Zadania na tablice

1. (*) Napisz funkcję `void rotate(int n, int t[], int k)`, która dokona rotacji cyklicznej tablicy `t` (zawierającej `n` liczb całkowitych) w prawo o `k` pozycji. Twoje rozwiązanie nie powinno korzystać z żadnych dodatkowych tablic pomocniczych. Parametr `k` może mieć dowolną wartość. W szczególności `k < 0` powinno spowodować rotację cykliczną tablicy w lewo.
2. Napisz procedurę `int* shuffle(int n, int a[], int m, int b[])`, która dla danych dwóch tablic postaci $\{x_1, x_2, \ldots, x_n\}$ oraz $\{y_1, y_2, \ldots, y_m\}$ stworzy tablicę postaci $\{x_1; y_1; x_2; y_2; \ldots\}$. Jeżeli jedna z danych tablic jest dłuższa, to jej końcowe elementy trafiają na koniec tablicy wynikowej.
   Na przykład: `shuffle(7, {3, 2, 8, 1, 9, 3, 6}, 3, {5, 7, 0}) == {3, 5, 2, 7, 8, 0, 1, 9, 3, 6}`.
3. (*) Dane są tablice liczb całkowitych uporządkowane niemalejąco. Oblicz ile różnych liczb występuje w tych tablicach.
4. Ciąg liczb naturalnych $x_0, x_1, \ldots$ nazwiemy ciągiem Bonifacego wyznaczonym przez skończony niepusty ciąg zerojedynkowy $\{b_0; \ldots; b_{k-1}\}$, jeśli $x_0 = 0$, $x_1 = 1, x_2 = 1$, a dla $i \geq 3$ zachodzi $x_i = x_{i-1} + x_{i-2}$ jeśli $b_{i \bmod k} = 0$ oraz $x_i = x_{i-1} + x_{i-3}$ dla $b_{i \bmod k} = 1$ (albo inaczej $x_i = x_{i-1} + x_{i-2-b_{i \bmod k}}$).
   Na przykład ciąg $\{1; 1; 0; 1\}$ wyznacza ciąg Bonifacego:
   $0, 1, 1, 1, 2, 3, 5, 7, 10, 15, 25, 35, 50, \ldots$.
   Napisz funkcję `int bonifacy(int n, int k, int b[])`, która dla liczby naturalnej n ($n \geq 0$) oraz co najmniej trzyelementowej zerojedynkowej tablicy $\{b_0, \ldots, b_{k-1}\}$ obliczy n-tą liczbę wyznaczonego przez tę listę ciągu Bonifacego.
5. (*) Napisz program wybierający element dominujący z tablicy większościowej (liczb całkowitych); uzasadnij jego poprawność.
6. Napisz funkcję `std::vector>podziel(int k, const std::vector x)` , która dla $k > 0$ i listy $[x_1; \ldots; x_n]$ podzieli ją na listę list postaci:

$$\{\{x_1, x_{k+1}, x_{2k+1}, \ldots\}, \{x_2, x_{k+2}, x_{2k+2}, \ldots\}, \ldots; \{x_k, x_{2k}, x_{3k}, \ldots\}\}$$

   Przykład: `podziel(3, {1, 2, 3, 4, 5, 6, 7}) = {{1, 4, 7}, {2, 5}, {3, 6}}\)`.
7. Napisz procedurę `int* podziel(int n, int a[])`, która dla danej tablicy postaci $\{a_0, a_2, \ldots, a_{n-1}\}$, zawierającej permutację zbioru $\{0, 1, 2, \ldots, n-1\}$, znajdzie jej podział na jak największą liczbę fragmentów postaci:

?

$$\{a_0, a_1, \ldots, a_{k_1-1}\}, \{a_{k_1}, a_{k_1+1}, \ldots, a_{k_2-1}\}, \ldots, \{a_{k_{m-1}}, a_{k_{m-1}+1}, \ldots, a_{k_m-1}\}$$

   taki że:

$$\begin{aligned} k_m &= n, \\ \{a_0, a_1, \ldots, a_{k_1-1}\} &= \{0, 1, 2, \ldots, k_1 - 1\} \quad (\text{równość zbiorów}), \\ \{a_{k_1}, a_{k_1+1}, \ldots, a_{k_2-1}\} &= \{k_1, k_1 + 1, \ldots, k_2 - 1\}, \\ &\vdots \\ \{a_{k_{m-1}}, a_{k_{m-1}+1}, \ldots, a_{k_m-1}\} &= \{k_{m-1}, k_{m-1} + 1, \ldots, k_m - 1\}. \end{aligned}$$

   Wynikiem powinna być tablica indeksów $\{k_1, k_2, \ldots, k_m\}$. Zauważ, że nie musimy znać wielkości tablicy wynikowej, ponieważ jej ostatni element jest równy $k_m = n$. Dla $n = 0$ poprawnym wynikiem jest tablica zawierająca jedną liczbę $k_m = n = 0$.
   Przykład: $\texttt{podziel}(11, \{2, 3, 1, 6, 5, 4, 7, 9, 10, 11, 8\}) == \{3, 6, 7, 11\}$, co odpowiada podziałowi: $\{2, 3, 1\}, \{6, 5, 4\}, \{7\}, \{9, 10, 11, 8\}$.

8. Rozważmy następującą metodę kompresji ciągów liczb całkowitych: Jeżeli w oryginalnym ciągu ta sama liczba powtarza się kilka razy z rzędu, to jej kolejne wystąpienia reprezentujemy za pomocą jednej tylko liczby. Konkretnie, i powtórzeń liczby k reprezentujemy w ciągu skompresowanym jako $2^{i-1} \cdot (2 \cdot k - 1)$. Napisz procedurę `int* dekompresuj(int n, int t[])` dekompresującą zadaną tablicę t zawierającą n liczb całkowitych. Możesz założyć, że lista skompresowana nie zawiera zer. Przykład: `dekompresuj(6, {1, 3, 3, 9, 42, 3}) == {1, 2, 2, 5, 11, 11, 2}`.
9. Napisz procedurę `int trojki(int size, int tab[])`, która dla zadanej tablicy dodatnich liczb całkowitych, uporządkowanej ściśle rosnąco, policzy ile jest takich trójek (a, b, c) liczb z danej tablicy, że:
   - $a < b < c$,
   - liczby a, b i c spełniają nierówność trójkąta, czyli $c < a + b$.
10. (*) [XIII OI] Mały Jaś dostał od rodziców na urodziny nową zabawkę, w której skład wchodzą rurka i krążki. Rurka ma nietypowy kształt -- mianowicie jest to połączenie pewnej liczby walców (o takiej samej grubości) z wyciętymi w środku (współosiowo) okrągłymi otworami różnej średnicy. Rurka jest zamknięta od dołu, a otwarta od góry. Krążki w zabawce Jasia są walcami o różnych średnicach i takiej samej grubości co walce tworzące rurkę. Jaś wymyślił sobie następującą zabawę. Mając do dyspozycji pewien zestaw krążków zastanawia się, na jakiej głębokości zatrzymałby się ostatni z nich, gdyby wrzucać je kolejno do rurki centralnie (czyli dokładnie w jej środek). Każdy kolejny krążek po wrzuceniu spada dopóki się nie zaklinuje (czyli nie oprze się o walec, w którym wycięty jest otwór o mniejszej średnicy niż średnica krążka), albo nie natrafi na przeszkodę w postaci innego krążka lub dna rurki. Napisz procedurę `int wrzucaj(int n_rurka, int rurka[], int n_krazki, int krazki[])`, która na podstawie listy średnic otworów w kolejnych walcach tworzących rurkę, oraz listy średnic kolejno wrzucanych krążków obliczy ile krążków zmieści się w rurce.
11. System -2-owy jest podobny do systemu binarnego, ale podstawą tego systemu jest liczba -2. Są dwie możliwe cyfry: 0 i 1. Ciąg cyfr postaci $c_k c_{k-1} \ldots c_1 c_0$ reprezentuje liczbę $\sum_{i=0}^{k} c_i (-2)^i$. Na przykład, liczbę 42 reprezentujemy jako 1111110_{-2}. Liczby w takim systemie możemy reprezentować jako tablice cyfr, w kolejności od mniej do bardziej znaczących, zakończoną liczbą -1.
    1. Napisz procedurę `long long int int_of_neg2(int cyfry[])`, która przekształca reprezentację w systemie -2-kowym w liczbę całkowitą.
    2. (*) [V OI, zadanie Bankomaty] Napisz procedurę `bool neg2_of_int(long long int n, int wynik[], int size)`, która przekształca daną liczbę całkowitą w jej reprezentację w systemie -2-kowym i umieszcza ją w tablicy `wynik`. Wartość zwracana określa, czy reprezentacja zmieściła się w danej tablicy.
    3. (*) Napisz procedurę `bool inc(int cyfry[], int wynik[], int size)`, która dla danej reprezentacji liczby x, wyznaczy reprezentację liczby $x + 1$ w systemie -2-kowym i umieści ją w tablicy `wynik`. Wartość zwracana określa, czy reprezentacja zmieściła się w danej tablicy.
    4. Napisz procedurę `bool inv(int cyfry[], int wynik[], int size)`, która dla danej reprezentacji liczby x, wyznaczy reprezentację liczby $-x$ w systemie -2-kowym i umieści ją w tablicy `wynik`. Wartość zwracana określa, czy reprezentacja zmieściła się w danej tablicy.
    5. Napisz procedurę `bool inv(int a[], int b[], int wynik[], int size)`, która dla danych dwóch reprezentacji liczb całkowitych w systemie -2-ym obliczy reprezentację ich sumy i umieści ją w tablicy `wynik`. Na przykład wynikiem dodawania `{1, 0, 1, 0, 0, 1, 1, -1}` i `{1, 0, 1, -1}`) == {0, 1, 1, 1, 1, 1, 1, -1}.
12. (*) Tablica liczb całkowitych (rozmiaru n) zawiera pewną permutację zbioru liczb $\{0, \ldots, n-1\}$. Napisz procedurę `bool next(int a[], int size)`, która przekształci tablicę `a` w następną, w porządku leksykograficznym, permutację zbioru liczb $\{0, \ldots, n-1\}$. Jeżeli dana tablica jest ostatnią w porządku leksykograficznym permutacją, należy przekształcić ją w pierwszą w porządku leksykograficznym permutację. W takim przypadku, funkcja powinna zwrócić `false`. W pozostałych przypadkach powinna zwrócić `true`.

13. Jaś pracuje wykonując zlecenia, które dostaje od swojego szefa. Wykonuje je w kolejności ich otrzymania. Wykonanie każdego zlecenia zajmuje mu określony czas. Masz daną tablicę par: moment otrzymania zlecenia, czas potrzebny na jego wykonanie, posortowaną ściśle rosnąco po momentach otrzymania zleceń. Napisz procedurę `nakiedy`, która dla każdego zlecenia wyznaczy ile czasu upłynie od momentu otrzymania zlecenia do momentu jego wykonania.
```
struct para {
    int zlecenie, wykonanie;
}
int* nakiedy(sturct para zlecenia[], int size);
```
Na przykład: `zlecenia ({(-1, 1), (2, 2), (3, 3), (4, 2), (10, 2)}, 5) == {1, 2, 4, 5, 2}`.
14. Dane są dwie tablice liczb całkowitych `a` i `b`. Napisz funkcję `bool podciag(int a[], int size_a, int b[], int size_b)` sprawdzającą, czy `a` jest podciągiem (niekoniecznie spójnym) `b`, tzn. czy usuwając z tablicy `b` pewne elementy można uzyskać tablicę `a`.
15. Dana jest tablica liczb całkowitych `a`. Napisz funkcję `int plateau(int a[], int size)`, która wyznacza długość najdłuższego spójnego fragmentu stałego w tablicy `a`.
16. (*) Ciąg różnicowy ciągu $(x_1, x_2, \ldots, x_n)$, to ciąg postaci $(x_2 - x_1, x_3 - x_2, \ldots, x_n - x_{n-1})$.
Ciągi reprezentujemy jako struktury postaci:
```
typedef struct {
  int dl;
  double el[];
} ciag;
```
Napisz procedurę, która oblicza ciąg różnicowy danego ciągu.
17. (*) Napisz procedurę, która dla danego ciągu wyznaczy: dany ciąg, jego ciąg różnicowy, ciąg różnicowy ciągu różnicowego, itd. aż do pustego ciągu. Zdefiniuj odpowiedni typ i napisz funkcję wyznaczającą ciąg ciągów różnicowych.
18. Wielomian postaci $a_0 \cdot x^0 + a_1 \cdot x^1 + \cdots + a_n \cdot x^n$ reprezentujemy w postaci struktury:
```
typedef struct {
  int stopien;
  int wsp[];
} wielomian;
```
postaci $\{.\texttt{stopien} = n, .\texttt{wsp} = \{a_0, a_1, \ldots, a_n\}\}$. Napisz procedurę `wielomian mnóż(wielomian u, wielomian w)` mnożącą dwa wielomiany.
19. Przy windzie (na parterze) ustawiła się kolejka n ludzi, którzy chcą wjechać windą (wszyscy do góry). Jednorazowo windą mogą jechać osoby o łącznym ciężarze nie większym niż w. Do windy zawsze wsiada maksymalnie dużo osób z początku kolejki, których łączny ciężar nie przekracza w. Napisz procedurę `int winda(int w, int n, int ludzie[])`, która mając dane w, n, oraz listę ciężarów kolejnych osób czekających na windę policzy minimalną liczbę kursów windy potrzebnych do obsłużenia czekających. Możesz założyć, że ciężary osób czekających na windę są liczbami dodatnimi, nie większymi od w.

Last modified: Sunday, 27 November 2022, 1:19 PM

Get the mobile app