PPJ

Powtórka - zadania

20 stycznia 2024

I. Dana jest jednowymiarowa tablica liczb całkowitych. Zaimplementuj fragment kodu, który wyświetla element (tablicy) najbliższy średniej arytmetycznej elementów tablicy.

II. Dana jest tablica zmiennych typu `char` przechowująca napis `Ala ma kota`. Napisz metodę, która policzy ile razy wystąpiły znaki składające się na ciąg dostarczony jako argument.

III. Utwórz metodę `merge` przyjmującą dwie tablice elementów typu `int` oraz zwracającą ich połączenie w postaci nowej tablicy, której elementy będą wstawiane naprzemiennie (raz z jednej tablicy, raz z drugiej). Gdy elementy którejś z tablic się wyczerpią, dopełnij tablicę wynikową pozostałymi elementami z drugiej tablicy.

IV. Utwórz metodę `trans` przyjmującą tablicę dwuwymiarową kwadratową elementów typu `int` oraz zwracającą nową tablicę, powstającą przez zamianę miejscami elementów (z wejściowej tablicy) położonych symetrycznie względem lewej przekątnej.

V. Utwórz metodę `rotate` przyjmującą tablicę dwuwymiarową prostokatną elementów typu `int` oraz zwracającą jej postać obróconą o 90^o zgodnie z ruchem wskazówek zegara. Dla tablicy wejściowej:

```
1  1  3  2
2  4  6  5
3  8  7  9
4  0  3  1
5  6  9  5
```

metoda powinna zwrócić tablicę:

```
1  6  0  8  4  1
2  9  3  7  6  3
3  5  1  9  5  2
```

VI. Utwórz statyczną metodę rekurencyjną przyjmującą posortowaną rosnąco tablicę liczb typu `int`, indeksy początkowy/końcowy i podaną wartość `value`

```
int binSearch(int arr[], int begin, int end, int value)
```

oraz zwracającą indeks elementu `value` w tablicy `arr` od pozycji `begin` do pozycji `end` (lub -1, jeśli takiej wartości nie ma) metodą `wyszukiwania binarnego`.

Metodę wyszukiwania binarnego można scharakteryzować następująco: wyszukiwanie zaczyna się od porównania `value` z elementem środkowym tablicy. Ponieważ elementy tablicy są posortowane rosnąco, więc jeśli `value` jest mniejszy niż element środkowy, to wystarczy `value` wyszukać w pierwszej połowie tablicy, a jeśli jest większy to w drugiej, itd.

VII. Dany jest nagłówek metody:

```
1 public static void swap(int[] tab, int source, int destination)
```

Uzupełnij ciało tej metody, tak aby wskazane przez parametry `source` i `destination` elementy tablicy zostały zamienione miejscami.

VIII. Algorytm sortowania bąbelkowego polega na porównaniu dwóch sąsiadujących elementów tablicy i gdy element poprzedzający jest większy niż następujący, następuje zamiana tych elementów miejscami. W ten sposób zagwarantujemy, że po jednym przejściu całej tablicy, największa wartość znajdzie się na miejscu o ostatnim indeksie w tablicy. Operacje powtarzamy dla pozostałej, ciągle nieposortowanej części tablicy. Proces powtarzamy aż do pełnego posortowania tablicy.

Zaimplementuj metodę statyczną, rekurencyjną

```
1 void bubleSortRe(int[] arr, int n)
```

realizującą algorytm sortowania bąbelkowego dla `n` pierwszych elementów tablicy `arr`.

IX. Kolejka jest strukturą danych, w której nowe dane dopisywane są na końcu kolejki, a pobierane są z początku kolejki.

Utwórz własną implementację `MyQueue`, która będzie:

- zawierała metodę `void put(...)` pozwalającą na dodanie do kolejki elementu;
- zawierała metodę `... get()` zwracającą element pobrany z kolejki.

Zrealizuj `MyQueue` wykorzystując tablicę.

Dane są dwa koszyki (oznaczone jako A i B) zawierające obiekty klasy `String`. W koszyku A znajdują się ciągi znaków: PSG, Atletico Madryt, Sporting CP, Inter, Benfica, Villarreal, RB Salzburg, Chelsea, natomiast w koszyku B znajdują się: Manchester City, Liverpool, Ajax Amsterdam, Real Madryt, Bayern Monachium, Manchester United, Lille, Juventus. Utwórz dwie struktury reprezentujące koszyki i naprzemiennie wyciągaj z nich elementy, które zostaną umieszczone w kolejce. Gdy koszyki będą już puste, wówczas rozpocznij wyciąganie elementów z kolejki. Wyciągaj sekwencyjnie po dwa elementy i powstałącą parę wyświetl na ekranie. Powtarzaj sekwencję aż do opróżnienia kolejki.