Elektrotehnički fakultet Sarajevo
Odsjek za računarstvo i informatiku
Predmet: Algoritmi i strukture podataka

Pripremna zadaća za vježbu 10

Cilj vježbe je upoznavanje sa različitim metodama predstavljanja grafova u memoriji, njihovim prednostima i nedostacima, te osnovnim algoritmima za grafove: obilazak grafa po dubini (DFS) i širini (BFS).

Rok je objavljen u informacionom sistemu zamger.

Zadatak 1.

Napisati apstraktnu klasu UsmjereniGraf koja služi za razvoj aplikacija za rad sa usmjerenim grafovima. U slučaju da vam je potreban neusmjereni graf, možete ga simulirati preko usmjerenog grafa tako što dodate grane u oba pravca. S druge strane, klasom NeusmjereniGraf ne bi bilo moguće simulirati usmjereni graf.

Klasa UsmjereniGraf treba da podržava sljedeće metode:

- Konstruktor koji prima jedan cjelobrojni parametar: broj čvorova. Ovaj konstruktor kreira graf sa datim brojem čvorova i bez grana (grane između pojedinih čvorova mogu biti dodate naknadno odgovarajućom metodom).
- Po potrebi destruktor, konstruktor kopije i operator dodjele.
- Metoda **dajBrojCvorova** vraća vrijednost atributa za broj čvorova.
- Metoda **postaviBrojCvorova** treba omogućiti povećanje broja čvorova, a ako se njome pokuša smanjiti broj čvorova metoda treba baciti izuzetak.[1]
- Metoda **dodajGranu** omogućuje da se između dva čvora doda grana. Parametri ove metode su: polazni čvor, dolazni čvor i težina grane. Polazni i dolazni čvor su vrijednosti tipa int (čvorovi su označeni brojevima 0, 1, 2, …) a težina je vrijednost tipa float. Navođenje parametra težina nije obavezno (ako se ne navede taj parametar potrebno je postaviti težinu na nulu).
- Metoda **obrisiGranu** uklanja granu između dva čvora, parametri su polazni i dolazni čvor.
- Metoda **postaviTezinuGrane** omogućuje promjenu težine već postojeće grane. Parametri su isti kao za dodajGranu.

---

1 Problem sa smanjivanjem broja čvorova leži u tome što su čvorovi označeni redom brojevima 0, 1, 2, … pa se smanjivanjem broja čvorova najprije postavlja pitanje koji od ovih čvorova je izbačen, a i kada se to odredi (recimo proslijeđivanjem parametra funkciji) onda je pitanje šta uraditi sa oznakama drugih čvorova, šta sa granama koje su povezane sa izbačenim čvorom i slično. Zbog toga ćemo se ograničiti na primjene koje ne zahtijevaju smanjenje broja čvorova. U slučaju, da je takvo nešto potrebno, možemo jednostavno ne koristiti određeni broj čvorova.

- Metoda **dajTezinuGrane** omogućuje da se očita težina grane, parametri su polazni i dolazni čvor.
- Metoda **postojiGrana** vraća bool vrijednost koja označava da li postoji grana između dva čvora, parametri su polazni i dolazni čvor.
- Sve navedene metode trebaju bacati izuzetke za besmislene vrijednosti ili u slučaju kada čvor ili grana ne postoji (a potrebna je).[2]

U algoritmima za rad sa grafovima često se javlja potreba da označavamo čvorove i grane nekim oznakama (brojevi, slova, stringovi itd.) Osim toga ova mogućnost je generalno korisna stvar. Da bismo podržali korištenje oznaka, u klasu UsmjereniGraf dodaćemo i sljedeće metode:

- **postaviOznakuCvora** dodjeljuje datom čvoru određenu oznaku. Parametri su broj čvora i oznaka.
- **dajOznakuCvora** vraća oznaku čvora datog kao parametar.
- Slične su metode **postaviOznakuGrane** i **dajOznakuGrane**, a granu kao i ranije označavamo polaznim i dolaznim čvorom.

Da bismo omogućili da oznaka čvora ili grane bude bilo šta, klasa UsmjereniGraf mora biti *generička klasa*. Koristićemo *isti tip* za oznaku čvora i oznaku grane (nazovimo ga TipOznake).

U algoritmima koji rade sa grafovima, pristup čvorovima i granama grafa je česta operacija, pa je jako bitno da je ta operacija efikasna. Također, često su zahtijevane operacije kao što su iteriranje kroz sve grane grafa ili sve grane grafa koje vode iz jednog čvora i slično.

Zbog toga ćemo uvesti još dvije klase: *Grana* i *Cvor* su pristupne klase za pojedinačne grane/čvorove grafa. Vi trebate napisati implementacije metoda ovih klasa i po potrebi dodati konstruktore, destruktore, konstruktore kopije, operatore dodjele i/ili druge funkcije koje vam budu potrebne.

- U klasu UsmjereniGraf (i njene izvedene klase) dodati metode *dajGranu* i *dajCvor* koje vraćaju instance ovih pristupnih klasa koje omogućavaju promjenu odgovarajućeg čvora odnosno ivice.
- Klasa Grana treba posjedovati metode: **dajTezinu, postaviTezinu, dajOznaku** i **postaviOznaku** čije je značenje očigledno.
- Također treba posjedovati metode **dajPolazniCvor** i **dajDolazniCvor** koje trebaju vraćati objekat tipa Cvor.
- Što se tiče klase Cvor, pored metoda **dajOznaku** i **postaviOznaku**, klasa Cvor treba posjedovati konstruktor koji prima *pokazivač na UsmjereniGraf* i redni broj čvora, te metodu **dajRedniBroj** koja vraća redni broj čvora.

Obratite pažnju da metode koje vraćaju Granu/Cvor ne vraćaju pokazivač niti referencu na Granu/Cvor. Pa ipak ove klase trebaju omogućiti promjenu vrijednosti u grafu npr:

---

2 Primijetimo da se u ovoj implemetaciji ne dopuštaju grafovi koji imaju više od jedne grane između dva čvora.

```
UsmjereniGraf graf(5);
graf.dodajGranu(0,1);
…
graf.dajGranu(0,1).postaviOznaku(15);
cout << graf.dajOznakuGrane(0,1); // 15
```

Razmislite na koji način organizovati atribute svih klasa iz zadatka kako bi ovo bilo moguće! Razlog za ovo je sljedeći: ima li ikakvog smisla grana ili čvor bez svog grafa??? Klase Grana i Cvor su samo pogodniji način korištenja metoda koje već posjeduje klasa UsmjereniGraf.

Također ćemo uvesti i novu klasu GranaIterator koja treba da omogući prolazak kroz grane grafa na efikasan način. Radi lakše implementacije dati ćemo kompletan prototip klase GranaIterator:

```
template <typename TipOznake>
class GranaIterator {
 public:
  Grana<TipOznake> operator*();
  bool operator==(const GranaIterator &iter) const;
  bool operator!=(const GranaIterator &iter) const;
  GranaIterator& operator++();
  GranaIterator operator++(int);
};
```

Ovdje GranaIterator prolazi kroz sve grane grafa, ali pri tome preskače nepostojeće grane. Metoda operator* vraća objekat tipa klase Grana (koju ste ranije razvili) koji odgovara grani na koju trenutno pokazuje iterator. operator== omogućuje poređenje da li dva iteratora pokazuju na istu granu, a operator ++ prelazi na sljedeću granu. (Data je prefiksna i postfiksna varijanta operatora.) Kada se dođe do posljednje grane u grafu operator++ treba baciti izuzetak

Ranije dato zaglavlje klase proširite po potrebi sa konstruktorima, destruktorima, konstruktorima kopije i operatorima dodjele i/ili drugim funkcijama koje vam budu potrebne. U klasu UsmjereniGraf dodajte metode ***dajGranePocetak*** i ***dajGraneKraj*** koje vraćaju iteratore ovog tipa koji omogućavaju iteriranje kroz sve grane grafa.

Zadatak 2.

Potrebno je napraviti i implementaciju usmjerenog grafa pomoću matrice susjedstva. U vašem programu trebate imati klasu MatricaGraf izvedenu iz ranije date klase UsmjereniGraf. Zatim implementirati metode za obilazak po širini i dubini (bfs i dfs 0.8b).

Za iteriranje kroz grane grafa implementiranog sa matricom susjedstva koristeći klasu GranaIterator očekivana kompleksnost je $O(n * n)$.

Primjer korištenja opisanih pristupnih klasa, kao i iteratorske klase za grane grafa lijepo se vidi iz sljedećeg jednostavnog programa:

```
int main()
{
  try {
    UsmjereniGraf<bool> *g = new MatricaGraf<bool>(3);
    g->dodajGranu(0, 1, 2.5);
    g->dodajGranu(1, 0, 1.2);
    g->dodajGranu(1, 2, 0.1);
    g->dodajGranu(0, 0, 3.14);
    for (GranaIterator<bool> iter = g->dajGranePocetak();
         iter != g->dajGraneKraj(); ++iter)
      cout << (*iter).dajPolazniCvor().dajRedniBroj() << " "
           << (*iter).dajDolazniCvor().dajRedniBroj() << " "
          << (*iter).dajTezinu() << endl;
    delete g;
  } catch (const char izuzetak[]) {
    cout << izuzetak << endl;
  }
  return 0;
}
```

Ispis ovog programa može biti:

```
0 0 3.14
0 1 2.5
1 0 1.2
1 2 0.1
```

Ili bilo koja permutacija ove tri linije zavisno od tačnog načina implemetacije. (Naravno u testovima je stavljen samo jedan odgovor kao tačan, ali ako je kod vas drugačiji redoslijed iteriranja to se neće smatrati za grešku.)

Primijetite da se koristi deklaracija `UsmjereniGraf<bool>` iako oznake čvorova i grane nisu uopšte potrebne za ovu primjenu. Da se ne bi dodatno komplikovala implementacija, mi ćemo koristiti najjednostavniji tip podataka (`bool` ili `char`) za oznake ondje gdje one nisu potrebne kako bismo minimiizirali iskorištenje memorije.

Zbog cikličnih zavisnosti klasa *Grana*, *Cvor* i *GranaIterator*, pogodno je koristiti *forward* deklaracije ovih klasa u odgovarajućim zaglavljima umjesto uključivanja zaglavlja drugih klasa.

```
template <typename TipOznake>
class Grana;
template <typename TipOznake>
class Cvor;

template <typename TipOznake>
class GranaIterator;
```