JMBAG **IME I PREZIME**

Algoritmi i strukture podataka – završni ispit

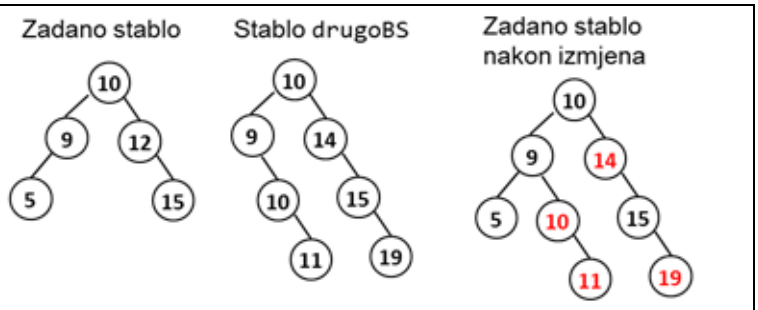
28. siječnja 2022.

Ispit donosi maksimalno 45 bodova. Ovaj primjerak ispita predajte s upisanim imenom, prezimenom i JMBAG-om.

Zadatak 1. (9 bodova)

Zadan je razred BStablo kojim se implementira binarno stablo:

```
template <typename T> class BStablo {
public:
   BStablo() : korijen(nullptr) {}
   void unos(const T& elem);
   …
protected:
   struct Cvor {
      T elem;
       shared_ptr<Cvor> lijevo, desno;
       Cvor(const T &novi) : …
   }
   shared_ptr<Cvor> korijen;
   void unos(shared_ptr<Cvor>& cvor,
      const T& elem);
};
```



a) Potrebno je napisati javni funkcijski član `zamijeniVecim` razreda `BStablo`, koji će vrijednost čvora zadanog stabla zamijeniti s vrijednošću čvora koji je na istom mjestu u drugom stablu (`drugoBS`) i to samo ako je vrijednost čvora stabla `drugoBS` veća od vrijednosti čvora zadanog stabla. U slučaju da na nekom mjestu u zadanom stablu ne postoji čvor, a na istom mjestu u stablu drugoBS čvor postoji, tada treba stvoriti novi čvor u zadanom stablu i u njega upisati vrijednost čvora na istom mjestu u stablu drugoBS (koristiti funkcijski član `unos`). Funkcijski član `zamijeniVecim` treba vratiti ukupan broj zamijenjenih i novoupisanih vrijednosti u zadanom stablu (za primjer na slici funkcija treba vratiti 4, jer su obavljene jedna zamjena i tri dodavanja novih čvorova u zadano stablo). Funkcijski član `zamijeniVecim` zadan je prototipom:

```
int zamijeniVecim(BStablo<T> drugoBS);
```

Napomene: članske funkcije `unos` (dvije funkcije s modifikatorima vidljivosti `public` i `protected`) već su zadane (vidjeti razred `BStablo`) i ne treba ih implementirati. Članska funkcija `void` **`unos`**`(Cvor**, const T& elem);` alocira memorijski prostor za novi čvor i u njega pohranjuje vrijednost `elem`.
Za realizaciju članske funkcije `zamijeniVecim` dozvoljeno je koristiti i pomoćne funkcijske članove i pomoćne funkcije. Pretpostavite da su definirani operatori usporedbe i operator pridruživanja za tip T.

b) Napišite odsječak glavnog programa u kojemu se poziva funkcijski član `zamijeniVecim`.

Zadatak 2. (8 bodova)

Zadan je niz brojeva:

`9, 0, 8, 4, 1, 3, 4, 2`

Ilustrirajte uzlazno sortiranje algoritmom **Shellsort** s koracima `{5, 2, 1}`. Potrebno je prikazati sadržaj polja nakon svake promjene.

Zadatak 3. (8 bodova)

Zadan je niz brojeva:

`2, 3, 6, 4, 5, 2`

Za zadani niz brojeva prikažite stvaranje gomile **maxheap** pomoću algoritma čije je vrijeme izvođenja **O(n)** (za n članova polja). U svakom koraku algoritma u kojemu se obavlja zamjena trebaju biti označeni članovi polja koji su zamijenjeni te treba prikazati polje nakon svake zamjene.
Počevši od gomile kreirane u prvome dijelu zadatka, prikažite uzlazno sortiranje zadanog niza *heapsortom*, prikazujući svaki korak sortiranja (navedite sadržaj polja nakon svake promjene).

Zadatak 4. (6 bodova)

Potrebno je napisati **rekurzivnu** člansku funkciju zadane klase DGraph koja će, **nakon** što je izvršen Dijkstrin algoritam za početni čvor v0, ispisati najkraći put od čvora v0 do čvora zadanog argumentom funkcije. Funkcija treba imati prototip: void PrintPath(Vertex<int>* v);. Zadane su klase Vertex i DGraph.

```
template<typename T> class Vertex{
   public:
      T item;
      vector<Vertex<T>*> next;
      vector<double> weights;
      Vertex<T>* previous = nullptr;
      double distance = INFINITY;
      bool traversed = false;
      Vertex(T item) {this->item = item;}
};
```

```
class DGraph{
   public:
      Vertex<int>* v0;
      vector<Vertex<int>*> allVertices;
      DGraph(){ /* kôd za kontruktor klase */ }
      void Dijkstra(){ /* implementacija Dijk. Algor, */ }
      void PrintPath(Vertex<int>* v){
       //funkcija koju je potrebno napisati
      }
};
```

Klasa *Vertex* modelira čvor grafa: *next* je vektor susjednih čvorova, *weights* udaljenosti do susjednih čvorova (za zadani čvor, udaljenost do čvora *next[i]* iznosi *weights[i]*), *previous* je pokazivač na prethodni čvor nakon što je obavljen Dijkstrin algoritam, *distance* je najkraća udaljenost do čvora nakon što je obavljen Dijkstrin algoritam, te *traversed* varijabla koja govori je li čvor već obiđen tijekom izvršavanja Dijkstgrinog algoritma. Klasa DGraph modelira graf nad kojim se pokreće Dijkstrin algoritam. *v0* je početni čvor Dijkstrinog algoritma, *allVertices* vektor koji pohranjuje sve čvorove grafa, DGraph() konstruktor, a Dijkstra() implementacija Dijkstrinog algoritma (koju ne treba pisati).

Zadatak 5. (5 bodova)

Izračunati LPS polje koje se koristi u **Knuth-Morris-Pratt** algoritmu za niz AACCBAAACCBAAC**.**

A	A	C	C	B	A	A	A	C	C	B	A	A	C

Zadatak 6. (9 bodova)

Zadana su sučelja koja služe implementaciji tablice raspršenog adresiranja.

```
…
template <typename T, typename K> class IHashableValue {
 public:
   virtual K GetKey() const = 0;
};
template <typename T, typename K> class HashElement {
 public:
   IHashableValue<T, K> *value;
   HashElement *next;
   HashElement(IHashableValue<T, K> *value) { this->value = value; }
};
template <typename T, typename K> class IHash {
 protected:
   size_t size;
   HashElement<T, K> **hash;
 public:
   virtual void Add(IHashableValue<T, K> *element) const = 0;
   virtual IHashableValue<T, K> *Get(K key) const = 0;
};
```

Svaki objekt ili zapis koji se upisuje u tablicu raspršenog adresiranja implementira sučelje `IHashableValue`. Varijabla `HashElement<T, K> **hash` služi pohrani tablice raspršenog adresiranja, a varijabla `size` određuje njenu veličinu.

a) **Potrebno je napisati klasu `HashChaining`** koja implementira sučelje `IHash` tako da koristi tehniku ulančavanja. Klasa `HashChaining` treba imati implementirane metode `Add` i `Get`. Za određivanje adrese pretinca koristi se hash funkcija `HashChangeBaseMethod`.

b) **Potrebno je napisati funkciju raspršenja** `int HashChangeBaseMethod(int key, size_t hashSize);` koja koristi metodu izmjene baze brojanja. Bazu brojanja treba promijeniti iz 10 u 11.

Napomena: *Konstruktor i destruktor nije potrebno implementirati. U funkciji* `HashChangeBaseMethod` *se nakon izmjene baze brojanja obavlja transformacija u raspon adresa hash tablice (hashSize) korištenjem operacije modulo (%).*

Rješenja:

1. zadatak (9 bodova)

```
template <typename T> int BinaryTree<T>::zamijeniVecim(BinaryTree <T> drugoBS) { // public
    return zamijeniVecim(korijen, drugoBS.korijen);
}

template <typename T> int BinaryTree<T>::zamijeniVecim(shared_ptr<Node>& cvor, shared_ptr<Node>
cvor2) { // protected
    if (cvor2) {
        int izmjena = 0;
        if (cvor) {
            if (cvor2->elem > cvor->elem) { // zamijeni vecim
                cvor->elem = cvor2->elem;
                ++izmjena;
            }
        }
        else { // novi cvor
            unos(cvor, cvor2->elem);
            ++izmjena;
        }
        return izmjena + zamijeniVecim(cvor->lijevo, cvor2->lijevo) +
            zamijeniVecim(cvor->desno, cvor2->desno);
    }
    return 0;
}
```

b) Odsječak glavnog progama, npr.:

```
BStablo<int> bs = BStablo<int>();
BStablo<int> drugoBS = BStablo<int>();
std::cout << bs.zamijeniVecim(drugoBS);
```

2. zadatak (8 bodova)

9, 0, 8, 4, 1, 3, 4, 2

Korak = 5	**Korak = 2**	**Korak = 1**
9 0 8 4 1 **9** 4 2	3 0 **3** 4 1 9 4 8	**1** 1 2 4 3 8 4 9
3 0 8 4 1 9 4 2	**2** 0 3 4 1 9 4 8	**0** 1 2 4 3 8 4 9
3 0 8 4 1 9 4 **8**	2 0 3 4 **3** 9 4 8	0 1 2 4 **4** 8 4 9
3 0 **2** 4 1 9 4 8	2 0 **2** 4 3 9 4 8	0 1 2 **3** 4 8 4 9
	1 0 2 4 3 9 4 8	0 1 2 3 4 8 **8** 9
	1 0 2 4 3 9 4 **9**	0 1 2 3 4 **4** 8 9
	1 0 2 4 3 **8** 4 9	

3. zadatak (8 bodova)

Stvaranje gomile *maxheap*:

```
2, 3, 6, 4, 5, 2
2, 5, 6, 4, 3, 2
6, 5, 2, 4, 3, 2
```

Uzlazno sortiranje korištenjem heapsort-a:

```
6, 5, 2, 4, 3, 2
2, 5, 2, 4, 3, 6
5, 4, 2, 2, 3, 6
3, 4, 2, 2, 5, 6
4, 3, 2, 2, 5, 6
2, 3, 2, 4, 5, 6
3, 2, 2, 4, 5, 6
2, 2, 3, 4, 5, 6
2, 2, 3, 4, 5, 6
2, 2, 3, 4, 5, 6
```

4. zadatak (6 bodova)

```
void PrintPath(Vertex<int>* v){
      if(v == nullptr) return;
      PrintPath(v->previous);
      cout << v->item << " ";
   }
```

5. zadatak (5 bodova)

A	A	C	C	B	A	A	A	C	C	B	A	A	C
0	1	0	0	0	1	2	2	3	4	5	6	7	3

6. zadatak (9 bodova)

```
template <typename T, typename K> class HashChaining : public IHash<T, K> {
 private:
   HashElement<T, K> **hash;
   int HashChangeBaseMethod(int key, size_t hashSize) const{
      int newKey = 0;
      int newBase = 1;
      while(key!=0){
         int oldDigit = key%10;
         newKey = newKey + (oldDigit * newBase);
         key = key/10;
         newBase *= 11;
      }
      return newKey%hashSize;
   }

 public:
   virtual void Add(IHashableValue<T, K> *element) const {
      int i = HashChangeBaseMethod(element->GetKey(), this->size);
      HashElement<T, K> *el = new HashElement<T, K>(element);
      el->next = hash[i];
      hash[i] = el;
   }

   virtual IHashableValue<T, K> *Get(K key) const {
      int i = HashChangeBaseMethod(key, this->size);
      HashElement<T, K> *head;
      for (head = hash[i]; head && (head->value->GetKey() != key);
           head = head->next)
         ;
      if (head == nullptr)
         return nullptr;
      return head->value;
   }
};
```