

# Egzamin z Zaawansowanego Programowania Komputerowego

18 czerwca 2021

1. (10 pkt.) Napisać funkcję

```
vector<int> a(vector<int> s, vector<int> t);
```

Jej parametrami są dwie tablice zawierające ściśle rosnące ciągi liczb. Funkcja powinna zwrócić tablicę (wektor) zawierającą wszystkie liczby, które znajdują się zarówno w tablicy  $s$ , jak i w tablicy  $t$ , uporządkowane w kolejności rosnącej. Zadbać o to, żeby napisana funkcja miała złożoność liniową.

2. (15 pkt.) Mówimy że tablica  $t$  jest pełna jeśli nie istnieją indeksy  $a < b < c$  takie, że  $t[a] > t[b] < t[c]$ .

Napisać funkcję

```
vector<int> b(vector<int> s);
```

która zwraca pełną tablicę  $t$  o takim samym rozmiarze jak  $s$ , która

- jest pełna,
- wszystkie jej elementy są nie mniejsze od odpowiadających im elementów  $s$  (tj.  $t[i] \geq s[i]$  dla wszystkich  $i$ )
- elementy  $t[i]$  są najmniejsze możliwe spośród spełniających powyższe warunki.

Przykłady:

$s$	{3,1,5}	{1,4,2,7,3,4,6,2}	{2,1,7,3,5,2,4,1,2}	{1,2,2,4,3}
wynik ( $t$ )	{3,3,5}	{1,4,4,7,6,6,6,2}	{2,2,7,5,5,4,4,2,2}	{1,2,2,4,3}

3. (10 pkt.) Dana jest funkcja

```
int c(int a, int b) {  
    if(a*b==0) return a+b;  
    return c(a,b-1)+c(a-1,b-1)+c(a-1,b);  
}
```

Napisać równoważną funkcję, która nie używa rekurencji i podać jej złożoność. Można założyć, że parametry  $a$  i  $b$  nie są ujemne.

4. (10 pkt.) Dana jest struktura

```
class ElListy{public: int wartosc; ElListy* nast; };
```

której będziemy używać do tworzenia list jednokierunkowych.

Napisać funkcję

```
ElListy* naKoniec(ElListy* a);
```

która przestawia pierwszy element listy  $a$  na koniec i zwraca wskaźnik do początku listy. Należy użyć istniejących już elementów.

5. (15 pkt.) Dana jest struktura

```
class Wezel{  
public:  
    int wartosc;  
    Wezel* lewy;  
    Wezel* prawy;  
};
```

której będziemy używać do tworzenia drzew.

Napisać funkcję

```
bool zrownowane(Wezel* d);
```

która zwraca true wtedy i tylko wtedy, gdy drzewo  $d$  jest zrównoważone, tj. w każdym węźle wysokości obu poddrzew różnią się nie więcej niż o 1.

6. (20 pkt.) Stworzyć klasę  $DG$ , której obiekty reprezentują drzewa genealogiczne. Osoby są reprezentowane przez napisy. Dla każdej osoby w drzewie możemy (choć nie musimy) znać jej rodziców. Zakładamy, że wszystkie osoby mają różne nazwy oraz że użytkownik używa klasy w poprawny sposób (tj. żadna osoba nie jest swoim przodkiem, nikt nie jest jednocześnie ojcem i matką, itp.).

```

class DG {
public:
    DG();
    void dodaj(string osoba, string matka, string ojciec);
    void ustawRodzicow(string osoba, string matka, string ojciec);
    void usun(string osoba);
    string matka(string osoba);
    vector<string> przodkowie(string osoba);
};

```

Opis metod publicznych, które należy napisać:

```
DG();
```

Tworzy puste drzewo genealogiczne.

```
void dodaj(string osoba, string matka, string ojciec);
```

Dodaje osobę wraz z informacją o jej rodzicach. Jeśli `matka=""` oznacza to, że matka jest nieznana (to samo w przypadku ojca). Można założyć, że jeśli rodzic został podany, to występuje już w drzewie genealogicznym.

```
void ustawRodzicow(string osoba, string matka, string ojciec);
```

Ustawia informację o rodzicach osoby znajdującej się już w drzewie. Można założyć, że parametry `matka` i `ojciec` znajdują się już w drzewie lub są puste.

```
void usun(string osoba);
```

Usuwa osobę. Jeśli ta osoba była rodzicem, jej dzieci będą miały po usunięciu nieznanego rodzica.

```
string matka(string osoba);
```

Zwraca matkę podanej osoby.

```
vector<string> przodkowie(string osoba);
```

Zwraca tablicę zawierającą przodków podanej osoby w linii męskiej, tj. ojca, dziadka, pradziadka, itd. aż do momentu, gdy przodek będzie nieznanym.