JMBAG **IME I PREZIME**

Algoritmi i strukture podataka – međuispit

29. studenoga 2021.

Ispit donosi maksimalno 35 bodova. Ovaj primjerak ispita trebate predati s upisanim imenom i prezimenom te JMBAG-om. Rješenje 3. zadatka upišite u za to predviđena mjesta na ovom papiru, a rješenja ostalih zadatka napišite na svojim papirima ili unutrašnjosti košuljice. Ispit se piše 105 minuta.

Zadatak 1. (7 bodova)

Zadani su razredi `Stack<T>` i `Queue<T>` za koji su definirani konstruktor, članske funkcije za dodavanje elementa te skidanje elementa. Definicije razreda i prototipovi navedenih funkcija su:

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

Napišite funkciju `preurediRed` koja će sve elemente iz reda koji su manji od neke zadane vrijednosti v prebaciti na početak tog istog reda, a ostale elemente na kraj tog istog reda. Nakon prebacivanja, elementi koji su manji od zadane vrijednosti v trebaju ostati u međusobno nepromijenjenom redoslijedu na početku reda, a elementi koji su veći ili jednaki v trebaju ostati u međusobno nepromijenjenom redoslijedu na kraju reda (vidjeti primjere). Za preuređenje elemenata reda dozvoljeno je koristiti isključivo pomoćni stog/stogove. Rješenja koja ne koriste isključivo pomoćni stog/stogove neće se priznati. Pretpostavite da su definirani operatori usporedbe za tip T. Prototip funkcije je:

```
template <class T> void preurediRed (Queue<T>* q, T v);
```

Napišite i odsječak glavnog programa u kojemu ćete definirati varijablu tipa `Queue`, napuniti tako definiran red podatcima tipa `int` te pozvati funkciju `preurediRed`.

Primjeri:

Red prije preuređenja: **IZLAZ --** 4, 1, 5, 7, 8, 2, 3, 9 **-- ULAZ**

Za v = 5, red nakon preuređenja treba biti: **IZLAZ --** 4, 1, 2, 3, 5, 7, 8, 9 **-- ULAZ**

Red prije preuređenja: **IZLAZ --** 14, 11, 5, 7, 8, 9 **-- ULAZ**

Za v = 5, red ostaje nepromijenjen nakon poziva funkcije `preurediRed`: **IZLAZ --** 14, 11, 5, 7, 8, 9 **-- ULAZ**

Zadatak 2. (7 bodova)

Zadani su nizovi A i B duljine `m` i `n`, koji predstavljaju **skupove** cijelih brojeva (`m` je broj elemenata skupa A, a `n` je broj elemenata skupa B). Brojevi unutar nizova uzlazno su poredani. Napišite rekurzivnu funkciju `jestPodskup` koja će provjeriti je li skup B podskup skupa A. Ako je B prazan skup (tj. ako je `n` = 0), funkcija treba vratiti logičku istinu. Ako je skup A prazan (tj. ako je `m` = 0), onda funkcija treba vratiti logičku laž. Vrijeme izvođenja funkcije treba biti O(`m`). Prototip funkcije je:

```
bool jestPodskup(int A[], int B[], int m, int n);
```

Primjeri:

A = {1, 2, 4, 6, 7, 9}, B = {1, 4, 9} → B je podskup skupa A (funkcija treba vratiti logičku istinu)

A = {1, 2, 4, 6, 7, 9}, B = {1, 4, 10} → B nije podskup skupa A (funkcija treba vratiti logičku laž)

Zadatak 3. (7 bodova) – RJEŠENJA UPISATI NA OVOME PAPIRU

Za funkciju **f** odredite vrijeme izvođenja u O i Ω notaciji i, ako je moguće, vrijeme izvođenja u Θ notaciji. Rješenje upišite u pravokutnik pored zadatka.

```
int f(int n) { // n >= 0, g(n) = Θ(n)
      if (n == 0) return 1;
      else {
            int umnozak = 1;
            for (int i = 1; i <= n; i *= 10)
                  umnozak *= g(5) + g(i);
            return umnozak;
      }
}
```

Rješenje:

Zadatak 4. (7 bodova)

Napisati rekurzivnu funkciju `JumpSearchRec` kojom se obavlja **rekurzivno blokovsko traženje** (eng. jump search) elementa u polju. Prototip funkcije `JumpSearchRec` i pozivajuća funkcija `JumpSearch` su zadani kôdom:

```
bool JumpSearchRec(int A[], const size_t n, const size_t blockSize, const int &item);
bool JumpSearch(int A[], const size_t n, const int &item) {
   return JumpSearchRec(A, n, sqrt(n), item);
}
```

Prilikom implementacije funkcije `JumpSearchRec` potrebno je koristiti već postojeću funkciju za slijedno traženje (ne treba ju zasebno implementirati) zadanu prototipom:

```
bool LinearSearch(int A[], const size_t n, const int &item);
```

Funkcija `JumpSearchRec` treba biti ostvarena u složenosti $O(n^{1/2})$, pri čemu je n broj elemenata polja u kojem se traži.

Napomena: Funkcija mora biti implementirana rekurzivno; iterativna rješenja se neće priznavati. Funkcije JumpSearch i LinearSearch ne treba pisati, već ih se samo koristi.

Zadatak 5. (7 bodova)

Zadana je dinamička jednostruka lista koja ima pokazivače na glavu (head) i rep (tail):

```
template <typename T> class List{
      private:
            template <typename X> class ListElement{
      public:
            X el;
            ListElement<X> *next;
      };
      ListElement<T> *head = nullptr;
      ListElement<T> *tail = nullptr;
}
```

Napisati člansku funkciju `RemoveFirst` klase `List` koja briše jedan element iz liste. Elemente se briše s glave liste. Funkcija za brisanje treba imati prototip:

```
bool RemoveFirst();
```

te vraćati `true` ako je brisanje uspjelo, a `false` inače (ako je lista prazna).

Napisati i vanjsku (ne člansku) funkciju koja će listu zadanu argumentom u potpunosti obrisati pozivanjem prethodno implementirane funkcije `RemoveFirst` koja ima prototip:

```
template <typename T> bool RemoveAllFromList(List<T> *lista);
```

Funkcija `RemoveAllFromList` vraća `true` ako je obrisan barem jedan element liste, a `false` ako je lista bila prazna.

RJEŠENJA

Zadatak 1. (7 bodova)

```
template <class T> void preurediRed(Queue<T>* q, T v) {
    Stack<T> Sm, Sv, Spom;
    T elem;
    while (q->dequeue(elem)) {
        if (elem < v) Sm.push(elem);
        else Sv.push(elem);
    }
    while (Sm.pop(elem))
        Spom.push(elem);
    while (Spom.pop(elem))
        q->enqueue(elem);

    while (Sv.pop(elem))
        Spom.push(elem);
    while (Spom.pop(elem))
        q->enqueue(elem);
}

int main(void) {
    Queue<int> q;
    // …
    preurediRed(&q, 15);

}
```

Zadatak 2. (7 bodova)

```
//Provjeriti je li skup B podskup skupa A. |A| = m, |B| = n
//Ako je B prazan skup (tj.ako je n = 0), funkcija treba vratiti logičku istinu.
//Ako je skup A prazan (tj.ako je m = 0), funkcija treba vratiti logičku laž.
bool jestPodskup(int A[], int B[], int m, int n) {
      if (n == 0) return true;
      else if (m == 0) return false;
      else if (n > m) return false;

      if (B[n - 1] > A[m - 1])
            return false;
      else if (B[n - 1] == A[m - 1])
            return jestPodskup(A, B, m - 1, n - 1);
      else if (n < m)
            return jestPodskup(A, B, m - 1, n);
}
```

Drugo moguće rješenje:

```
bool jestPodskup(int A[], int B[], int m, int n) {
   if (n == 0) {
      return true;
   } else if (m == 0) {
      return false;
   }

   if (A[m - 1] == B[n - 1]) {
      return jestPodskup(A, B, m - 1, n - 1);
   } else {
      return jestPodskup(A, B, m - 1, n);
   }
}
```

Zadatak 3. (7 bodova)

Rješenje: f(n) = Ω(n) = O(n) = Θ(n)

Objašnjenje rješenja:

Općenito, petlja se obavlja k = $\lfloor log_{10} n \rfloor$ + 1 puta. Npr. ako je n = 10^k, gdje je k prirodni broj, tada je broj prolaza kroz petlju k + 1. Npr. za n = 100, petlju prolazimo za i = 1, 10, 100, tj. ukupno 3 puta (3 = $\lfloor log_{10} 100 \rfloor$ + 1)

Pretpostavimo zbog jednostavnosti da je n = 10^k, gdje je k prirodni broj, pa vrijedi:

za i = 1: Θ(1) + Θ(1)

za i = 10: Θ(1) + Θ(10)

za i = 100: Θ(1) + Θ(100)

…

za i = n = 10^k: Θ(1) + Θ(10^k)

Ukupno: (k+1) · Θ(1) + Θ(1 + 10 + … + 10^k) = Θ(k) + Θ(10^k) = Θ(log n) + Θ(n) = Θ(n)

Usporedbe radi, primjer programskog odsječka za koji bi vrijeme izvođenja bilo Θ(n·log n):

```
int f(int n) { // n >= 0, g(n) = Θ(n)
       if (n == 0) return 1;
       else {
              int umnozak = 1;
              for (int i = 1; i <= n; i *= 10)
                     umnozak *= g(5) + g(n);
              return umnozak;
       }
}
```

Zadatak 4. (7 bodova)

```
bool JumpSearchRec (int A[], const size_t n, const size_t blockSize, const int &item) {
      int next = std::min((int)blockSize, (int)n - 1);
      // find the right block
      if (next < n - 1 && A[next] <= item) {
         return JumpSearchRec (((int *)A) + blockSize, n - blockSize, blockSize, item);
      }
      // search the block
      if (next == n - 1) {
         ++next;
      }
      return LinearSearch (A, next, item);
}
```

Zadatak 5. (7 bodova)

```
bool RemoveFirst(){
     if(head != nullptr){
        ListElement<T> *pom = head;
        head = head->next;
        delete pom;
        if(head == nullptr)
            tail = nullptr;
         return true;
      }
      return false;
  }

template <typename T> bool RemoveAllFromList(List<T> *lista){
   bool hasRemoved = false;
   while(lista->RemoveFirst()) {hasRemoved = true; };
   return hasRemoved;
}
```