JMBAG **IME I PREZIME**

Algoritmi i strukture podataka – ispit

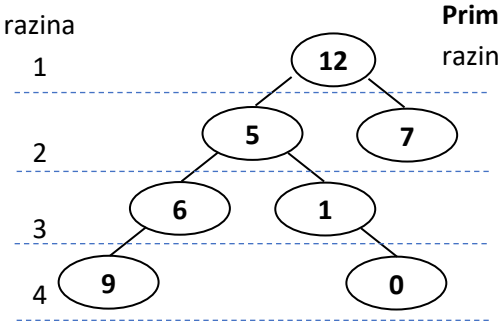
31. kolovoza 2020.

Nije dopušteno korištenje globalnih i statičkih varijabli te naredbe goto. Ispit donosi maksimalno 70 bodova. Ovaj primjerak ispita trebate predati s upisanim imenom i prezimenom te JMBAG-om.

Zadatak 1. (14 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;
    …
};
```



Primjer: Stablo s četiri razine.

Potrebno je napisati javni funkcijski član `maxNaRazini` razreda `BStablo`, koji će vraća postoji li čvor na zadanoj razini te, ako postoji, kolika je najveća vrijednost čvora na zadanoj razini. Član je zadan prototipom

```
bool maxNaRazini(int razina, T* najveci);
```

Dozvoljeno je koristiti pomoćni funkcijski član i pomoćne funkcije. Napišite i odsječak glavnog programa u kojemu se poziva funkcijski član `maxNaRazini`.

Zadatak 2. (14 bodova)

Neka je zadano polje **A** koje se sastoji od **n** pozitivnih cijelih brojeva. Polje je organizirano tako da se na prvih **k** mjesta (**k** nije poznat unaprijed) nalaze neparni brojevi, a na preostalih **n-k** mjesta parni brojevi. Neparni (i parni) brojevi su međusobno poredani tako da element s većim indeksom ima veću ili jednaku vrijednost u odnosu na element s manjim indeksom, a iste parnosti.

Napišite funkciju `traziBroj` koja vraća 1 ako je zadani element x element polja A, odnosno 0 ako nije. Funkcija `traziBroj` treba imati složenost $O(\log_2 n)$, a prototip funkcije treba biti:

```
int traziBroj(int A[], int n, int x);
```

Za rješavanje problema dozvoljeno je koristiti samo pomoćne funkcije koje sami napišete.

Napomena: Primjer polja A je 3, 7, 15, 8, 10, 12, 20, 22.

Zadatak 3. (12 bodova)

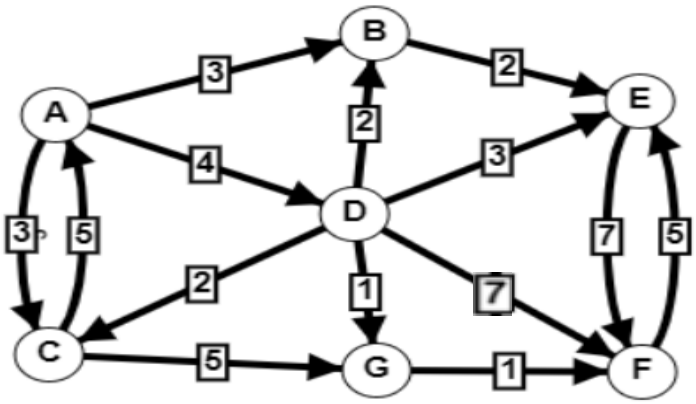
U cjelobrojnom polju pohranjen je sljedeći niz brojeva:

6, 3, 5, 9, 1, 2, 7, 4

Za zadani niz brojeva prikažite stvaranje gomile (max heap) algoritmom čije je vrijeme izvođenja $O(n \cdot \log n)$ (za n članova polja).

Prikazati stablo u svakom koraku algoritma te označiti članove koji zamjenjuju mjesta u pojedinom koraku algoritma.

Zadatak 4. (12 bodova)



Na slici lijevo prikazan je **usmjereni graf.**

(8 bodova) 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.

(4 boda) Napišite i matricu susjedstva za zadani graf.

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

Zadatak 5. (12 bodova)

Zadan je razred `Queue<T>` kojim se implementira red.

```
template <class T> class Queue{
public:
   Queue();
   bool enqueue (T data);
   bool dequeue (T &data);
};
```

U redu se nalaze jednaki elementi poredani jedan neposredno iza drugog. Potrebno je napisati funkciju `ucestalost` koja treba imati prototip:

```
template <class T>
int ucestalost(Queue <T> &q, int& brojNajvecih)
```

Funkcija `ucestalost` treba vratiti koliko se najviše puta neki element pojavljuje u redu, dok u varijabli `brojNajvecih` na izlazu treba biti broj različitih elemenata koji se pojavljuju najviše puta (jer je moguće da se više različitih elemenata pojavljuje jednak broj puta). Ulazni red treba nakon završetka rada funkcije ostati nepromijenjen.
Primjer: Neka je zadan red A , A , B , B , B , C , D , D , E , E , E , F . Funkcija `ucestalost` treba vratiti 3, a `brojNajvecih` treba imati vrijednost 2 jer se elementi B i E pojavljuju po 3 puta.
Napomena: u izradi funkcije `ucestalost` *nije dozvoljeno koristiti i/ili mijenjati internu implementaciju reda (polja ili pokazivače iz razreda Queue<T>), već je potrebno koristiti članske funkcije* enqueue *i* dequeue.

Zadatak 6. (6 bodova)

Odredite vrijeme izvođenja u O, Ω i, ako je moguće, Θ notaciji za zadani programski isječak. Ako se vrijeme izvođenja u Θ notaciji ne može odrediti, navedite tako u rješenju. Rješenja upišite u tablicu s desne strane zadatka.

```
// pretpostavka:
// O(f(x,y))= Ω(f(x,y))= Θ(f(x,y))=x

int f1(int n){
  s=0;
  for(i=1; i<=n; i++)
    for(j=1; j<=n; j*=2)
      s+= f(j,n-i)+ f(i,n-j);
  return s;
}
```

O	
Ω	
Θ	

Rješenja

Zadatak 1.

```
template <typename T> bool BStablo:: maxNaRazini(int razina, T* maxElement) {
	return maxNaRazini(korijen, razina, maxElement);
};

template <typename T>
bool BStablo<T>::maxNaRazini(shared_ptr<Cvor>& cvor, int razina, T* najveci) {
	if (cvor) {
		if (razina > 1) { // varijablu razina smanjujemo dok ne dođe do 1
			T najL, najR;
			bool uspjehL, uspjehR;
			uspjehL = maxNaRazini(cvor->lijevo, razina - 1, &najL); // lijevo
			uspjehR = maxNaRazini(cvor->desno, razina - 1, &najR); // desno
			if (uspjehR && uspjehL) {  // ako imamo oba kandidata, uzmemo većeg
				if (najL > najR) *najveci = najL; else *najveci = najR;
			}
			else if (uspjehR) *najveci = najR;
			else if (uspjehL) *najveci = najL;
			else return false; // ako su i uspjehR i uspjehL false
			return true;
		}
		else if (razina == 1) { // došao sam na traženu razinu i vraćam vrijednost čvora
			*najveci = cvor ->elem;
			return true;
		}
	}
	return false; // ovdje dolazim ako ili nema čvora ili je razina manja od 1
}
```

Odsječak iz glavnog programa, npr.:

```
BStablo<int> bStablo; // nakon toga neka inicijalizacija elemenata …

bool uspjeh;
int razina, najV;
uspjeh=bStablo.maxNaRazini(razina, &najV);
```

Zadatak 2.

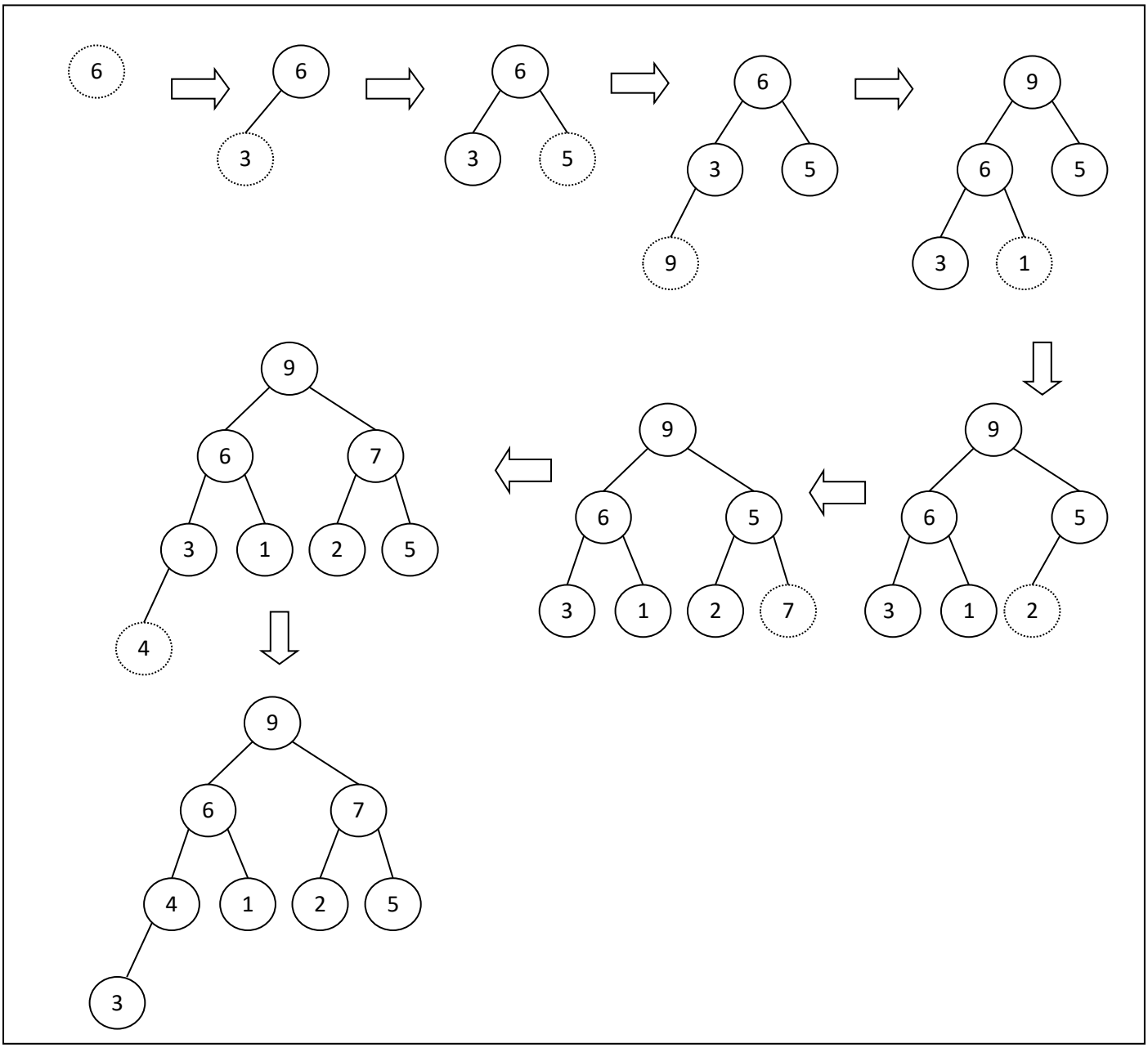
```
int bSearch(int A[], int n, int x) {   // standardno binarno pretraživanje
    int left, right, middle;
    left = 0;
    right = n - 1;
    while (left <= right) {
        middle = (left + right) / 2;
        if (A[middle] < x) left = middle + 1;
        else if (A[middle] > x) right = middle – 1;
        else return 1;
    }
    return 0;
}
```

U funkciji traziBroj se prvo postupkom sličnim binarnom pretraživanju odredi granica između neparnih i parnih brojeva, a onda se poziva funkcija bSearch.

```
int traziBroj(int A[], int n, int x) {
    int lijevo, desno, sredina, k;
    bool paranSredina, paranPrijeSredine,paranPoslijeSredine;

    if ((A[0] % 2 == 0) || (A[n-1] % 2 == 1)) {// ako su svi brojevi iste parnosti...
        return bSearch(A, n, x);
    }
    // tražim k, tj. na kojem mjestu je granica parnih i neparnih.
    // kriterij: A[k] parno, A[k-1] neparno
    lijevo = 0;
    desno = n - 1;
    while (lijevo <= desno) {
        sredina = (lijevo + desno) / 2;
        paranSredina = (A[sredina] % 2==0);
        paranPrijeSredine = (A[sredina - 1] % 2 == 0);
        if (paranSredina && !paranPrijeSredine) {
            k = sredina; break;
        }
        if (paranPrijeSredine) { // ako su oba parna ..
            desno = sredina - 1;
        }
        if (!paranSredina) { // ako su oba neparna, pogledaj i prvog sljedećeg
           paranPoslijeSredine = (A[sredina + 1] % 2 == 0);
           if (paranPoslijeSredine) {
               k = sredina + 1;
               break;
           }
           lijevo = sredina+1;
        }
    }
    if (x % 2 == 1) { // ako je x neparan ...
        return bSearch(A, k, x);
    }
    else { // ako je x paran
        return bSearch(&(A[k]), n - k, x);
    }
}
```

Zadatak 3.



Zadatak 4.

a)

1. Početno stanje Q={0,∞ ,∞ ,∞ ,∞ ,∞}

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

2. Uzimamo A, označimo da je A obiđen.
Q={3 (B) ,3(C) ,4(D) ,∞ ,∞ ,∞}

	A	B	C	D	E	F	G
Udaljenost	0	3	3	4	∞	∞	∞
Prethodnik	-	A	A	A	-	-	-

3. Uzimamo vrh B iz Q, označimo da je obiđen i podešavamo udaljenosti. Q={3(C) ,4(D), 5(E) ,∞ ,∞}

	A	B	C	D	E	F	G
Udaljenost	0	3	3	4	5	∞	∞
Prethodnik	-	A	A	A	B	-	-

4. Uzimamo C iz Q , označimo da je obiđen i podešavamo udaljenosti. Q={4 (D), 5(E), 8(G) ,∞ }

	A	B	C	D	E	F	G
Udaljenost	0	3	3	4	5	∞	8
Prethodnik	-	A	A	A	B	-	C

5. Uzimamo D iz Q , označimo da je obiđen i podešavamo udaljenosti. Q={ 5 (E) ,11(F) ,5(G)}

	A	B	C	D	E	F	G
Udaljenost	0	3	3	4	5	11	5
Prethodnik	-	A	A	A	B	D	D

5. Uzimamo E iz Q , označimo da je obiđen i podešavamo udaljenosti. Q={11(F) ,5(G)}

	A	B	C	D	E	F	G
Udaljenost	0	3	3	4	5	7	5
Prethodnik	-	A	A	A	B	D	D

6. Uzimamo G iz Q , označimo da je obiđen i podešavamo udaljenosti Q={ 6(F)}

	A	B	C	D	E	F	G
Udaljenost	0	3	3	4	5	6	5
Prethodnik	-	A	A	A	B	G	D

7. Za vrh F označimo da je obiđen. Time smo obišli sve vrhove, Q={}

b)

	A	B	C	D	E	F	G
A	0	1	1	1	0	0	0
B	0	0	0	0	1	0	0
C	1	0	0	0	0	0	1
D	0	1	1	0	1	1	1
E	0	0	0	0	0	1	0
F	0	0	0	0	1	0	0
G	0	0	0	0	0	1	0

5. Zadatak

```
template <class T>
int ucestalost(Queue <T> &q, int& brojNajvecih)
{
    int maxUcestalost, tUcestalost;
    Queue<T> qPom;
    T stariElement, trenutniElement;

    brojNajvecih = 0; maxUcestalost = 0;
    if (!q.dequeue(trenutniElement)) {
        return 0; // ako je red prazan
    };
    qPom.enqueue(trenutniElement); // sačuvaj trenutni element
    stariElement = trenutniElement; // zapamti ga
    tUcestalost = 1; // njegova trenutna učestalost
    while (q.dequeue(trenutniElement)) {   // dok ih ima ..
        qPom.enqueue(trenutniElement);
        if (trenutniElement == stariElement) { // jednak prethodnom?
            tUcestalost++; // povećaj učestalost
        }
        else { // trenutni element je različit od prethodnog
            if (tUcestalost > maxUcestalost) { // ako je riječ o novom najčešćem …
                brojNajvecih = 1;               // broj najvećih je 1
                maxUcestalost = tUcestalost;    // i zapamtimo najveću učestalost
            }
            else if(tUcestalost == maxUcestalost) {
                brojNajvecih++;                // uvećaj broj najvećih
            }
            stariElement = trenutniElement;   // idemo ponovo …
            tUcestalost = 1;
        }
    }
    if (tUcestalost > maxUcestalost) {   // nakon petlje ne zaboraviti provjeriti zadnji
        brojNajvecih = 1;                // element koji je bio u obradi
        maxUcestalost = tUcestalost;
    }
    else if (tUcestalost == maxUcestalost) {
        brojNajvecih++;
    }
    while (qPom.dequeue(trenutniElement))   // vrati sve nazad na početni red.
        q.enqueue(trenutniElement);
    return maxUcestalost;
}
```

Zadatak 6.

- Vanjska petlja (po varijabli i) se izvodi n puta,
- unutarnja petlja (po varijabli j) se izvodi floor($\log_2(n)$)+1 puta (floor(x) - zaokruži na manje). Radi jednostavnosti, računamo da se izvodi $\log_2(n)$ puta.
- Složenost možemo razbiti na računanje složenosti dva izraza: f(j,n-i) te f(i, n-j).
- Za f(j, n-i) ukupna složenost je O(n*2n) . Prvi n je n ponavljanje vanjske petlje, a član 2n se dobije kao zbroj geometrijskog reda 1+2+4+ … + 2^K , gdje je K=floor($\log_2$ (n)) .
- Za f(i, n-j) ukupna složenost je O($n^2 \log_2(n)$) . Naime, složenost izračuna ovisi samo o prvom argumentu (tj. o i), tako da se računanje složenosti svede na računanje izraza
  $1* \log_2(n)+2* \log_2(n)+3* \log_2(n)+ \ldots + n* \log_2(n) = (1+2+\ldots+n)* \log_2(n) = (n^2+n)/2 * \log_2(n)$
- Iz navedenog slijedi da je O(f1)= O($n^2 \log_2(n)$)+ O(n^2)= O($n^2 \log_2(n)$) , što isto vrijedi i za Ω(f1) i moguće Θ(f1)