Pismeni ispit

Objektno orijentirano programiranje

17. lipnja 2019.

Upute

Pismeni se ispit piše 120 minuta. Studenti smiju koristiti isključivo materijale s predavanja i vježbi te C++ referencu te Python dokumentaciju. Niti jedan drugi oblik komunikacije nije dopušten te će se svako nepridržavanje pravila kažnjavati udaljavanjem s ispita i prijavom uredu pročelnika. Nakon što završite s pisanjem ispita, rješenja smjestite po zadacima u mapu `PREZIME_IME_PISMENI3`, tu mapu zipajte u datoteku `PREZIME_IME_PISMENI3.zip` te ju pošaljite na oop@mathos.hr s naslovom e-mail poruke: "PREZIME IME: OOP PISMENI 3".

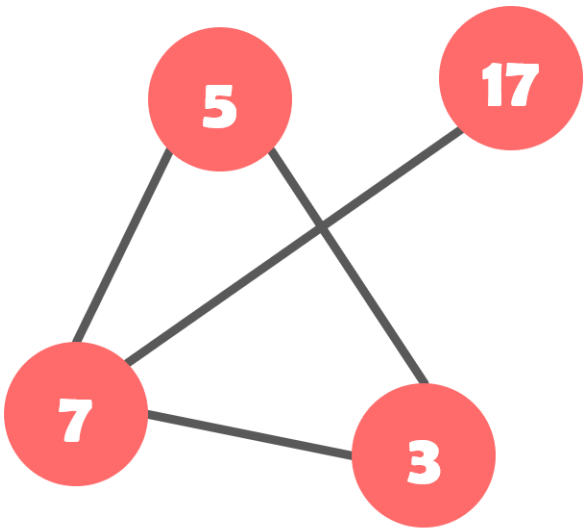
Zadatak 1. (C++) [40 bodova]

Graf je matematički koncept koji se definira kao uređeni par skupa vrhova[1] i bridova

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

Svaki vrh $u \in V$ iz skupa vrhova može *biti susjedan* ili s nijednim, ili s više ostalih vrhova iz V. Kada kažemo da je vrh $u \in V$ susjedan s vrhom $v \in V$, to znači da postoji brid $e \in E$ u skupu bridova, pri čemu je relacija "biti susjedan" *simetrična*: ako je $u \in V$ susjedan $v \in V$, onda je i $v \in V$ susjedan $u \in V$.

Graf vizualiziramo kao na Slika 1,



Slika 1 Ilustracija grafa

[1] Drugi naziv za vrh je i *čvor.*

pri čemu u konkretnom primjeru vrijedi da je:

$$V = \{3, 5, 7, 17\}$$

$$E = \{\{3, 5\}, \{3, 7\}, \{5, 7\}, \{7, 17\}\}.$$

Jedan od načina na koji možemo implementirati graf je *lista susjedstva*. Za svaki vrh v u grafu, njegove susjede spremamo u listu, a isto tako sve vrhove u grafu spremamo u listu.

Deklaracija klase koja služi za pohranu vrha je sljedeća:

```
template <typename T>
class Vertex {
private:
    T data;
    list<Vertex*> neighbors;
public:
    Vertex(T);
    void addNeighbor(Vertex*);
    bool isNeighbor(Vertex*);
    void removeNeighbor(Vertex*);
    ~Vertex();
};
```

Vidljivo je da instanca ove klase enkapsulira neki podatak tipa `T` imena `data` (npr. na Slika 1 to bi bio broj), a isto tako, ova klasa enkapsulira listu vrhova `neighbors` koji su susjedi tome vrhu.

Konstuktor `Vertex(T)` instancira novi objekt koji predstavlja vrh pri čemu se enkapsulirani član `data` inicijalizira s onime što je prosljeđeno u konstruktoru, dok lista susjedstva ostaje prazna. Metoda `addNeighbor` dodaje novi vrh u listu susjedstva, a isto tako tome novom vrhu dodaje vrh koji je pozvao metodu u listu susjedstva, ali samo ukoliko ti vrhovi međusobno nisu u listi susjedstva, što se može provjeriti uz pomoć metode `isNeighbor`. Metoda `removeNeighbor` uklanja susjedstvo između vrhova.

Deklaracija klase koja predstavlja graf je:

```
template <typename T>
class Graph {
private:
    list<Vertex<T>*> vertices;
public:
    Graph();
    Graph(const Graph&);
    void addVertex(Vertex<T>*);
    void removeVertex(Vertex<T>*);
    ~Graph();
};
```

Lista `vertices` predstavlja listu svih vrhova u grafu. Metoda `addVertex` dodaje novi vrh u listu vrhova, dok metoda `removeVertex` uklanja vrh iz liste vrhova (pri uklanjanju vrhova je potrebno pripaziti na susjedstvo!).

Unutar `main`-a instancirajte graf koji je dan na Slika 1 te ga ispišite na sljedeći način pri čemu je potrebno koristiti preopterećeni operator `operator<<` i za objekt klase `Graph<T>` i za objekte klase `Vertex<T>`:

```
Vertex at 0xffffcb60 with data 3 and neighbors: 5       7
Vertex at 0xffffcb78 with data 5 and neighbors: 3       7
Vertex at 0xffffcb90 with data 7 and neighbors: 3       5       17
Vertex at 0xffffcba8 with data 17 and neighbors: 7
```

Implementirajte metodu

```
template <typename T>
unordered_map<Vertex<T>*, Vertex<T>*> traversal(const Graph<T>& g, Vertex<T>* root)
```

koja će biti implementacija obilaska grafa po širini (bread-first-search). Konkretno, metoda će sustavno otkrivati sve vrhove koji su prvi susjedi vrha `root` koji do tada nisu otkriveni, zatim sve vrhove koji su udaljeni za dva brida od vrha `root` koji do tada nisu otkriveni, itd… Ova metoda vraća `unordered_map` čiji su elementi parovi pokazivača na vrhove pri čemu je prvi element pokazivač na konkretni čvor u obilasku, a drugi element pokazivač na njegovog prethodnika. Primjer ispisa pri pozivu

```
auto predecessors = traversal(g, v);
```

za prethodno instancirani graf `g`, pri čemu je `v` instanca klase `Vertex<T>` čiji je atribut `data` jednak `17`, treba biti:

```
Predecessor of 5 in traversal with root 17: 7
Predecessor of 3 in traversal with root 17: 7
Predecessor of 17 in traversal with root 17: nullptr
Predecessor of 7 in traversal with root 17: 17
```

Zadatak 2. (C++) [35 bodova]

Klasa `Person` deklarirana je s:

```
class Person {
protected:
    string firstName, lastName;
    unsigned int age;

public:
    Person();
```

```
    Person(const Person& p);
    Person(const string& firstName, const string& lastName, unsigned int age);

    void setAge(unsigned int age);

    Person& operator=(const Person& p);

    virtual ostream& printPerson(ostream& os) const;
};
```

Definirajte odgovarajuće članove ove klase. Preopteretite odgovarajući operator tako da za primjer instance:

```
Person p1{ "Marko", "Maric", 25 };
```

ispis bude:

```
>>> Marko Maric is 25 years old
```

Metoda `printPerson` treba se pozvati unutar `operator<<`. Ova je metoda virtualna kako bismo ju, nasljeđujući klasu `Person`, dalje modificirali u nasljeđenim klasama.

`operator=` potrebno je preopteretiti tako da on pridružuje neku instancu klase pri čemu se svi atributi kopiraju po vrijednosti. Ukoliko npr. pozovemo:

```
Person p1{ "Marko", "Maric", 25 };
Person p4 = p1;
p4.setAge(24);
```

atribut `p4.age` se mora se razlikovati od atributa `p1.age`.

Zatim definirajte klasu `Employee` koja nasljeđuje klasu `Person`, a enkapsulira dodatno `string company` koji predstavlja ime kompanije. Deklaracija te klase je:

```
class Employee: public Person {
protected:
    string company;
public:
    Employee();
    Employee(const Employee& employee);
    Employee(const string& firstName, const string& lastName, unsigned int age,
const string& company);
    ostream& printPerson(ostream& os) const;
   };
```

Na kraju, inicijalizirajte polje `people` koje se sastoji od pokazivača na instance tipa `Person`, istovremeno unutar njega smjestite barem 3 pokazivača na objekte od kojih barem 1 nije pokazivač na objekt bazne klase. Prođite kroz to polje te ispišite njegove članove. Primjer ispisa je:

```
Marko Maric is 25 years old
Ljubo Ljubic is 26 years old
Ivan Ivic is 27 years old
Zdravko Zdravković is 30 years old and works in HP
```

Napomena: Primijetite da su se objekti ispisali u ovisnosti o tome jesu li instance bazne ili nasljeđene klase.

Zadatak 3. (Python) [25 bodova]

Koristeći klasu `queue.LifoQueue` implementirajte klasu `Queue` koja treba predstavljati FIFO strukturu podataka. Deklaracija klase je sljedeća:

```python
class Queue:

    def __init__(self):
        pass

    def enqueue(self, item):
        pass

    def dequeue(self):
        pass

    def is_empty(self):
        pass
```

Nakon što definirate klasu `Queue`, instancirajte ju, instancu popunite s deset vrijednosti, zatim istovremeno ispisujte elemente reda uz pomoć odgovarajućeg operatora te izbacujte elemente sve dok ta instanca nije prazna.