JMBAG **IME I PREZIME**

Algoritmi i strukture podataka – završni ispit

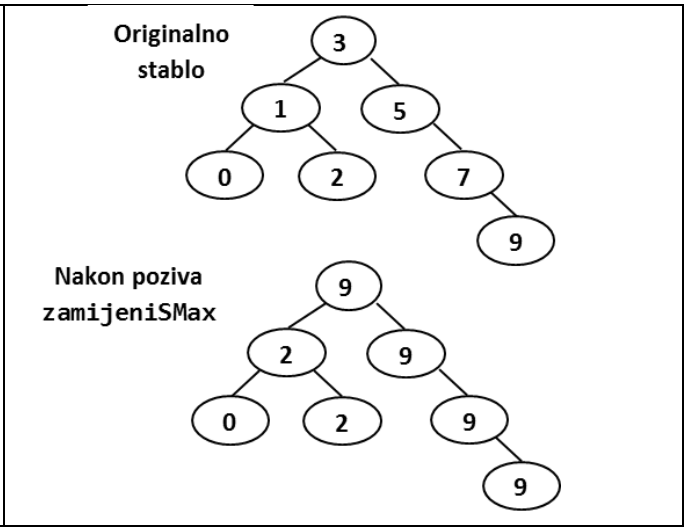
31. siječnja 2020.

Ispit donosi maksimalno 40 bodova. Ovaj primjerak ispita trebate predati s upisanim imenom i prezimenom te JMBAG-om.

Zadatak 1. (8 bodova)

Zadan je razred `BStablo` kojim se implementira binarno stablo:

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



a) Potrebno je napisati javni funkcijski član `zamijeniSMax` razreda `BStablo`, koji će svaki element stabla zamijeniti s najvećim elementom u njegovim podstablima (vidjeti sliku), a zadan je prototipom:

```
void zamijeniSMax();
```

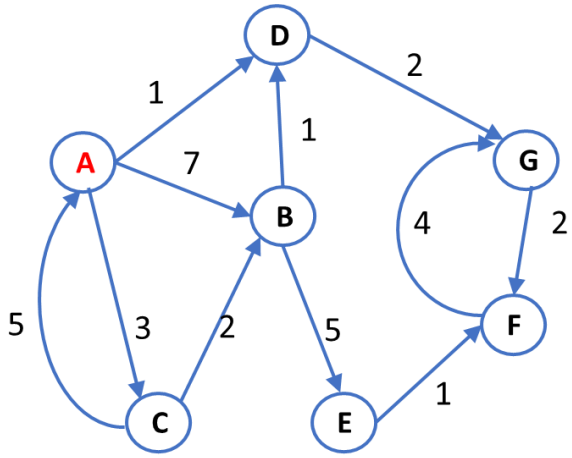
Dozvoljeno je koristiti pomoćni funkcijski član i pomoćne funkcije.

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

c) Odredite složenost vaše implementacije `zamijeniSMax` u O, Ω i, ako je moguće, Θ notaciji.

Zadatak 2. (6 bodova)

Zadan je graf prikazan slikom:



Korištenjem **Dijkstrinog algoritma** odredite najkraću udaljenost od vrha **A** do ostalih vrhova u grafu. Prikažite sadržaj pratećih pomoćnih struktura (vektora prethodni i udaljenost te pomoćnog prioritetnog reda) u svakom koraku algoritma.

Napomena: Za svaki vrh, njemu susjedne vrhove promatramo abecednim redoslijedom.

Zadatak 3. (6 bodova)

Zadan je niz brojeva:

`9, 8, 6, 4, 1, 3, 2`

Za zadani niz brojeva prikažite stvaranje gomile **minheap** 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.

Zadatak 4. (8 bodova)

Zadana su sučelja koja služe implementaciji tablice raspršenog adresiranja.

```
#define c1 0.5
#define c2 0.5
…
template <typename T, typename K> class IHashableValue {
 public:
   virtual K GetKey() const = 0;
};
template <typename T, typename K> class IHash {
 protected:
   size_t size;
   IHashableValue<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 `IHashableValue<T, K> **hash` služi pohrani tablice raspršenog adresiranja, a varijabla `size` određuje njenu veličinu.

Potrebno je napisati klasu `HashQuadraticProbing` koja implementira sučelje `IHash` tako da koristi tehniku otvorenog adresiranja s kvadratnim ispitivanjem. Klasa `HashQuadraticProbing` treba imati implementiran konstruktor (u kojem se inicijalizira tablica raspršenog adresiranja veličine `size`) te metodu `Get`.

Napomena: Metodu `Add` i destruktor nije potrebno implementirati.
Kao funkciju raspršenja potrebno je koristiti unaprijed definiranu funkciju `int HashMultiplicationMethod(int key)`.

Zadatak 5. (6 bodova)

Zadan je niz brojeva:

`4, 1, 12, 7, 11, 10`

Ilustrirajte uzlazno sortiranje algoritmom **Quicksort**. Stožer za Quicksort bira se metodom aproksimacije medijana temeljem prvog, srednjeg i zadnjeg člana, pri čemu vrijedi da je cutoff = 3 nakon čega se sortira bez navođenja koraka. Potrebno je prikazati sadržaj polja nakon svake promjene.

Zadatak 6. (6 bodova)

Algoritam Rabin-Karp, koji služi pronalasku podniza u nizu, koristi u svojem radu izračun sažetka (hash-a) podniza. Napisati funkciju **ReHash** koja se koristi u algoritmu Rabin-Karp i koja temeljem sažetka trenutnog podniza duljine m računa sažetak idućeg podniza duljine m. Funkcija mora imati složenost O(1) i prototip:

```
int ReHash(int oldHash, char leadingDigit, char lastDigit, int mostSigDigitValue);
```

Parametar `oldHash` predstavlja sažetak trenutnog podniza, `leadingDigit` predstavlja prvi lijevi znak trenutnog podniza, a `lastDigit` predstavlja zadnji znak idućeg podniza. Parametar `mostSigDigitValue` predstavlja unaprijed izračunatu težinsku vrijednost s kojom se množi najznačajnija znamenka `leadingDigit`.

Napomena: Kako bi se uspješno izvela funkcija `ReHash` potrebno je koristiti konstante `D` i `PRIM` definirane pretprocesorskim direktivama:

```
#define D 256
#define PRIME 251
```

Rješenja:

1. zadatak (8 bodova)

a)

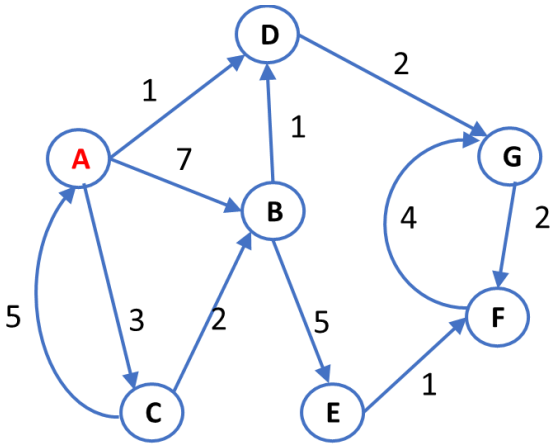
```
template <typename T> void BStablo<T>:: zamijeniSMax() {
	if (korijen != nullptr) zamijeniSMax(korijen);
};

template <typename T> T BStablo<T>:: zamijeniSMax(shared_ptr<Cvor> &cvor) {
	T stari = cvor->elem;
	if (cvor->lijevo && cvor->desno) {
		T l = zamijeniSMax(cvor->lijevo);
		T d = zamijeniSMax (cvor->desno);
		cvor->elem = l > d ? l : d;
	}
	else if (cvor->lijevo) {
		cvor->elem  = zamijeniSMax (cvor->lijevo);
	}
	else if (cvor->desno) {
		cvor->elem = zamijeniSMax(cvor->desno);
	}
	return (cvor->elem > stari) ? cvor->elem : stari;
}
```

b) Odsječak glavnog progama, npr.:

```
BStablo<int> bStablo = BStablo<int>();
bStablo.zamijeniSMax();
```

c) Θ(n), gdje je n broj čvorova u stablu

2. zadatak (6 bodova)



1. Početno stanje

Q **= {0, ∞, ∞, ∞, ∞, ∞, ∞}**

	A	B	C	D	E	F	G
Udaljenost	0	∞	∞	∞	∞	∞	∞
Preth	-	-	-	-	-	-	-

2. Uzimamo vrh A iz *Q* i označavamo da je obiđen.

	A	B	C	D	E	F	G
Udaljenost	0	7	3	1	∞	∞	∞
Preth	-	A	A	A	-	-	-

Q **= {1, 7, 3, ∞, ∞, ∞}**

3. Uzimamo vrh D iz *Q* i označavamo da je obiđen. Podešavamo udaljenosti.

	A	B	C	D	E	F	G
Udaljenost	0	7	3	1	∞	∞	3
Preth	-	A	A	A	-	-	D

Q **= {3(C), 7, 3(G), ∞, ∞}**

4. Uzimamo vrh C iz *Q* i označavamo da je obiđen. Podešavamo udaljenosti.

	A	B	C	D	E	F	G
Udaljenost	0	5	3	1	∞	∞	3
Preth	-	C	A	A	-	-	D

Q **= {3 (G), 5, ∞, ∞}**

5. Uzimamo vrh G iz *Q* i označavamo da je obiđen. Podešavamo udaljenosti.

	A	B	C	D	E	F	G
Udaljenost	0	5	3	1	∞	5	3
Preth	-	C	A	A	-	G	D

Q **= {5 (B), 5 (F), ∞}**

6. Uzimamo vrh B iz *Q* i označavamo da je obiđen.

	A	B	C	D	E	F	G
Udaljenost	0	5	3	1	10	5	3
Preth	-	C	A	A	B	G	D

Q **= {5 (F), 10}**

7. Uzimamo vrh F iz *Q* i označavamo da je obiđen. Podešavamo udaljenosti.

	A	B	C	D	E	F	G
Udaljenost	0	5	3	1	10	5	3
Preth	-	C	A	A	B	G	D

Q **= { 10 }**

8. Uzimamo vrh E iz *Q* i označavamo da je obiđen. Time smo obišli sve vrhove.

	A	B	C	D	E	F	G
Udaljenost	0	5	3	1	10	5	3
Preth	-	C	A	A	B	G	D

Q **= { }**

3. zadatak (6 bodova)

9	8	6	4	1	3	2
9	8	**2**	4	1	3	**6**
9	**1**	2	4	**8**	3	6
1	1	2	4	8	3	6
1	**4**	2	4	8	3	6
1	4	2	**9**	8	3	6

4. zadatak

```
template <typename T, typename K>
class HashQuadraticProbing : public IHash<T, K> {
 private:
   void Empty() {
      size_t i;
      for (i = 0; i < this->size; i++) {
         hash[i] = nullptr;
      }
   }

 public:
   virtual IHashableValue<T, K> *Get(K key) const {
      int address = HashMultiplicationMethod(key);
      int index;
      for (int i = 0; i < this->size; i++) {
         index = (int)fmod((address + c1 * i + c2 * i * i), this->size);
         if (hash[index] == nullptr)
            return nullptr;
         if (hash[index]->GetKey() == key) {
            return hash[index];
         }
      }
      return nullptr;
   }
   HashQuadraticProbing(size_t size) {
      this->size = size;
      this->hash = new IHashableValue<T, K> *[this->size];
      this->Empty();
   }
};
```

5.

4, 1, **12**, 7, 11, **10** odabir stožera/medijan
4, 1, **10**, 7, 11, **12** sortiranje stožera/medijan
4, 1, 11, 7, **10**, 12 sakrivanje stožera 10
4, 1, **7**, **11**, 10, 12 zamjena elemenata na indeksima i=2 i j=3
4, 1, 7, 11, **10**, 12 obnova stožera - i=3 (zamjena vrijednosti 11 i 10)
4, **1**, 7, 10, 11, 12 sortiranje desnog i lijevog dijela Insertion Sortom (cutoff)
1, 4, 7, 10, 11, 12 u lijevom dijelu zamjena 1 i 4, ostalih zamjena nema

6. zadatak

```
   static int ReHash(int oldHash, char leadingDigit, char lastDigit) {
      int oldHashWithoutLeadingDigit = oldHash - leadingDigit * mostSigDigitValue;
      int newHashwithoutLastDigit = oldHashWithoutLeadingDigit * D;
      int newHash = (newHashwithoutLastDigit + lastDigit) % PRIME;
      if (newHash < 0) // in case new hash is negative, covert to a positive num
         newHash += PRIME;
      return newHash;
   }
```