zadatak8

Grupa 4

Decembar 2024

1 Povezanost grafova

Pri rešavanju realnih programskih problema cesto se javlja potreba za kretanjem kroz graf duž njegovih ivica. Zbog toga uvodimo pojam šetnje po grafu.

Definicija 1. *Neka je $G = (V, E, \psi)$ multigraf. Neka su $e_1, \dots, e_n \in E$ i $v_0, \dots, v_n \in V$ proizvoljne grane i cvorovi sa osobinom $\psi(e_i) = \{v_{i-1}, v_i\}$, za svako $i \in \{1, \dots, n\}$ Tada niz:*

$$v_0 e_1 v_1 e_2 \dots e_n v_n$$

nazivamo $v_0 v_n$-šetnjom dužine n u grafu G između cvorova v_0 i v_n. cvorove v_0 i v_n nazivamo krajnjim cvorovima šetnje

Posebni slucajevi šetnje:

1. ***Staza*** *- Šetnja koja nema ponavljanje grana, tj. ako za sve $i, j \in \{1, \dots, n\}$ sa osobinom $i \neq j$ važi $e_i \neq e_j$.*

2. ***Put*** *- Šetnja koja nema ponavljanje cvorova, tj. ako za sve $i, j \in \{0, 1, \dots, n\}$ sa osobinom $i \neq j$ važi $v_i \neq v_j$ (osim eventualno $v_0 = v_n$)*

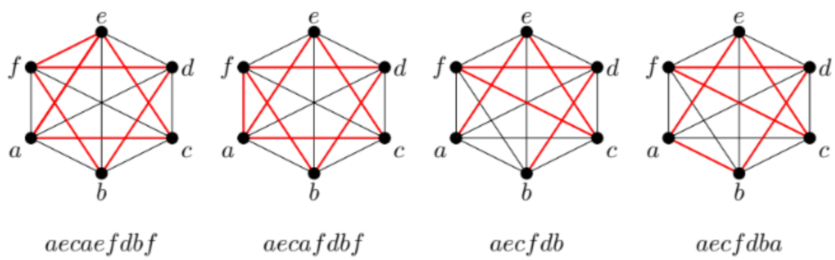
Ako je graf G prost, šetnju ćemo zapisati skraćeno kao:

$$v_0 v_1 \dots v_n$$

—

Ako su krajnji cvorovi šetnje jednaki, tj. $v_0 = v_n$, tada definišemo:

- Šetnja dužine bar jedan je **zatvorena**
- Zatvorena staza se naziva **kružna**
- Zatvoren put se naziva **kontura**



Slika 1: Graficki prikaz redom: šetnje, staze, puta, konture

Teorema 1. *Ako u grafu postoji uv-šetnja (staza), onda postoji i uv-put.*

Dokaz. Neka postoji uv-šetnja:

$$u = v_0 v_1 \dots v_n = v,$$

gde su $v_0 = u$ i $v_n = v$. Ova šetnja može sadržati ponovljene cvorove. Ako se neki cvor v_i ponavlja, možemo razdvojiti šetnju na dva dela:

- Prvi deo od u do v_i,
- Drugi deo od v_i do v.

Uklanjanjem ponovljenog dela između dva pojavljivanja v_i, dobijamo kraću šetnju. Ponavljanjem ovog postupka, dolazimo do uv-puta, tj. šetnje bez ponovljenih cvorova. □

2 Povezani grafovi

Neka je $G = (u, v, \psi)$ multigraf. Kažemo da su cvorovi u i v povezani ako je:

- $u = v$ ili
- $u \neq v$

i postoji uv-put u G

Kažemo da je graf G povezan akko $|V| = 1$ ili za svako $u, v \in V$ važi da su u i v povezani. Ako neki cvorovi nisu medjusobno povezani graf G je nepovezan i može se razložiti na povezane komponente.

Takodje se moze definisati kao:

Za cvorove $u, v \in V$ važi: u i v su povezani ako postoji podskup ivica $P \subseteq E$ takav da P cini put izmedju u i v.

Dokaz da je relacija "je povezan sa"relacija ekvivalencije

Neka $G = (V, E)$ bude graf, gde je V skup cvorova, a E skup ivica. Relacija "je povezan sa"je definisana kao:

$$a \sim b \iff \text{Postoji put izmedju cvorova } a \text{ i } b$$

Za dokazivanje da je ova relacija ekvivalencija, potrebno je da proverimo tri svojstva:

1. **Refleksivnost:** Svaki cvor je povezan sa samim sobom.

   Za svaki cvor $a \in V$, postoji put izmedju a i a (u grafovima to može biti petlja). Dakle, $a \sim a$.

   $$a \sim a \quad \text{(jer postoji put od } a \text{ do } a\text{, tj. petlja na } a\text{)}$$

   Dakle, relacija je refleksivna.

2. **Simetricnost:** Ako je cvor a povezan sa cvorom b, onda je i cvor b povezan sa cvorom a.

   Ako postoji put od a do b, postoji i obrnuti put od b do a, što znaci da je $b \sim a$. Dakle, relacija je simetricna.

   $$a \sim b \implies b \sim a \quad \text{(jer ako postoji put od } a \text{ do } b\text{, postoji i put od } b \text{ do } a\text{)}$$

3. **Tranzitivnost:** Ako je cvor a povezan sa cvorom b, a cvor b povezan sa cvorom c, onda je i cvor a povezan sa cvorom c.

   Ako postoji put od a do b i put od b do c, onda postoji put od a do c. Dakle, relacija je tranzitivna.

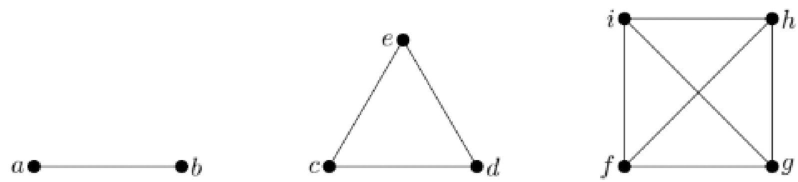
   $$a \sim b \text{ i } b \sim c \implies a \sim c \quad \text{(jer ako postoji put od } a \text{ do } b \text{ i od } b \text{ do } c\text{, postoji put od } a \text{ do } c\text{)}$$

Pošto relacija "je povezan sažadovoljava sve tri osobine (refleksivnost, simetricnost i tranzitivnost), zakljucujemo da je to **relacija ekvivalencije**.

3 Komponenta Povezanosti Grafa

3.1 Uvod

Neka je G = (a, b, c, d, e, f, g, h, i, E) graf na slici:



Broj komponenti povezanosti datog grafa je ω(G) = 3. Komponente povezanosti su podgrafovi koji su indukovoani skupovima cvorova a,b, c,d,e, i f,g,h,i.

3.2 Teorema

Multigraf G = (V, E, ψ) je povezan akko ω(G) = 1.

Dokaz. G je povezan akko za svaka dva cvora u, v $\in$ V postoji uv-put u G. To dalje vazi akko svi cvorovi pripadaju istoj klasi ekvivalencije u odnosu na relaciju "je povezan sa", što važi akko ω(G) = 1. □

3.3 Definicija

Neka je G = (V, E, ψ) sa osobinom ω(G) k $\geq$ 1.
cvor v $\in$ V je razdelni (artikulacioni) ako je ω(G - v) > k.
Grana e $\in$ E je razdelna (most) ako je ω(G - e) > k.

4 Prost graf sa n cvorova i manje od $n-1$ grana nije povezan

4.1 Formulacija tvrdnje

Neka je $G = (V, E)$ prost graf sa $n = |V|$ cvorova i $m = |E|$ grana. Ako važi $m < n - 1$, tada graf G nije povezan. [?]

Dokaz. Dokazaćemo tvrdnju kontrapozicijom. Pretpostavimo da je graf G povezan. Tada, prema definiciji povezanosti, postoji put između svakog para cvorova u G.

Za svaki povezani graf sa n cvorova, minimalan broj grana potreban da ostane povezan jednak je $n - 1$. Ovo se može dokazati sledećim argumentima:

1. Ako G ima $n-1$ grana i nema ciklusa, onda je G stablo. Znamo da je stablo povezan graf sa minimalnim brojem grana, tj. $n-1$.

2. Ako G ima manje od $n-1$ grana ($m < n-1$), tada mora postojati najmanje dva cvora između kojih ne postoji put. Ovo se može dokazati posmatranjem da dodavanjem grana do $n-1$ možemo dobiti stablo koje je povezano.

Stoga, ako $m < n-1$, graf G ne može biti povezan. Ovo završava dokaz tvrdnje. □

5 Primer nepovezanog prostog grafa

5.1 Uvod

U ovom dokumentu prikazujemo konstrukciju prostog grafa sa n cvorova i najmanje $n-1$ grana, koji nije povezan. Takav graf se može lako konstruisati kreiranjem više komponenti koje su drveća ili disjunktni skupovi cvorova povezanih sa minimalnim brojem grana.

5.2 Algoritam

- Sledeći Java program omogućava unos broja cvorova n i generiše nepovezan graf sa tacno $n-1$ grana:

```java
// Java program za generisanje nepovezanog grafa sa n cvorova i
    n-1 grana
import java.util.*;

public class DisconnectedGraph {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Unesite broj cvorova (n): ");
        int n = scanner.nextInt();

        if (n < 2) {
            System.out.println("Graf sa manje od 2 cvora ne moze
                biti nepovezan sa n-1 grana.");
            return;
        }

        System.out.println("Generisanje nepovezanog grafa sa " + n
            + " cvorova i " + (n - 1) + " grana:");

        // Kreiranje grana tako da graf ima tacno n-1 grana i nije
            povezan
        List<int[]> edges = generateDisconnectedGraph(n);
```

```
        printEdges(edges);
    }

    // Metoda za generisanje nepovezanog grafa sa n cvorova i n-1
        grana
    public static List<int[]> generateDisconnectedGraph(int n) {
        List<int[]> edges = new ArrayList<>();

        // Prva komponenta: drvo sa k cvorova i k-1 grana
        int k = n / 2;
        for (int i = 1; i < k; i++) {
            edges.add(new int[]{i, i + 1});
        }

        // Druga komponenta: preostali cvorovi sa jednom granicnom
            vezom
        for (int i = k + 1; i <= n; i++) {
            edges.add(new int[]{k, i});
        }

        return edges;
    }

    // Metoda za ispis grana
    public static void printEdges(List<int[]> edges) {
        for (int[] edge : edges) {
            System.out.println(edge[0] + " - " + edge[1]);
        }
    }
}
```

5.3 Objašnjenje i primer rezultata

Algoritam deli cvorove na dve komponente: jednu koja formira povezano drvo sa k cvorova i $k-1$ grana, i drugu koja povezuje preostale cvorove na jedan cvor iz prve komponente, stvarajući tako nepovezan graf. Na primer, za $n = 6$, dobijamo sledeći skup grana:

$$1-2,\ 2-3,\ 3-4,\ 4-5,\ 4-6$$

Rezultirajući graf sadrži dve komponente i nije povezan.

5.4 Brisanje grane iz konture prostog grafa

Neka je G = (V, E) povezan i neka je C kontura u grafu G. Ako je e grana konture, onda je G - e povezan.

Dokaz. Izaberimo proizvoljno dva cvora $u, v \in V$. Kako je G povezan, postoji uv-put

$$P = uv_1 \dots v_i v_{i+1} \dots v_{n-1} v.$$

Imamo dve mogućnosti:

(i) Ako e ne pripada uv-putu, onda je P uv-put u $G - e$.

(ii) Ako e pripada uv-putu, onda možemo pretpostaviti da je to grana $\{v_i, v_{i+1}\}$ i da je kontura C oblika

$$C = v_i u_1 u_2 \dots u_k v_i.$$

To znaci da u grafu $G - e$ postoji put Q od v_i do v_{i+1}:

$$Q = v_i u_1 u_2 \dots u_k v_{i+1}.$$

Onda je

$$P_2 = uv_1 \dots v_{i-1} Q v_{i+2} \dots v_{n-1} v,$$

staza u grafu $G - e$ od u do v. Prema Teoremi 61, $G - e$ je povezan. Znaci, za svaka dva cvora u grafu $G - e$ postoji put koji ih povezuje.

□

6 Primeri

6.1 Primer 1: Provera da li je graf povezan

6.1.1 Kod

```
# Klasa za neusmeren graf
class Graph:
    def __init__(self, vertices):
        self.V = vertices # Broj cvorova
        self.graph = {v: [] for v in range(vertices)} # Susedna lista za
            cvorove

    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u) # Dodajemo granu u oba smera

    def is_connected(self):
        visited = [False] * self.V # Pratimo poseene cvorove
        self.dfs(0, visited) # Pokreemo DFS od cvora 0
        return all(visited) # Graf je povezan ako su svi cvorovi poseeni

    def dfs(self, v, visited):
        visited[v] = True # Obeleavamo cvor kao poseen
        for neighbor in self.graph[v]: # Obilazimo susede
            if not visited[neighbor]:
                self.dfs(neighbor, visited)

# Primer  korienja
g = Graph(4)
g.add_edge(0, 1)
g.add_edge(1, 2)
g.add_edge(2, 3)

print("Da li je graf povezan?", g.is_connected()) # Ocekivani ishod: True
```

6.1.2 Objašnjenje

- `add_edge`: Dodaje granu između dva cvora.
- `is_connected`: Proverava povezanost koristeći pretragu u dubinu (DFS).
- `dfs`: Rekurzivna funkcija za obilazak suseda cvora.

6.2 Primer 2: Pronalaženje komponenti povezanosti

6.2.1 Kod

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = {v: [] for v in range(vertices)}

    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)

    def connected_components(self):
        visited = [False] * self.V
        components = []

        for v in range(self.V): # Proveravamo svaki cvor
            if not visited[v]:
                component = []
                self.dfs(v, visited, component)
                components.append(component)

        return components

    def dfs(self, v, visited, component):
        visited[v] = True
        component.append(v) # Dodajemo cvor u trenutnu komponentu
        for neighbor in self.graph[v]:
            if not visited[neighbor]:
                self.dfs(neighbor, visited, component)

# Primer koriscenja
g = Graph(6)
g.add_edge(0, 1)
g.add_edge(1, 2)
g.add_edge(3, 4)

print("Komponente povezanosti:", g.connected_components())
# Ocekivani ishod: [[0, 1, 2], [3, 4], [5]]
```

6.2.2 Objašnjenje

- Metoda `connected_components` pronalazi sve komponente povezanosti grafa.
- Pomoćna metoda `dfs` koristi se za obilazak suseda u trenutnoj komponenti.

6.3 Primer 3: Provera svojstva povezanog grafa nakon brisanja grane

6.3.1 Kod

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = {v: [] for v in range(vertices)}

    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)

    def remove_edge(self, u, v):
        self.graph[u].remove(v)
        self.graph[v].remove(u)

    def is_connected(self):
        visited = [False] * self.V
        self.dfs(0, visited)
        return all(visited)

    def dfs(self, v, visited):
        visited[v] = True
        for neighbor in self.graph[v]:
            if not visited[neighbor]:
                self.dfs(neighbor, visited)

# Primer  korienja
g = Graph(4)
g.add_edge(0, 1)
g.add_edge(1, 2)
g.add_edge(2, 3)
g.add_edge(3, 0) # Graf je ciklican

print("Pre uklanjanja grane, graf povezan:", g.is_connected()) # True

g.remove_edge(3, 0)
print("Nakon uklanjanja grane, graf povezan:", g.is_connected()) # True
```

6.3.2 Objašnjenje

- Metoda `remove_edge` briše granu između dva cvora.
- Kod proverava povezanost grafa pre i nakon brisanja grane.