.nadjiHigh(i);

    for (int i = 0; i < n; i++)
tempTree.cvorovi[i].high = nizzz[i];

     if ((tempTree.cvorovi[a].low == tempTree.cvorovi[a].preor-
der) && ((tempTree.cvorovi[a].preorder + tempTree.cvorovi[a].ND)
> tempTree.cvorovi[a].high))
return 1;
    else
return 0;
}
```

Unutar segmenta koda se upotrebljavaju pomoćne funkcije koje nisu ovdje predstavljene, a koriste se za izračunavanje vrijednosti *low*, *high* i *preorder* opisanih u drugoj glavi.

Iako se Tardžanov algoritam unutar programa koristi kao pomoćna funkcija za formiranje Ojlerove šetnje, u aplikaciji je omogućeno i ispisivanje mostova za dati graf. Primjer iz aplikacije je dat na Slici 9 gdje su mostovi označeni ljubičastom bojom.



Slika 9: *Primjer izračunanja mostova u aplikaciji*

3.3 *Hamiltonovi grafovi – implementacija*

Oreov kriterijum je prilično jednostavan i prikazan je u sljedećem segmentu koda. Iako je povoljan zbog svoje jednostavnosti, kao što je već rečeno postoje grafovi koji jesu Hamiltonovi, ali ne zadovoljavaju Oreov kriterijum. Ovaj kriterijum samo daje odgovor na pitanje da li postoji Hamiltonova sekvenca, ali ne otkriva niz tjemena u njoj.

```
public int isHamilton()
{
    for (int i=0; i<n; i++)
for (int j=0; j<n; j++)
    if (matrica[i, j] == 0)
    {
if (stepenCvora(i) + stepenCvora(j) < n)
    return 0;
    }
    return 1;
}
```

Pomoću sljedećeg segmenta koda se implementira izračunavanje Palmerove sekvence ako je ispunjen Oreov kriterijum. Palmerov algoritam predstavlja logičnu dopunu na Oreov kriterijum pošto se kao rezultat vraća put, tj. niz tjemena. Pomoćna funkcija *reverse* je jednostavna funkcija za permutaciju podniza tjemena kako je opisano u drugom odjeljku rada.

```
public int[] hamilton()
{

    int[] hamilton = new int[n + 1];

    for (int i = 0; i < n + 1; i++)
hamilton[i] = i % n + 1;

for(int i=0; i<n; i++)
{
    if(matrica[hamilton[i]-1,hamilton[i+1]-1]==0)
    {
for(int j=i+2; j<n+i+1; j++)
```

```
      if (matrica[hamilton[j % n]-1, hamilton[i % n]-1]
== 1 && matrica[hamilton[(j + 1) % n] - 1, hamilton[(i +
1) % n] - 1] == 1)
      {
   reverse(hamilton, i + 1, j);
   if (i == n - 1) hamilton[n] = hamilton[0];
   break;
      }
      }
   }
       return hamilton;
   }
```

U aplikaciji je omogućeno i *brute force* izračunavanje Hamiltonove sekvence na način da se vrši provjera svake moguće permutacije rednih brojeva tjemena. Ovo naravno predstavlja usko grlo sistema pošto je ukupan broj permutacija n!. Samim tim, program neće efikasno moći da izračuna Hamiltonovu sekvencu za graf koji sadrži preko 10 tjemena (kao i kod računanja determinante Kirhofove matrice).

4. ZAKLJUČAK

Teorija grafova zauzima značajno mjesto u računarskim naukama. Različite diskretne strukture koje se pojavljuju u računarstvu pogodno se opisuju grafovima. Tu spadaju, na primjer, programske strukture (npr. dijagrami toka računarskog programa), strukture podataka (npr. binarna stabla), računarske mreže i dr.

Samim tim, nameće se kao potreba za raspolaganjem alata pomoću kojih se može eksperimentisati sa svim glavnim konceptima iz teorije grafova na jednom mjestu. Polazeći od ovog motiva realizovana je jednostavna, intuitivna aplikacija za rad sa stablima, Ojleorovim i Hamiltonovim grafovima. Kao posebnu prednost aplikacije treba istaći vizuelizaciju grafova realizovanu pomoću dostupnih alata u okruženju *Visual Studio* 2012.

U radu nisu implementirani koncepti iz oblasti planarnih grafova i bojenja grafova. Glavni razlog je nepostojanje dobro poznatih i ispitanih algoritama iz tih oblasti koji bi se mogli efikasno realizovati pomoću raspoloživog programskog jezika.

LITERATURA

[1] Dragoš M. Cvetković, Slobodan K. Simić, Odabrana poglavlja iz diskretne matematike, Akademska misao, Beograd, 2002.
[2] Darko Veljan, Kombinatorika sa teorijom grafova, Školska knjiga, Zagreb, 1989.
[3] James A. Anderson, Discrete Mathematics with Combinatorics, Pearson Education, Inc, 2004.
[4] http://www.maa.org/publications/periodicals/convergence/leonard-eulers-solution-to-the-konigsberg-bridge-problem
[5] http://www.wolfram.com/mathematica/
[6] http://www.graphviz.org/
[7] http://networkx.github.io/
[8] R. Endre Tarjan, A Note on Finding the Bridges of a Graph, Inform. Process Lett., 1974.
[9] David Pritchard, An Optimal Distributed Bridge-Finding Algorithm, ACM Symposium on Principles of Distributed Computing - PODC, 2006.
[10] Michael T. Goodrich, Roberto Tamassia, Data Structures and Algorithms in Java, Second Edition, John Wiley & Sons, Inc, 2001.

Zlatko Dejanović, magistar računarstva i informatike,Univerzitet u Banjoj Luci, Elektrotehnički fakultet, RS, BiH
Kontakt: zlatko.dejanovic (at) etfbl.net
Oblasti interesovanja: računarske mreže, performanse računarskih sistema, kriptografija

dr Milorad Božić, Univerzitet u Banjoj Luci, Elektrotehnički fakultet, RS, BiH
Kontakt: mbozic (at) etfbl.net
Oblasti interesovanja: automatika, vještačka inteligencija