Zadatak 9

Grupa 4

Decembar 2024

1 Stablo i šuma

1.1 Definicija stabla

Stablo je povezan neusmeren graf bez ciklusa. Drugim rečima, to je graf u kome izmeu bilo koja dva čvora postoji tačno jedan prost put.

Formalno: Graf $T = (V, E)$ je stablo ako je povezan i acikličan.

1.2 Definicija šume

Šuma je neusmeren graf bez ciklusa. Šuma može biti nepovezana, i svaka njena povezana komponenta je stablo.

1.3 Primer stabla

$$V = \{A, B, C, D\}, \quad E = \{\{A, B\}, \{A, C\}, \{C, D\}\}$$

Ovo je povezani graf bez ciklusa - dakle, stablo.

1.4 Primer šume

$$V = \{A, B, C, D, E, F\}, \quad E = \{\{A, B\}, \{C, D\}, \{E, F\}\}$$

Tri nepovezane komponente, svaka bez ciklusa - dakle, šuma.

2 Ekvivalentne karakterizacije stabla

Za neusmeren graf $G = (V, E)$ sledećih pet tvrdnji su meusobno ekvivalentne i karakterišu stablo:

1. G je povezan i nema cikluse.
2. G je acikličan i ima tačno $|V| - 1$ ivica.
3. G je povezan i ima tačno $|V| - 1$ ivica.
4. Izmeu bilo koja dva čvora u G postoji tačno jedan prost put.
5. G je acikličan i dodavanjem bilo koje ivice formira se tačno jedan ciklus.

2.1 Leme

Lema 1: Ako je graf G povezan i acikličan, onda ima tačno $|V| - 1$ ivica. $\Rightarrow$ Tvrdnje (1) i (2) su ekvivalentne.

Lema 2: Ako graf ima tačno $|V| - 1$ ivica i acikličan je, tada je povezan. $\Rightarrow$ Tvrdnje (2) i (3) su ekvivalentne.

Lema 3: Ako izmeu bilo koja dva čvora postoji tačno jedan put, graf je povezan i acikličan. $\Rightarrow$ Tvrdnje (1) i (4) su ekvivalentne.

Lema 4: Ako je graf acikličan, i dodavanjem bilo koje nove ivice dobija se tačno jedan ciklus, tada ima $|V| - 1$ ivica i povezan je. $\Rightarrow$ Tvrdnje (1) i (5) su ekvivalentne.

3 Algoritmi za konstrukciju pokrivajuceg stabla

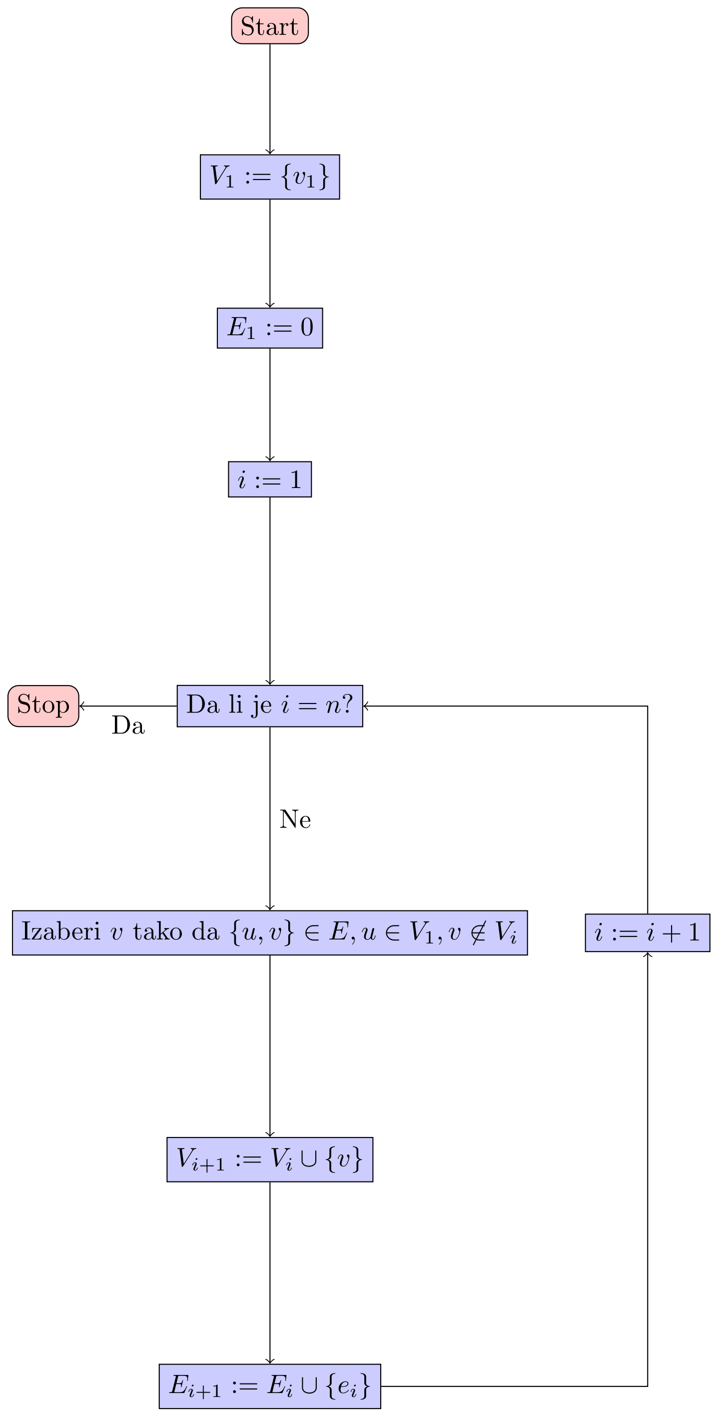
Matematicari su na razne nacine resili problem konstrukcije pokrivajuceg stabla, koristeci veliki broj raznovrsnih algoritama, od kojih cemo mi spomenuti nekoliko.

Algoritam 1 Neka je $G = (V, E)$ povezan graf, gde je $V = \{v_1, ..., v_n\}$. Prvi algoritam koji ćemo predstaviti prikazan je na dijagramu ispod. U prvom koraku se bira proizvoljan čvor v_1. U svakom narednom koraku, podgrafu se dodaje jedan novi čvor koji nije prethodno izabran i za koji postoji grana u grafu koja je incidentna sa tim novim čvorom i jednim već izabranim čvorom. U podgraf se dodaje ta grana. Kako se u svakom koraku dodaje jedna grana i jedan čvor, algoritam staje nakon što je posle prvog koraka izvršeno još $n - 1$ koraka algoritma (što kontroliše brojač i). Pokrivajuće stablo grafa je (V_n, E_n).

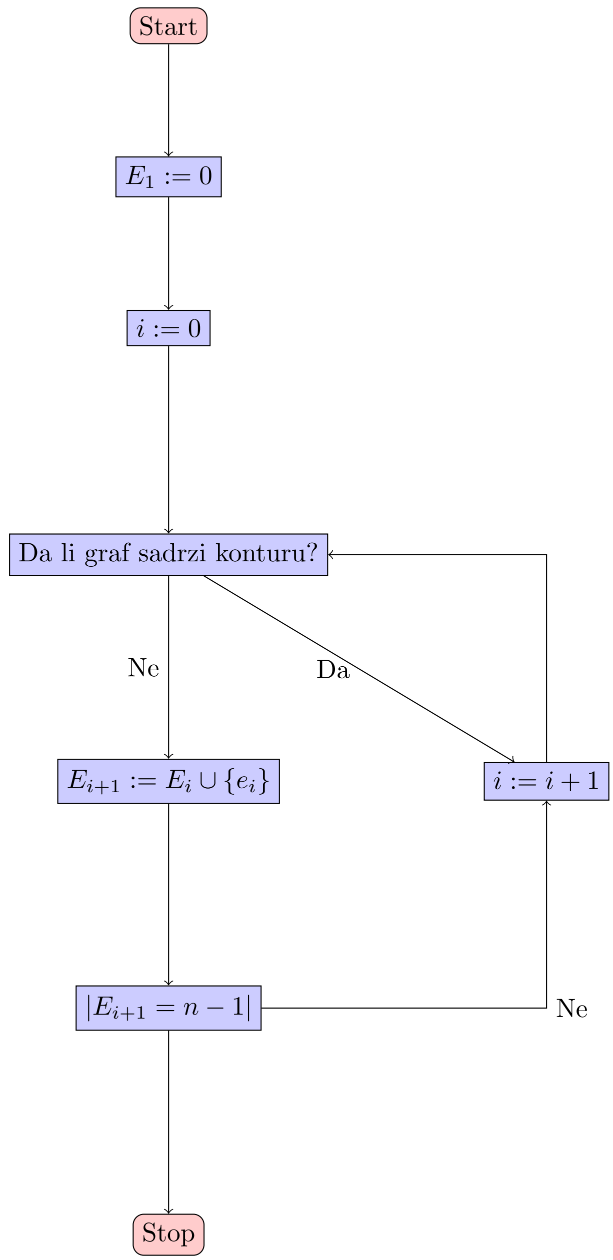
Algoritam 2 Neka je $G = (V, E)$ povezan graf, gde je $V = \{v_1, ..., v_n\}$ i neka su grane proizvoljno uredene u niz

$$(e_1, e_2, ..., e_m).$$

Algoritam za odreivanje pokrivajućeg stabla dat je na dijagramu ispod. U prvom koraku, podgraf sadrži samo granu e_1. Svaki sledeći korak prvo proverava da li naredna grana pravi konturu dodavanjem u prethodno konstruisani podgraf. Ako ne pravi, onda se ta grana dodaje podgrafu, inače algoritam prelazi na proveru naredne grane u nizu. Algoritam staje u trenutku kada je izabrano $n - 1$ grana. Tada je pokrivajuvajuće stablo (V, E_j), gde je $|E_j| = n - 1$, za neko $j \in \{1, ..., m\}$.


Start
$V_1 := \{v_1\}$
$E_1 := 0$
$i := 1$
Da li je $i = n$?
Stop
Da
Ne
Izaberi v tako da $\{u, v\} \in E, u \in V_1, v \notin V_i$
$i := i + 1$
$V_{i+1} := V_i \cup \{v\}$
$E_{i+1} := E_i \cup \{e_i\}$

Algoritam 1


Start
$E_1 := 0$
$i := 0$
Da li graf sadrzi konturu?
Ne
Da
$E_{i+1} := E_i \cup \{e_i\}$
$i := i + 1$
$|E_{i+1} = n - 1|$
Ne
Stop

Algoritam 2

4 Priferov niz

Jedan od dokaza za odredjivanje broja oznacenih stabala je Priferov niz. Najbolje bi bilo da dokaz oslikamo primerom.

Zadatak 1. *Za $n = 2$ imamo jedno, a za $n = 3$ imamo 3 razlicita oznacena stabla.*

Proof. Dokaz da Priferov niz odredjuje broj oznacenih stabala zasniva se na principu bijekcije gde se svakom oznacenom stablu sa n cvorova pridruzuje Priferov niz definisan na sledeci nacin:

$$(p_1, p_2, ...p_n - 2)1 \leq p_i \leq n1 \leq i \leq n - 2$$

□

Neka je $n \geq 2$. broj razlicitih oznacenih stabala sa cvorovima $1, 2, ..., n$ jednak je n^{n-2}

Proof. Ako je $n = 2$, imamo jedno označeno stablo i tvrdjenje važi. Posmatraćemo sada $n \geq 3$ i pokazaćemo dva podtvrdjenja: (i) svakom stablu sa čvorovima $\{1, \ldots, n\}$ možemo na jedinstven način pridružiti Prüferov niz $(p_1, \ldots, p_{n-2})$, koji čine $n-2$ cela broja iz skupa $\{1, \ldots, n\}$ (koja se mogu ponavljati); (ii) svaki niz $(p_1, \ldots, p_{n-2})$ sa osobinom $\{p_1, \ldots, p_{n-2}\} \subseteq \{1, \ldots, n\}$ je Prüferov niz nekog stabla sa n cvorova. □

Algoritam za odredjivanje Pruferovog koda

Algorithm 1 Odredjvanje Pruferovog koda za stablo

Require: Stablo sa n čvorova
Ensure: Pruferov kod kao niz od $n - 2$ elemenata
0: Inicijalizuj prazan niz $PruferCode$
0: **while** broj čvorova > 2 **do**
0: Pronadji list (čvor sa stepenom 1) sa najmanjim indeksom
0: Dodaj njegovog suseda u $PruferCode$
0: Ukloni list iz stabla i smanji stepen njegovog suseda za 1
0: **end while**
0: **return** $PruferCode$ =0

Primer

Razmotrimo stablo sa čvorovima:

$$1 \text{ povezan sa } 2, 3, 4; \quad 3 \text{ povezan sa } 5.$$

Koraci algoritma:

- Prvi list sa najmanjim indeksom je 2, njegov sused 1 se dodaje u kod.
- Uklanja se 2, sledeći list je 3, njegov sused 1 se dodaje.
- Nastavlja se sve dok se ne dobije niz od $n-2$ elemenata.

Pruferov kod za ovo stablo je:

$$[1, 1, 3].$$

Primer 2

Razmotrite stablo sa sledećim čvorovima i granama:

$$1 \text{ povezan sa } 2, 3, 4; \quad 3 \text{ povezan sa } 5; \quad 5 \text{ povezan sa } 6.$$

Koraci algoritma:

1. Početni list sa najmanjim indeksom je 2. Njegov sused 1 se dodaje u Pruferov kod.
2. Nakon uklanjanja 2, sledeći list je 4. Njegov sused 1 se dodaje u Pruferov kod.
3. Sledeći list je 6. Njegov sused 5 se dodaje u Pruferov kod.
4. Sledeći list je 5. Njegov sused 3 se dodaje u Pruferov kod.
5. Ostaju čvorovi 1 i 3, algoritam se zaustavlja.

Pruferov kod za ovo stablo je:

$$[1, 1, 5, 3].$$

4.1 Primer stabla dobijenog Prüferovim kodom

Razmotrimo Prüferov kod:

$$P = (4, 4, 5, 5)$$

[?]

4.1.1 Rekonstrukcija stabla

Rešenje. Postupak za rekonstrukciju stabla iz Prüferovog koda:

1. Započnemo sa nizom svih čvorova u grafu. Broj čvorova je $n = k + 2$, gde je k dužina Prüferovog koda. U ovom primeru, $k = 4$, pa je $n = 6$. Čvorovi su $\{1, 2, 3, 4, 5, 6\}$.

2. Brojimo pojavljivanja svakogvora u Prüferovom kodu:

$$P = (4, 4, 5, 5)$$

vorovi 4 i 5 se pojavljuju dva puta, dok ostali vorovi $\{1, 2, 3, 6\}$ nemaju pojavljivanja.

3. Iterativno povezujemo vor sa najmanjim stepenom (koji se ne pojavljuje u kodu) sa prvim vorom iz Prüferovog niza:

   - Spojimo 1 (najmanji čvor koji nije u Prüferovom kodu) sa 4. Uklonimo 4 iz niza P, ostaje $(4, 5, 5)$.
   - Spojimo 2 sa 4. Uklonimo 4, ostaje $(5, 5)$.
   - Spojimo 3 sa 5. Uklonimo 5, ostaje (5).
   - Spojimo 6 sa 5. Uklonimo 5, niz je prazan.

4. Kada Prüferov niz postane prazan, poslednji preostali čvorovi $\{5, 6\}$ su povezani.

4.1.2 Rezultat

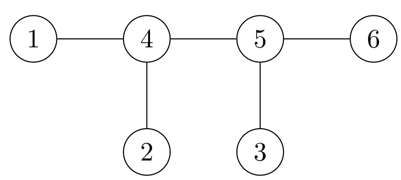
Dobijeno stablo je:

$$E = \{(1, 4), (2, 4), (3, 5), (5, 6)\}$$

[?] □

4.2 Ilustracija

Grafički prikaz stabla:



4.3 Zaključak

Prüferov kod pruža elegantan način za reprezentaciju stabla i njegovu rekonstrukciju. U ovom primeru, kod $P = (4, 4, 5, 5)$ generiše stablo sa $n = 6$ čvorova i granama $E = \{(1, 4), (2, 4), (3, 5), (5, 6)\}$.

5 Težinski graf

5.1 Uvod

Težinski graf je proširenje osnovnog pojma grafa u kojem svaka ivica ima pridruženu težinu. Težine su brojevi koji predstavljaju neku osobinu ivice, poput dužine, troška, vremena ili kapaciteta. Težinski grafovi se često koriste za modelovanje situacija u kojima veze izmeu čvorova imaju kvantitativne vrednosti.

5.2 Definicija

Težinski graf G je uredjena trojka:

$$G = (V, E, w),$$

gde su:

- V: Neprazan skup čvorova (čvorovi grafa), tj. $V = \{v_1, v_2, \ldots, v_n\}$,
- E: Skup ivica, gde:
  - Za *neusmeren graf*, $E \subseteq \binom{V}{2} \cup \{\{v, v\} \mid v \in V\}$, tj. skup **neuredjenih parova** čvorova, sa mogućim petljama,
  - Za *usmereni graf*, $E \subseteq V \times V$, tj. skup **uredjenih parova** čvorova,
- $w : E \to \mathbb{R}$: Funkcija težina koja svakoj ivici $e \in E$ pridružuje realan broj $w(e)$, nazvan težina ivice.

5.3 Osobine težinskog grafa:

1. **Težine ivica:** Svaka ivica $e \in E$ ima pridruženu težinu $w(e)$, koja može biti:
   - *Pozitivna* (npr. udaljenost, kapacitet),
   - *Negativna* (npr. gubitak),
   - *Nula* (neutralna veza).
2. **Vrste težinskih grafova:**
   - *Neusmereni težinski graf:* Ivica $\{u, v\}$ ima istu težinu bez obzira na smer prelaska.
   - *Usmereni težinski graf:* Ivica (u, v) ima težinu $w(u, v)$, koja se razlikuje od težine $w(v, u)$ (ako postoji).
3. **Petlje:** Težinski graf može dozvoliti **petlje**, tj. ivice koje povezuju čvor sa samim sobom:

   $$e = \{v, v\} \quad \text{(za neusmeren graf)} \quad \text{ili} \quad e = (v, v) \quad \text{(za usmeren graf)}.$$

   Funkcija težina w pridružuje težinu i ovim ivicama.
4. **Specifične težine:** Težine mogu imati specifična značenja zavisno od problema koji se modeluje (npr. trošak prelaska, kapacitet kanala, vreme potrebno za prelazak).
5. **Posebni slučajevi:** Ako su sve težine jednake (npr. $w(e) = 1$ za sve $e \in E$), težinski graf se svodi na običan graf (neusmereni ili usmereni).

5.4 Primer težinskog grafa

Neusmereni težinski graf:

- Skup čvorova: $V = \{A, B, C, D\}$,
- Skup ivica: $E = \{\{A, B\}, \{A, C\}, \{B, D\}, \{C, D\}\}$,
- Funkcija težina:

$$w(\{A, B\}) = 3,\ w(\{A, C\}) = 5,\ w(\{B, D\}) = 2,\ w(\{C, D\}) = 4.$$

Usmereni težinski graf:
Funkcija težina za usmerene ivice:

$$w(A, B) = 3,\ w(B, A) = 1,\ w(A, C) = 5,\ w(C, D) = 4,\ w(D, A) = 2,$$

5.5 Primene težinskih grafova

- **Udaljenosti i putevi:** Koriste se za rešavanje problema poput najkraćeg puta (algoritmi Dijkstre, Bellman-Ford, Floyd-Warshall).
- **Optimizacija troškova:** Modelovanje troškova prelaska izmedju čvorova (npr. mreže puteva, transportni sistemi).
- **Maksimalni protok:** Modelovanje kapaciteta izmedju čvorova u mrežama.
- **Analiza mreža:** Težinski grafovi koriste se za analizu društvenih, komunikacionih i energetskih mreža.

6 Najpoznatiji algoritmi za Minimalno Pokrivajuće Stablo (MST)

6.1 Kruskalov Algoritam

Kruskalov algoritam koristi metod povezivanja komponenti uz pomoć ivica u grafu. Počinje sa sortiranjem svih ivica po težini, a zatim povezuje čvorove tako da nikada ne formira cikluse.

Pseudokod:

1. Sortiraj sve ivice u rastućem redosledu po težini.
2. Kreiraj skupove za svaki čvor.
3. Za svaku ivicu (u, v):
   - Ako u i v nisu u istom skupu, dodaj ivicu u MST.
   - Spoji u i v u isti skup.
4. Nastavi dok ne obuhvatiš $|V| - 1$ ivica.

6.2 Primov Algoritam

Primov algoritam koristi metod proširivanja stabla. Počinje sa bilo kojim čvorom i dodaje ivice prema najmanjoj težini dok ne obuhvati sve čvorove.

Pseudokod:

1. Izaberi proizvoljan čvor kao početni.
2. Dodaj sve ivice koje izlaze iz početnog čvora u prioritetni red.
3. Ukloni ivicu sa najmanjom težinom iz reda i dodaj je u MST.
4. Dodaj sve nove ivice koje izlaze iz novog čvora.
5. Nastavi dok svi čvorovi nisu u stablu.

7 Algoritmi za pronalazak najkraceg puta u grafu

Zadatak 2. *Jedan od najpoznatijih algoritama za pronalaženje najkraćeg puta u grafu je Dijkstrin algoritam. Primenjuje se u više oblasti poput robotike, telekomunikacija, navigacionim sistemima i razvoju igara. Njegov nedostatak je što radi samo sa granama koje imaju nenegativnu težinu.*

Rešenje.

- Primer Java algoritma koji od polaznog čvora traži najkraće rastojanje do svakog čvora:

```
public Map<Integer, Integer> shortestPath(int n,
    List<List<Integer>> edges, int src) {
      // graf predstavljamo pomocu liste
      Map<Integer, List<int[]>> adj = new HashMap<>();
      for (int i = 0; i < n; i++) {
          adj.put(i, new ArrayList<>());
      }
      // popunjavamo je sa susednim cvorovima
      for (List<Integer> edge : edges) {
          int s = edge.get(0), d = edge.get(1), weight =
              edge.get(2);
          adj.get(s).add(new int[]{d, weight});
      }

      Map<Integer, Integer> shortest = new HashMap<>();
      PriorityQueue<int[]> minHeap = new
          PriorityQueue<>(Comparator.comparingInt(a -> a[0]));

      // pocetak pretrage po sirini
      minHeap.add(new int[]{0, src});

      while (!minHeap.isEmpty()) {
          int[] current = minHeap.poll();
          int w1 = current[0];
```

```
            int n1 = current[1];

            if (shortest.containsKey(n1)) continue;
            shortest.put(n1, w1);

            for (int[] neighbor : adj.get(n1)) {
                int n2 = neighbor[0];
                int w2 = neighbor[1];
                if (!shortest.containsKey(n2)) {
                    minHeap.add(new int[]{w1 + w2, n2});
                }
            }
        }

        // za cvorove do kojih se ne moze doci vracamo -1

        for (int i = 0; i < n; i++) {
            if (!shortest.containsKey(i)) {
                shortest.put(i, -1);
            }
        }

        return shortest;
    }
```

□

Zadatak 3. *Jedan od najefikasnijih algoritama za pronalaženje najkraćeg puta u grafu je A* algoritam. Koristi heuristiku kako bi usmerio pretragu ka cilju, čime smanjuje broj čvorova koje je potrebno obraditi. Primenjuje se u oblastima poput veštačke inteligencije, robotike, navigacionih sistema i razvoja video igara. Njegova prednost je brzina, ali zahteva pažljivo odabranu heuristiku kako bi dao najbolje rezultate.*

Rešenje. •

```
public static List<Node> aStar(Node start, Node goal) {
        PriorityQueue<Node> openSet = new PriorityQueue<>(); //
            otvoreni skup cvorova
        Set<Node> closedSet = new HashSet<>(); // zatvoreni skup
            cvorova

        start.gCost = 0;
        openSet.add(start);

        while (!openSet.isEmpty()) {
            Node current = openSet.poll(); // uzimamo najjeftiniji
                cvor

            // ako smo stigli do cilja, rekonstruisi put
            if (current == goal) {
```

```
                return reconstructPath(goal);
            }

            closedSet.add(current);

            // obilazimo susede trenutnog cvora
            for (Edge edge : current.neighbors) {
                Node neighbor = edge.target;
                if (closedSet.contains(neighbor)) continue; //
                    preskoci obradjene

                double tentativeGCost = current.gCost + edge.weight;

                if (tentativeGCost < neighbor.gCost) {
                    neighbor.gCost = tentativeGCost;
                    neighbor.hCost = heuristic(neighbor, goal);
                    neighbor.parent = current; // postavljamo
                        roditelja za rekonstrukciju

                    if (!openSet.contains(neighbor)) {
                        openSet.add(neighbor);
                    }
                }
            }

        }

        return null; // ako nije pronadjen put
    }
```

□

Zadatak 4. *Algoritam koji se koristi za pronalaženje najkraćeg puta u grafu sa negativnim težinama grana je Bellman-Ford algoritam. Omogućava izračunavanje najkraćih puteva od izvora do svih ostalih čvorova i detektuje negativne cikluse u grafu. Primenjuje se u oblastima poput ekonomije, mrežnog rutiranja i optimizacije. Njegov nedostatak je što je sporiji u odnosu na druge algoritme.*

Rešenje.

```
public static boolean bellmanFord(List<Edge> edges, int vertices, int
     source) {
   double[] distance = new double[vertices];
   int[] predecessor = new int[vertices];

   Arrays.fill(distance, Double.MAX_VALUE); // postavljamo sve
       udaljenosti na beskonacno
   Arrays.fill(predecessor, -1); // postavljamo sve roditelje na -1
       (nema roditelja)
   distance[source] = 0; // udaljenost izvora od samog sebe je 0
```

```java
// relaksacija svih grana
for (int i = 0; i < vertices - 1; i++) {
    for (Edge edge : edges) {
        if (distance[edge.source] != Double.MAX_VALUE &&
            distance[edge.source] + edge.weight <
            distance[edge.destination]) {
           distance[edge.destination] = distance[edge.source] +
               edge.weight;
           predecessor[edge.destination] = edge.source; //
               postavljamo roditelja
        }
    }
}

for (Edge edge : edges) {
    if (distance[edge.source] != Double.MAX_VALUE &&
        distance[edge.source] + edge.weight <
        distance[edge.destination]) {
        System.out.println("graf sadrzi negativni ciklus.");
        return false;
    }
}

// stampanje rezultata
System.out.println("najkrace udaljenosti od izvora:");
for (int i = 0; i < vertices; i++) {
    System.out.println("cvor " + i + ": " + (distance[i] ==
        Double.MAX_VALUE ? "beskonacno" : distance[i]));
}

// rekonstrukcija puteva
System.out.println("najkraci putevi:");
for (int i = 0; i < vertices; i++) {
    if (i != source) {
        System.out.print("put do cvora " + i + ": ");
        printPath(predecessor, i); // printPath funkciju treba
            implementirati, izostavljeno je zbog nepovezanosti sa
            algoritmom
        System.out.println();
    }
}

return true;
```

□