JMBAG **IME I PREZIME**

Algoritmi i strukture podataka – međuispit

28. studenoga 2021.

Ispit donosi maksimalno 35 bodova. Ovaj primjerak ispita trebate predati s upisanim imenom i prezimenom te JMBAG-om. Rješenja 2. zadatka upišite u za to predviđena mjesta na ovom papiru, a rješenja ostalih zadataka napišite na svojim papirima ili unutrašnjosti košuljice. ***Korištenje statičkih varijabli (ključna riječ static) nije dozvoljeno ni u jednom zadatku!***

Zadatak 1. (9 bodova)

a) Zadano je polje prirodnih brojeva A, koje ima n članova (n ≥ 1). Potrebno je napisati **rekurzivnu** funkciju koja će u polju A obrnuti redoslijed parnih brojeva u polju. Naputak: potrebno je zamijeniti *k*-ti parni broj slijeva s *k*-tim parnim brojem zdesna (npr. prvi parni broj slijeva treba zamijeniti s prvim parnim brojem zdesna; vidjeti primjere). Neparni brojevi trebaju ostati na svojim mjestima.

Nije dozvoljeno korištenje pomoćnih struktura podataka. Funkcija treba vratiti broj obavljenih zamjena. Prototip funkcije je:

```
int zamijeniParne(int A[],int n);
```

Primjeri:

Polje A **prije** poziva funkcije	Polje A **nakon** poziva funkcije	**Broj obavljenih zamjena**
A = {10, 2, 5, 6, 7, 9, 1}	A = {6, 2, 5, 10, 7, 9, 1}	1 (zamijenjeni su 10 i 6)
A = {3, 1, 10, 2, 5, 6, 7, 8, 9, 1}	A = {3, 1, 8, 6, 5, 2, 7, 10, 9, 1}	2 (zamijenjeni su 10 i 8 te 2 i 6)
A = {1, 5, 1}	A = {1, 5, 1}	0
A = {12}	A = {12}	0

Napomena: nerekurzivna rješenja neće biti priznata.

b) Odredite složenost funkcije koju ste implementirali u a) dijelu zadatka u O, Ω i Θ notaciji.

Zadatak 2. (8 bodova) – RJEŠENJA UPISATI NA OVOME PAPIRU

Za funkcije **f i g** odredite, ako je moguće, vrijeme izvođenja u Θ notaciji, a ako nije moguće, odredite vrijeme izvođenja u O i Ω notaciji. Rješenje upišite u pravokutnik pored zadatka.
U primjeru a) polje A je polje prirodnih brojeva, čiji elementi mogu, ali i ne moraju biti sortirani (ne znamo unaprijed).

a)
```
int f(int A[], int n) { // n >= 0, pom(A, n) = Θ(log2n)
   if (n == 0) return 1;
   else {
      int zbroj = 0;
      for (int i = 1; i < n - 2; i++) {
         if (A[i] < A[i + 2])
            zbroj += pom(A, i + 2);
         else
            zbroj += i * (i + 2);
      }
      return zbroj;
   }
}
```

Rješenje:

b) double **g**(int k) { // k >= 0
  double t = 1.;
  if (k <= 1) **return** 1.;
  for (int i = 1; i < k; i++) {
  t *= **g**(i);
  }
  return t;
}
	Rješenje:

Zadatak 3. (9 bodova)

Potrebno je implementirati metode `AddFirst`, `Find` i destruktor dvostruko povezane linearne liste implementirane pokazivačima. Lista treba biti ostvarena korištenjem predložaka (eng. templates). Metoda `AddFirst` dodaje novi element na početak liste, metoda `Find` dojavljuje nalazi li se element u listi, a destruktor oslobađa memoriju svih elemenata liste. Atom liste `ListElement` i prototipovi funkcija zadani su odsječcima koda u nastavku.

```
template<typename T>
class ListElement{
   public:
      T val;
      ListElement<T>* next;
      ListElement<T>* prev;
};
```

```
template<typename T>
class DoubleList{
   …
   public:
      void AddFirst(T val){ // vaš kod za AddFirst }
      bool Find(T val){ // vaš kod za Find }
      ~ DoubleList(){ // vaš kod za destruktor }
};
```

Zadatak 4. (9 bodova)

Napisati **rekurzivnu** funkciju koja izvorišni red (`sourceq`) razdvaja u dva nova reda na način da se svi članovi na parnim pozicijama stavljaju u prvi red (`queue1`), a svi članovi na neparnim pozicijama u drugi red (`queue2`). Nakon izvršavanja funkcije izvorišni red ostaje prazan. Funkcija **mora** imati prototip:

```
template<typename T>
void split(Queue<T>* sourceq, Queue<T>* queue1, Queue<T>* queue2);
```

Sva tri reda su adekvatno stvorena i inicijalizirana prije poziva funkcije, te ih se u funkciji može koristiti. Pri radu s redom `Queue<T>` dozvoljeno je koristiti samo funkcije enqueue i dequeue:

```
template<typename T>
class Queue{
   private:
      // privatne članske varijable koje se ne smiju
      // koristiti/mijenjati
   public:
      bool enqueue(T val){
            // Dodavanje u red. Vraća true ako je dodavanje
            // uspjelo, a false inače
      }
      bool dequeue(T& val){
            // Skidanje iz reda. Vraća true ako je skidanje
            // uspjelo, a false inače
      }
};
```

Primjer:

- `sourceq = 0,1,2,3,4,5,6,7,8,9`
- `queue1 = 0,2,4,6,8`
- `queue2 = 1,3,5,7,9`

Korištenje članskih varijabli koje se koriste za implementaciju reda **nije dozvoljeno**.

Promjena prototipa funkcije i pozivanje pomoćnih funkcija nije dozvoljeno!

Rješenja:

1. zadatak

a)

```
int zamijeniParne(int A[], int n) {
	int brZamjena = 0; // broj zamjena
	int pomakniPocetak = 0;
	int pomakniKraj = 0;
	if (n > 1) {
		if (A[0] % 2 == 0 && A[n - 1] % 2 == 0) {
		// A[0] i A[n-1] su parni, pa ih možemo zamijeniti
			int temp = A[0];
			A[0] = A[n - 1];
			A[n - 1] = temp;
			pomakniPocetak = 1;
			pomakniKraj = 1;
			brZamjena = 1;
		}
		if (A[0] % 2) { // A[0] neparan
			pomakniPocetak = 1;
		}
		if (A[n - 1] % 2) { // A[n-1] je neparan
			pomakniKraj = 1;
		}
		// ažuriraj početni element i duljinu polja za sljedeći rekurzivni poziv fje
		brZamjena += zamijeniParne(&A[pomakniPocetak], n - pomakniPocetak - pomakniKraj);
	}
	return brZamjena;
}
```

b) $O(n)$, $\Omega(n)$ i $\Theta(n)$

2. zadatak:

- najlošiji slučaj:
  ako je polje sortirano (uvijek je A[i] < A[i + 2]), onda je složenost:
  $T(n) = \log(3) + \log(4) + \ldots + \log(n) = \log(3 \cdot 4 \cdot \ldots \cdot n) = \log(1/2) + \log(n!) = \Theta(\log(n!))$

- najbolji slučaj:
  ako je polje sortirano obrnutim redoslijedom ili su svi elementi jednaki (uvijek je A[i] >= A[i + 2]) , onda je složenost: $\Theta(n)$

Konačno, uzimajući u obzir sve slučajeve za funkciju **f** vrijedi: $\Omega(n)$ i $O(\log(n!))$ (ili $O(n\log n)$)

b) $T(1) = \Theta(1)$

$T(2) = T(1) + \Theta(1) = T(1)$

$T(3) = T(1) + T(2) = 2T(1)$,

$T(4) = T(1) + T(2) + T(3) = T(1) + T(1) + 2T(1) = 4T(1) = 2^2T(1)$, … ,

$T(k) = T(1) + T(2) + \ldots + T(k-1) = T(1) + T(1) + \ldots + 2^{k-3}T(1) = 2^{k-2}T(1) = 2^{k-2}\Theta(1) = \Theta(2^k)$

3. zadatak

```
template<typename T>
class ListElement{
   public:
      T val;
      ListElement<T>* next;
      ListElement<T>* prev;
};

template<typename T>
class DoubleList{
   public:
      ListElement<T>* head;
      ListElement<T>* tail;
      DoubleList() {this->head = nullptr; this->tail=nullptr;}

      void AddFirst(T val){
         ListElement<T>* newElem = new ListElement<T>();
         newElem->val = val;
         newElem->next = this->head;
         if(this->head != nullptr) this->head->prev = newElem;
         newElem->prev = nullptr;
         this->head = newElem;
         if(this->tail == nullptr) this->tail = newElem;
         return;
      }
      bool Find(T val){
         ListElement<T>* tmp = head;
         while(tmp){
            if(tmp->val == val) return true;
            tmp = tmp->next;
         }
         return false;
      }
      ~List(){
         while(head){
            ListElement<T>* tmp = head->next;
            delete head;
            head = tmp;
         }
         tail = nullptr;
      }
};
```

4. zadatak

```
template<typename T>
void split(Queue<T>* sourceq, Queue<T>* queue1, Queue<T>* queue2){
   T val;
   if(sourceq->dequeue(val)){
      queue1->enqueue(val);
      split(sourceq, queue2, queue1);
   }
}
```