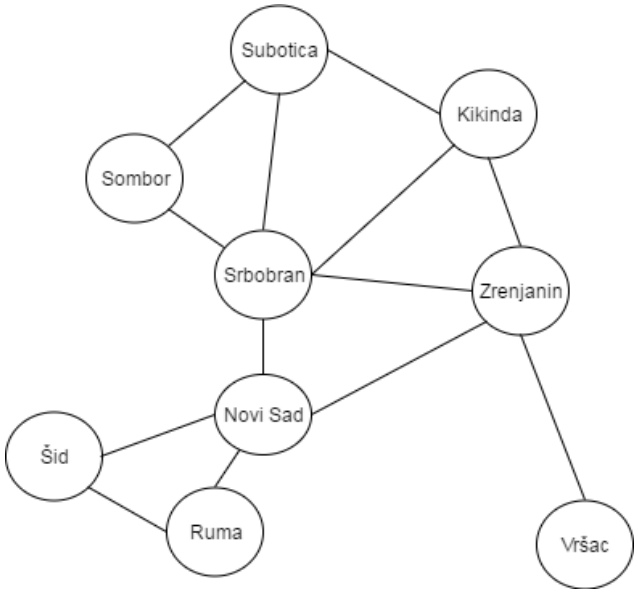
Grafovi

Autori:
Milan Segedinac
Goran Savić

1. Grafovi

Graf je struktura koju čine *čvorovi* (*nodes*) grafa i *veze* (*edges*) između čvorova. Grafovi su veoma važne strukture u računarstvu jer se pomoću njih mogu predstaviti bilo kakve *relacije među entitetima*. Na primer, putna mreža se može predstaviti kao graf u kome su čvorovi gradovi, a veze putevi koji povezuju gradove; komunikacione mreže mogu da se posmatraju kao grafovi u kojima su čvorovi komunikacioni uređaji, a veze komunikacione linije; društvena mreža se može predstaviti kao graf u kom su čvorovi individue, a prijateljstva veze između njih; neuronska mreža je graf u kome su čvorovi ćelije (neuroni), a veze sinapse. Klase u programu, tabele u bazi podataka i dokumenti koji se razmenjuju preko interneta mogu se posmatrati kao grafovi. Pojednostavljena mreža puteva Vojvodine predstavljena pomoću grafa data je na slici ispod. Rešavanje problema iz programiranja često se svodi na predstavljanje problema pomoću grafa i primenu nekog od standardnih algoritama za pronalazak rešenja.



Slika – Mreža puteva Vojvodine predstavljena pomoću grafa

Vrste grafova

U zavisnosti od problema koji rešavamo možemo koristiti različite vrste grafova:

- *Usmereni* (*directed*) i *neusmereni* (*undirected*) grafovi – grafovi u kojima veze između čvorova imaju smer smatramo usmerenim. Tipično, mreža međugradskih puteva je neusmeren graf jer međugradski putevi najčešće imaju trake u oba smera. Mreža ulica u gradu bi mogla da se predstavi usmerenim grafom jer ulice mogu da budu i jednosmerne. Ako ne naglasimo drugačije, smatramo da je graf usmeren.
- *Težinski* (*weighted*) i *bestežinski* (*unweighted*) grafovi – u težinskom grafu svakoj vezi je dodeljena numerička vrednost (težina). Mreža puteva mogla bi se predstaviti pomoću težinskog grafa u kome težine predstavljaju udaljenost gradova. Ako ne naglasimo drugačije smatramo da je graf bestežinski.

- *Labelirani* (*labeled*) i *nelabelirani* (*unlabeled*) grafovi – ukoliko je svakoj vezi dodeljen naziv ili identifikator, graf smatramo labeliranim. Mreža ulica u gradu bi se mogla predstaviti pomoću labeliranog grafa u kom su labele nazivi ulica. Ako ne naglasimo drugačije smatramo da je graf nelabeliran.

Predstavljanje grafova

Graf možemo predstaviti tako što će svaki čvor imati *listu susednih čvorova* (*adjacency list*, u literaturi se koristi i termin *lista povezanosti*). Obratite pažnju da je suštinska razlika između ovako predstavljenog grafa i spregnute liste u tome što u grafu čvor može da ima *više od jednog susednog* čvora, a u linked listi čvor *ima najviše* jedan susedni čvor. U stvari, spregnuta lista je specijalni slučaj grafa. Takođe, stablo je specijalni slučaj usmerenog grafa u kome svaki čvor može da ima najviše jednog roditelja.

Ukoliko za čvorove *x* i *y* u grafu postoji veza *(x,y)* onda će čvor *x* u svojoj listi susednih čvorova imati čvor *y*. Implementacija čvora koja omogućuje ovakvu reprezentaciju grafa prikazana je listingom ispod.

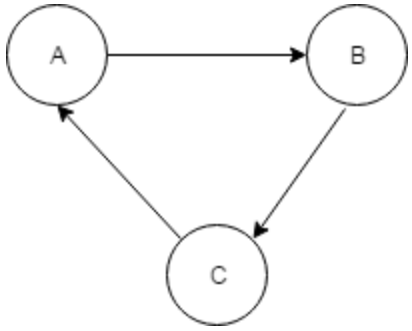
```
public class Node<T> {

	public Node(T value) {
		super();
		this.value = value;
	}

	T value;

	List<Node<T>> adjacencyList = new ArrayList<Node<T>>();

	public void addAdjacentNode(Node<T> adjacentNode){
		this.adjacencyList.add(adjacentNode);
	}

	public boolean removeAdjacentNode(Node<T> adjacentNode){
		if(adjacencyList.contains(adjacentNode)){
			this.adjacencyList.remove(adjacentNode);
			return true;
		}
		else {
			return false;
		}
	}

	public boolean adjacentTo(Node<T> adjacentNode){
		return adjacencyList.contains(adjacentNode);
	}
}
```

Listing – predstavljanje grafa listama susedstva za čvorove (deo klase `Node`)

Klasa `Node` prestavlja čvor u grafu. Susedni čvorovi predstavljeni su listom `adjacencyList`. Dodavanje veze između čvorova realizuje se dodavanje čvora u listu `adjacencyList` pozivom metode `addAdjacentNode`. Uklanjanje se realizuje pozivom metode `removeAdjacentNode`. Metoda `adjacentTo` proverava da li postoji veza između čvorova.

Prolazak kroz graf

Kako bismo mogli da ispišemo sve vrednosti čvorova u grafu? Mogli bismo da krenemo od proizvoljnog čvora, ispišemo njegovu vrednost i pristupimo njegovim susednim čvorovima. Za svaki od susednih čvorova ispišemo vrednost i pristupimo njegovim susednim čvorovima i tako dok ne dođemo do čvorova koji nemaju susedne čvorove. (Ovakav algoritam smo koristili za ispis stabla.) Šta bi se desilo ako bismo ovaj algoritam upotrebili za ispis vrednosti čvorova u grafu sa slike ispod?



Slika – Ciklični graf

Krenuli bismo od čvora A, ispisalo bi se „A“ i prešlo na njegove susedne čvorove. Njegov jedini susedni čvor je B, pristupili bismo njemu i ispisala bi se njegova vrednost („B“). Zatim bismo pristupili susednim čvorovima čvora B, a to je jedino čvor C. Bilo bi ispisano „C“ pa bismo pristupili susednim čvorovima čvora C, a to je jedino A. Bilo bi ispisano ponovo „A“, pa ponovo „B“, pa „C“,... Naš program bi ušao u beskonačnu petlju koja je posledica činjenice da je graf *cikličan*.

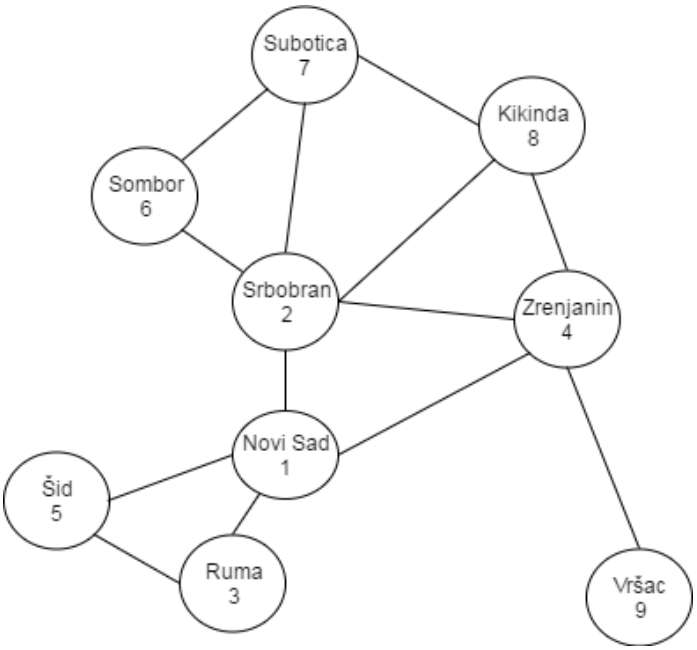
Jedan od osnovnih problema u radu sa grafovima je *prolazak kroz graf* (*graph traversal*) – kako posetiti svaki čvor i svaku vezu u grafu na *sistematičan* način, što znači na *efikasan* (izbeći ponovne posete istih čvorova i veza) i *korektan* (tako da znamo da nismo neki čvor ili vezu izostavili) način. Jedan od algoritama prolaska kroz graf bazira se na tome da svaki čvor obeležimo jednim od tri moguće oznake:

- *Neotkriven* – čvor u inicijalnom stanju
- *Otkriven* – čvor je pronađen ali još uvek nisu proverene sve njegove veze.
- *Obrađen* – proverene su sve veze ovog čvora

Pored toga nam treba lista svih čvorova koju su označeni kao *otkriveni*. Na početku samo jedan (početni) čvor smatramo da je otkriven. Da bismo obradili čvor, moramo da evaluiramo svaku vezu koja vodi iz njega. Ukoliko veza vodi u *neotkriven* čvor, taj čvor ćemo prevesti u stanje *otkriven* i dodati ga u listu otkrivenih čvorova. Ukoliko ova veza vodi u *obrađen* čvor, ignorisaćemo je. Ignorisaćemo i veze koje vode u *otkrivene* čvorove zato što su ovi čvorovi već dodati u listu otkrivenih. Nakon što evaluiramo sve veze za posmatrani čvor, taj čvor označavamo kao *obrađen* i uklanjamo ga iz liste otkrivenih čvorova. Redosled u kom ćemo evaluirati veze za čvor odrediće koji algoritam prolaska kroz graf ćemo koristiti.

Algoritam prvi u širinu

U algoritmu *prvi u širinu* (*breadth-first*) čim posetimo čvor dodaćemo sve njegove susedne čvorove u listu otkrivenih, a posećeni čvor ćemo označiti da je obrađen. Ako bismo ovaj algoritam koristili za prolazak kroz graf gradova iz ove lekcije, redosled obrađivanja čvorova bio bi kao na slici ispod.



Slika – BFS prolazak kroz graf gradova

1. U listu otkrivenih čvorova dodali bismo prvi čvor (Novi Sad), pa bi lista bila [Novi Sad]
2. Krenuli bismo od čvora Novi Sad, sve njegove susedne čvorove označili kao otkrivene, a Novi Sad kao obrađen. Lista otkrivenih čvorova bi bila [Srbobran, Ruma, Šid, Zrenjanin]
3. Prvi otkriveni čvor iz liste (Srbobran) bismo obradili, sve njegove susedne čvorove preveli u stanje otkriven. Lista otkrivenih bi bila [Ruma, Šid, Zrenjanin, Sombor, Subotica, Kikinda]
4. Preuzimanje i obrađivanje čvorovi bi se vršilo sa početka liste, a novi čvorovi bi se dodavali na kraj. Čvorovi koji su već posećeni ne bi bili ponovo obrađivani (bili bi ignorisani).