Zaawansowane programowanie komputerowe

Zadania przygotowawcze do egzaminu
czerwiec 2022

A. Funkcje.

A1. Napisać funkcję, która zwraca wartość true wtedy i tylko wtedy, gdy wektor v jest uporządkowany (rosnąco lub malejąco).

```
bool f(vector<int> v);
```

A2. Napisać funkcję, która zwraca medianę elementów z tablicy t o rozmiarze n, tj. dowolną liczbę m taką, że zbiory {i: t[i]>m} i {i:t[i]<m} są równoliczne.

```
int f(vector<double> t);
```

A3. Napisać funkcję

```
int f(int a, int b, int n);
```

która zlicza liczbę sposobów, na jakie można zapisać liczbę n w postaci sumy liczb a i b (utożsamiamy różne kolejności składników)

A4. Napisać funkcję

```
bool f(string a, string b);
```

która zwraca true wttw jeden napis jest przesunięciem drugiego. Np. przesunięciami napisu "ABCD" są napisy "BCDA", "CDAB", "DABC" i oczywiście "ABCD".

A5. Napisać funkcję

```
int f(vector<int> t);
```

która zwraca długość najdłuższego ciągu kolejnych elementów tablicy t tworzących ciąg arytmetyczny.

A6. Napisać funkcję

```
bool f(int n, int a, int b, int c)
```

która zwraca true wtedy i tylko wtedy, gdy n można przedstawić w postaci sumy, której wszystkie składniki są równe jednej z liczb a, b, c. Oszacować złożoność napisanej funkcji.

A7. Napisać funkcję

```
int g(vector<int> t)
```

która zwraca długość najdłuższego ciągu kolejnych elementów tablicy t o dodatniej sumie. Jeśli żaden z elementów tablicy nie jest dodatni, funkcja powinna zwrócić 0.

A8. Napisać funkcję

```
int f(int n, int a, int b)
```

która zwraca liczbę przedstawień liczby n w postaci sumy składników równych a lub b. Zakładamy, że wszystkie parametry są dodatnie. Rozróżniamy różne kolejności składników. Oszacować złożoność napisanej funkcji.

A9. Napisać funkcję

```
vector<double> g(vector<double> t1, vector<double> t2, vector<double> t3)
```

Parametry t1, t2, t3 to posortowane rosnąco wektory. Funkcja powinna stworzyć nowy wektor i przepisać do niego elementy z wektorów t1, t2, t3 tak, aby zachować porządek rosnący. Należy zadbać o to, aby funkcja działała jak najszybciej.

A10. Napisać funkcję

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

która zwraca najmniejszą liczbę całkowitą dodatnią, która nie występuje w tablicy t. Oszacować złożoność napisanej funkcji.

A11. Napisać funkcję

```
bool a(string s);
```

która przestawia, o ile to możliwe, znaki w napisie `s` tak, aby dwa jednakowe znaki nie znajdowały się na sąsiednich pozycjach, a nastrępnie zwraca `true`. Jeśli nie jest to możliwe, należy zwrócić `false`, a napis `s` powinien pozostać niezmieniony.

A12. Napisać funkcję

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

która zwraca najmniejszą liczbę całkowitą większą od 1, która nie jest podzielna przez żaden element tablicy t.

A13. Napisać funkcję

```
string b(string s, string t);
```

która oblicza sumę dwóch liczb całkowitych dodatnich, których przedstawienia dziesiętne są zapisane w napisach s i t. Wynik należy zwrócić w stworzonym przez funkcję napisie. Jeśli s lub t nie jest poprawnie zapisaną liczbą, należy pusty napis. *Uwaga*: nie możemy zakładać, że liczby są na tyle małe, aby dało się je zapisać używając typów standardowych (np. int). *Podpowiedź:* wartość cyfry możemy uzyskać używając wyrażenia c-'0', a cyfra odpowiadająca liczbie n to '0'+n.

A14. Napisać funkcję

```
void a(vector<string> v);
```

która drukuje wszystkie napisy występujące dokładnie jeden raz w wektorze v. Oszacować złożoność napisanej funkcji.

A15. Napisać funkcję

```
int b(vector<int> t, int x);
```

która przestawia wszystkie elementy wektora t mniejsze lub równe x na początkowe miejsca w tablicy, a większe od x na końcowe (tak aby t[0] … t[a] <= x oraz x<t[a+1] … t[t.size()-1] dla pewnego indeksu a). Należy zadbać o to, aby funkcja działała jak najszybciej (tzn. w czasie liniowym).

B. Rekurencja.

B1. Dana jest funkcja

```
int f(int n) {
	if(n<3) return 1;
	return f(n-1)+f(n-2)+f(n-3);
}
```

a. Obliczyć przybliżoną złożoność tej funkcji ze względu na parametr n.
b. Napisać równoważną funkcję nierekurencyjną.

B2. Dana jest funkcja

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

a. Obliczyć przybliżoną złożoność tej funkcji.
b. Napisać równoważną funkcję nierekurencyjną.

B3. Napisać funkcję, która wypisuje na ekran wszystkie możliwe przedstawienia liczby n w postaci nierosnącej sumy liczb całkowitych dodatnich. Oszacować złożoność tej funkcji.

B4. Dana jest funkcja

```
int f(int k) {
	if(k) return 1+k%10+f(k/10);
	return 0;
}
```

Obliczyć złożoność tej funkcji w zależności od k. Napisać równoważną funkcję, która nie używa rekurencji.

B5. Dana jest funkcja

```
int f(int n, int k) {
	if(n==1) return 2;
	if(n%k==0)
		return k*k*f(n/k, k);
	else
		return f(n, k+1);
```

```
}
```
Obliczyć złożoność wywołania tej funkcji w zależności od n, jeśli parametr k jest równy 2. Napisać równoważną funkcję, która nie używa rekurencji.

B6. Dana jest funkcja

```
int c(int n) {
      if(n<=0) return 0;
      if(n==1) return -1;
      return -c(n-1)-c(n-1)-c(n-2);
}
```

Napisać równoważną funkcję, która nie używa rekurencji i działa jak najszybciej.

C. Klasy.

C1. Stworzyć klasę Tablica przechowującą elementy typu double.

```
class Tablica {
public:
      Tablica(int n, double x); // tworzy nową tablicę wypełnioną elementami x
      void ustaw(int i, double v); // ustawia wartość o wskazanym indeksie
      double wartosc(int i); // zwraca wartość o podanym indeksie
      Tablica& dolacz(Tablica& t); // dołącza do tablicy elementy z tablicy
będącej parametrem
      int rozmiar(); // zwraca rozmiar tablicy
};
```

C2. Stworzyć klasę Czas, która przechowuje aktualny czas (godzinę i minutę)

```
class Czas {
public:
      Czas();
      Czas(int h, int m);
      Czas& dodajGodziny(int ileGodzin);
      Czas& dodajMinuty(int ileMinut);
      void drukuj(); // drukuje godzinę w formacie 12-godzinnym (np. 7:30AM lub
1:14PM)
      int zaIleMinut(Czas t); // podaje, za ile minut nastąpi podany czas
};
```

C3. Napisać klasę ZbiorNapisow, która przechowuje zbiory napisów

```
class ZbiorNapisow {
public:
      ZbiorNapisow(); // tworzy pusty zbiór
      void dodaj(char* napis); // dodaje napis
      void usun(char* napis); // usuwa napis
      bool nalezy(char* napis); // sprawdza, czy podany napis należy do zbioru
      int ile(); // zwraca liczbę napisów
      double sredniaDlugosc(); // zwraca średnią długość napisu ze zbioru
};
```

C4. Labirynt to prostokątny budynek podzielony na kwadratowe pomieszczenia jednakowej wielkości. Pomieszczenia są ponumerowane parami liczb całkowitych (a,b), 0<=a<s, 0<=b<w, gdzie s i w są rozmiarami labiryntu. Pomiędzy sąsiednimi pomieszczeniami może znajdować się przejście, pomieszczenia skrajne nie mogą mieć wyjść na zewnątrz. Napisać klasę Labirynt, której obiekty reprezentują labirynty opisane powyżej.

```
class Labirynt {
public:
```

```
	Labirynt(int s, int w); // tworzy nowy Labirynt o rozmiarach s,w bez przejść pomiędzy polami
	void wstawPrzejscie(int x, int y, char k); // wstawia przejście z pomieszczenia (x,y) w kierunku k
	void usunPrzejscie(int x, int y, char k); // usuwa przejście z pomieszczenia (x,y) w kierunku k
	bool jestPrzejscie(int x, int y, char k); // zwraca informację czy jest przejście
}
```
Możliwe kierunki: N, E, S, W.

C5. Napisać klasę T której obiekty reprezentują uporządkowane ciągi liczb typu double o długości nie większej niż 100.
```
class T {
public:
	T(); // tworzy nowy pusty ciąg
	void dodaj(double x); // dodaje nową liczbę x
	bool nalezy(double x); // zwraca przynależność
	void usun(double x); // usuwa podaną liczbę (tylko jedno wystąpienie)
	void usunWszytkie(double x); // usuwa wszystkie wystapienia
	double& operator[](int i); // zwraca i-ty najmniejszy element lub 0 jeśli indeks jest niepoprawny
};
```

C6. Napisać klasę Drzewa, którek obiekty reprezentują zbiory drzew w lesie. Każde drzewo jest reprezentowane przez dwie współrzędne i literę kodującą gatunek drzewa. Zakładamy, że drzew jest nie więcej niż 100.
```
class Drzewa {
public:
	Drzewa(); // tworzy pusty las
	void dodaj(double x, double y, char gatunek); // dodaje nowe drzewo
	void usun(char gatunek); // usuwa wszystkie drzewa o podanym gatunku
	int ile(char gatunek); // zwraca liczbę drzew o podanym gatunku
	char najblizszy(double x, double y); // zwraca gatunek drzewa rosnącego najbliżej zadanego punktu ('?' jeśli nie ma żadngo drzewa)
}
```

C7. Stworzyć klasę Mecze, której obiekty reprezentują zbiory meczów piłkarskich pomiędzy drużynami ponumerowanymi 1,..,n. Zakładamy, że dane dwie drużyny mogą rozegrać przeciwko sobie co najwyżej jeden mecz.
```
class Mecze {
public:
	Mecze(int ileDruzyn);
	void dodajMecz(int drA, int drB, int bramkiA, int bramkiB);
	bool graly(int drA, int drB);
	int ileMeczow(int dr);
	int ileZwyciestw(int dr);
	int max();
};
```

Opis metod publicznych, które należy napisać:

```
	Mecze(int ileDruzyn);
```
Tworzy pusty zbiór meczów pomiędzy drużynami `1,...,ileDruzyn`
```
	void dodajMecz(int drA, int drB, int bramkiA, int bramkiB);
```
Dodaje mecz pomiędzy drużynami `drA` i `drB` zakończony wynikiem bramkiA:bramkiB. Jeśli te drużyny rozegrały już mecz, nic się nie dzieje.
```
	bool graly(int drA, int drB);
```
Zwraca `true` jeśli dane drużyny grały mecz.
```
	int ileMeczow(int dr);
```
Zwraca liczbę meczów rozegranych przez drużynę `dr`.
```
	int ileZwyciestw(int dr);
```

Zwraca liczbę zwycięstw odniesionych przez drużynę `dr`.

```
int max();
```

Zwraca numer drużyny, która strzeliła najwięcej bramek. Jeśli takich drużyn jest wiele, należy podać tą o najwyższym numerze.

C8. Stworzyć klasę PlanLekcji, której obiekty reprezentują rozkład zajęć szkolnych. Rozkład dotyczy dni od poniedziałku do piątku. Każdego dnia mogą odbywać się lekcje o ponumerowanych terminach (od 1 do n). W każdym terminie może odbywać się (lub nie) lekcja o podanej nazwie (np. matematyka lub historia).

```
class PlanLekcji {
public:
      PlanLekcji(int ileTerminow);
      void dodajLekcje(int dzien, int termin, char* nazwa);
      void usunLekcje(int dzien, int termin);
      int ileLekcji();
      int poczatek(int dzien);
      bool okienko(int dzien);
      bool czyJest(int dzien, char* nazwa);
};
```

Opis metod publicznych, które należy napisać:

```
      PlanLekcji(int ileTerminow);
```

Tworzy pusty plan lekcji, każdego dnia dostępne są terminy od 1 do `ileTerminow`.

```
      void dodajLekcje(int dzien, int termin, char* nazwa);
```

Dodaje lekcję dnia `dzien` (1 – poniedziałek, 5 - piątek) o podanych terminie i nazwie.

```
      void usunLekcje(int dzien, int termin);
```

Powoduje, że w dany dzień o podanym terminie nie ma lekcji.

```
      int ileLekcji();
```

Zwraca liczbę lekcji w tygodniu.

```
      int poczatek(int dzien);
```

Zwraca termin pierwszej lekcji danego dnia lub 0 jeśli żadnej nie ma.

```
      bool okienko(int dzien);
```

Zwraca `true` wtedy i tylko wtedy, gdy danego dnia istnieje przerwa pomiędzy lekcjami (np. w terminach 1,2,5 są lekcje, a w terminach 3,4 ich nie ma).

```
      bool czyJest(int dzien, char* nazwa);
```

Zwraca `true` wtedy i tylko wtedy, gdy danego dnia odbywa się lekcja o podanej nazwie.

C9. Stworzyć klasę `Macierz`, której obiekty reprezentują kwadratowe macierze o współczynnikach typu `double`.

```
class Macierz {
public:
      Macierz(int rozmiar);
      Macierz(Macierz& m, int i, int j);
      void ustaw(int x, int y, double wartosc);
      double wartosc(int x, int y);
      void przestawWiersze(int x1, int x2);
      bool rosnaceWiersze(int x1, int x2);
};
```

Opis metod publicznych, które należy napisać:

```
      Macierz(int rozmiar);
```

Tworzy macierz jednostkową o podanym rozmiarze (na przekątnej: 1, poza przekątną: 0)

```
      Macierz(Macierz& m, int i, int j);
```

Tworzy macierz, która jest taka jak m z usuniętym i-tym wierszem i j-tą kolumną.

```
      void ustaw(int x, int y, double wartosc);
```

Ustawia wartość w x-tym wierszu i y-tej kolumnie.

```
      double wartosc(int x, int y);
```

Zwraca liczbę stojącą w x-tym wierszu i y-tej kolumnie.

```
      void przestawWiersze(int x1, int x2);
```

Przestawia wiersze o podanym numerach.

```
      bool rosnaceWiersze(int x1, int x2);
```

Zwraca true wtedy i tylko wtedy, gdy wszystkie wiersze macierzy są rosnące.

D. Listy i drzewa.

Dana jest struktura

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

której będziemy używać do tworzenia list jednokierunkowych.

D1. Napisać funkcję

```
ElListy* sklej(ElListy* a, ElListy* b);
```

która zwraca listę, w której występują najpierw elementy listy a (kolejność elementów powinna być zachowana), a następnie elementy listy b. Należy użyć istniejących już elementów z list a i b.

D2. Napisać funkcję

```
bool petla(ElListy* a);
```

która zwraca true wtedy i tylko wtedy, gdy lista wskazywana przez a zawiera pętlę, tzn. jeśli przechodząc do kolejnych elementów listy nigdy nie dojdziemy do końca.

D3. Napisać funkcję

```
ElListy* usun(ElListy* a, int w);
```

która usuwa z listy a wszystkie wartości równe w i zwraca tą listę.

D4. Napisać funkcję

```
bool zawiera(ElListy* a, ElListy* b);
```

która zwraca true wtedy i tylko wtedy, gdy każda wartość znajdująca się na liście b znajduje się również na liście a.

D5. Napisać funkcję

```
ElListy* nakoniec(ElListy* a);
```

która przestawia pierwszy element listy na koniec i zwraca wskaźnik do jej pierwszego elementu. Należy użyć istniejących już elementów z listy a.

Dana jest struktura

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

którą będziemy używać do tworzenia drzew.

D6. Liściem nazywamy węzeł drzewa, który nie ma poddrzew, tj. oba wskaźniki lewy, prawy mają wartość 0. Napisać funkcję

```
int ileLisci(Wezel* d);
```

która zwraca liczbę liści w drzewie d.

D7. Napisać funkcję

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

która zwraca true wtedy i tylko wtedy, gdy wszystkie wartości w drzewie d są różne.