Sortowanie kubełkowe w języku Java

- Wprowadzenie
- Film samouczek
- Przeczytaj
- Sprawdź się
- Dla nauczyciela

Źródło: Amy Parkes, domena publiczna.

Jak już wiemy, sortowanie kubełkowe to jeden z najbardziej intuicyjnych algorytmów sortowania, którego używamy w życiu codziennym. Ogólnie mówiąc, polega on na umieszczeniu pewnych wartości w odpowiadających im kubełkach, a następnie wyjęciu ich stamtąd w określonej, pożądanej przez nas kolejności. W tym e-materiale dowiesz się, jak zaimplementować ten algorytm w języku Java.

Implementacje sortowania kubełkowego w innych językach programowania zostały omówione w e-materiałach:

- Sortowanie kubełkowe w języku C++,
- Sortowanie kubełkowe w języku Python.

Więcej zadań? Przejdź do: Sortowanie kubełkowe – zadania maturalne.

> **Twoje cele**
>
> - Zaimplementujesz w języku Java algorytm sortowania kubełkowego.
> - Przeanalizujesz krok po kroku program realizujący algorytm sortowania kubełkowego dla liczb należących do danego zakresu.
> - Scharakteryzujesz złożoność pamięciową algorytmu sortowania kubełkowego.

Film samouczek

Polecenie 1

Napisz program sortujący dane za pomocą metody kubełkowej. Przetestuj jego działanie dla wyników ankiety, dotyczącej opinii o traktacie lizbońskim. Ankieta została opracowana przez CBOS i przeprowadzona po referendum w Irlandii w lipcu 2008 roku.

Ankietowanym zadano pytanie: Jaka jest pana/pani opinia o traktacie lizbońskim, po referendum w Irlandii?

Wyniki ankiety:

Raczej dobra – 18 %.

Zdecydowanie dobra – 3 %.

Trudno jednoznacznie powiedzieć – 29%.

Zdecydowanie zła – 21%.

Raczej zła – 29%.

Specyfikacja problemu:

Dane:

- `liczbaElementow` – liczba naturalna dodatnia; liczba elementów do posortowania
- `dane` – tablica składająca z liczby elementów równej `liczbaElementow` zawierająca liczby naturalne z przedziału $[0, 100]$; tablica liczb do posortowania
- `wartoscMin` – liczba naturalna; najmniejsza z wartości do posortowania odczytana z tablicy `dane`
- `wartoscMax` – liczba naturalna; największa z wartości do posortowania odczytana z tablicy `dane`

Wynik:

- `dane` – posortowana niemalejąco tablica

```
public static void main(String[] args){
    // Tutaj dodaj własny kod. Użyj funkcji
    // System.out.print();
  }
```

```

```

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/RA0wrDfCnDwSx

Lekcja poświęcona sortowaniu kubełkowemu w języku Java

Plik o rozmiarze 897.00 B w języku polskim

Polecenie 3

Opracuj notatkę podsumowującą najważniejsze informacje przedstawione w filmie.

Przeczytaj

Sortowanie kubełkowe – implementacja w języku Java

Algorytm sortowania kubełkowego polega na zliczeniu wystąpień danych wartości w odpowiednich kubełkach. Na podstawie liczby elementów znajdujących się w danym kubełku należy utworzyć zbiór wynikowy.

Zbiorem wejściowym i wyjściowym (wynikowym) w języku Java może być dowolny kontener. W tym e-materiale skorzystamy z tablic.

Załóżmy, że tablica wejściowa została już wczytana i utworzona.

Specyfikacja problemu:

Dane:

- `tabWejsciowa` – tablica wejściowa zawierająca liczby całkowite
- `minLiczba` – liczba całkowita; najmniejszy element w podanej tablicy
- `maxLiczba` – liczba całkowita; największy element w podanej tablicy

Wynik:

- `tab` – posortowana niemalejąco tablica liczb całkowitych

> **Już wiesz**
>
> Długość tablicy odczytujemy za pomocą atrybutu `length`.

```
1 static int[] przykladTab = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
2 System.out.println(przykladTab.length); // Wypisze 10 na wyjści
```

Algorytm sortowania kubełkowego stosujemy najczęściej dla typów całkowitoliczbowych. Jednak możliwe jest wykorzystanie go (po zmianach) do sortowania liczb rzeczywistych.

Długość tablicy wynikowej nie powinna być krótsza od długości tablicy wejściowej. Jeśli w tablicy wejściowej są jedynie liczby dodatnie, można odnaleźć największą liczbę, a następnie utworzyć tablicę o długości tej liczby + 1. Nie jest to jednak optymalne rozwiązanie.

> **Przykład 1**
>
> Jeśli w tablicy wejściowej znajdowałyby się liczby: `{1000, 999, 1002, 1002, 999}`, to tworząc tablicę wynikową jedynie na podstawie największej z tych liczb, otrzymalibyśmy tablicę zawierającą 1003 elementy, przy czym potrzebujemy jedynie

> komórek na unikalne kubełki odpowiadające za wartości: `999`, `1000`, `1002`. Oznacza to, że mielibyśmy nadmiarowo zaalokowane 1000 komórek pamięci. Jednak liczbę nadmiarowych komórek da się w tym algorytmie ograniczyć.

Z przedstawionego przykładu wynika, że można utworzyć mniejszą tablicę wynikową. W tym celu odnajdujemy najmniejszą liczbę w tablicy wejściowej i odejmujemy ją od największej liczby.

> **Przykład 2**
>
> Użyjmy tablicy z poprzedniego przykładu: `{1000, 999, 1002, 1002, 999}`. Tym razem odnajdujemy największą i najmniejszą liczbę.
> `minLiczba = 1002`
> `minLiczba  = 999`
> `1002 - 999 = 3`
>
> W tablicy wejściowej znajdują się trzy unikalne elementy. Należy jednak pamiętać, że tablica wynikowa, którą chcemy utworzyć, powinna mieć co najmniej cztery elementy. Algorytm nie uwzględnia faktu, że w zbiorze nie ma liczby `1001` i przypisuje tej liczbie własny kubełek.
>
> > **Ważne!**
> >
> > W tym przykładzie każda z liczb z przedziału `<999, 1002>` zajmuje jeden kubełek, numerowany od 0 do 3 włącznie.
>
> Liczba 3 jest więc ostatnim indeksem, w którego komórce będziemy mogli coś zapisać, żeby nie wyjść poza przedział tablicy.
>
> Deklaracja tablicy wygląda następująco:

```
1 int[] przyklad = new int[maxLiczba - minLiczba + 1];
```

W optymalnym algorytmie sortowania kubełkowego znajdujemy najmniejszą oraz największą liczbę, aby utworzyć tablicę wynikową o optymalnej długości.

Nie wspomnieliśmy jeszcze o sytuacji, w której sortuje się liczby ujemne. Przeanalizujmy przykład, w którym obliczymy długość tablicy w takim przypadku.

> **Przykład 3**
>
> Dla tablicy wejściowej `{-5, -4, -3, -3, -4}` istnieje pozorny problem, polegający na tym, że największą liczbą z tego zbioru jest liczba `-3`, a w języku Java nie można utworzyć tablicy w której indeks jest liczbą ujemną.
>
> Długość tablicy wynikowej obliczymy, korzystając ze wzoru `maxLiczba - minLiczba`. Odnajdujemy najmniejszą liczbę, czyli `-5`, a następnie odejmujemy ją od największej

liczby: `-3 - (-5) = 2`.
Otrzymujemy informację, że największym dopuszczalnym indeksem tablicy będzie indeks o numerze 2.
Postępujemy więc analogicznie do przykładu 2. Zwiększamy otrzymaną liczbę o 1, a następnie deklarujemy tablicę w następujący sposób:

```
1 int[] przyklad = new int[maxLiczba - minLiczba + 1];
```

Rzeczywiście, w sprawdzanej przez nas tablicy były 3 unikalne elementy. Wyznaczanie długości tablicy zliczającej w algorytmie nie ulega więc modyfikacji względem poprzedniego przykładu.

Algorytm nie zmienia się również w przypadku zbioru składającego się z liczb ujemnych i dodatnich. Przeanalizujmy ostatni już przykład wyznaczania rozmiaru tablicy pomocniczej.

Przykład 4

Dla tablicy `{-5, 5}` mamy następującą sytuację:
`maxLiczba = 5`
`minLiczba = -5`
`5 - (-5) = 10`
Ponownie do wyniku działania dodajemy 1, uzyskując na koniec tablicę kubełków składającą się z 11 elementów.

Możemy zauważyć tu marnotrawstwo pamięci. W omawianym przykładzie może wydawać się ono nieznaczące, jednak dla zbioru złożonego z liczb: $\{-2^{63},\ 2^{63}\}$ mogłoby stanowić duże obciążenie dla programu.

Wynika to z faktu, że optymalny algorytm sortowania kubełkowego ma złożoność pamięciową $O(max\ -\ min\ +\ 1)$.

Ważne!

Dla dużych rozpiętości danych algorytm sortowania kubełkowego jest nieoptymalny pamięciowo.

Zgodnie z założeniem przyjętym na początku tej sekcji – tablica wejściowa została już wprowadzona do programu. Rozwiązanie zaimplementujemy w ramach klasy `SortowanieKubelkowe`, która zawiera metodę `main`.

Lista kroków algorytmu:

1. Znajdź największą (`maxLiczba`) i najmniejszą (`minLiczba`) wartość z danego zbioru.
2. Stwórz tablicę pomocniczą (kubełków) o długości `maxLiczba - minLiczba + 1`.

3. Zlicz wystąpienia każdej z wartości, a następnie przypisz obliczone liczby wartości do odpowiednich kubełków.
4. Począwszy od kubełka o najmniejszej/największej wartości wykonuj kroki 5-6.
5. Jeśli kubełek nie jest pusty, to do tablicy wynikowej wpisz wartość (`indeksKubełka + minLiczba`) do pierwszego niewypełnionego jeszcze indeksu.
   Jeśli kubełek jest pusty, przejdź do kolejnego kubełka.
   Jeśli nie ma już więcej kubełków, przejdź do kroku 7.
6. Zdekrementuj liczbę elementów w aktualnym kubełku i wróć do kroku 5.
7. Zwróć tablicę wynikową `tab`.

Przejdźmy do implementacji algorytmu. Najpierw tworzymy metodę pomocniczą do odnajdywania największej i najmniejszej liczby w danej tablicy. Zostanie ona zaimplementowana w ramach wspomnianej wcześniej klasy.

```
public static int[] znajdzMinMax(int[] tab){
    int minLiczba = tab[0];
    int maxLiczba = tab[0];
    for (int i = 1; i < tab.length; i++) {
        if (minLiczba > tab[i]) minLiczba = tab[i];
        if (maxLiczba < tab[i]) maxLiczba = tab[i];
    }
    return new int[]{minLiczba, maxLiczba};
}
```

W przytoczonej implementacji skorzystaliśmy ze stałych `Integer.MAX_VALUE` i `Integer.MIN_VALUE`. Stałe te pochodzą z klasy `Integer`, dostępnej w ramach `Java.lang package`. Aby ich użyć, nie trzeba dołączać żadnych pakietów ani klas.

```
public class SortowanieKubelkowe {
    static int[] tabWejsciowa = {999, 888, 222, 111, -444};
}
```

Metoda sortująca przedstawia się następująco:

```
public static int[] posortuj(int [] tab) {
    int[] kubelki;
    int[] MinMax = znajdzMinMax(tab); //odnalezienie minimalnej i
    int min = MinMax[0]; // przypisanie minimalnej liczby dla czy
    int max = MinMax[1]; // przypisanie maksymalnej liczby dla cz
```

```
    kubelki = new int[max - min + 1]; // utworzenie tablicy o dlu

    for (int i = 0; i < tab.length; i++) {
        kubelki[tab[i] - min]++; // zliczenie liczb do odpowiedni
    }

    int j = 0;
    int i = 0;

    while (i < tab.length) {
        if (kubelki[j] != 0) { // jeśli kubełek nie pusty
            tab[i] = min + j; // wypełnij element spod indeksu i
            i++;
            kubelki[j]--; // usuń element z kubełka
        } else {
            j++; // przejdź do nastepnego kubełka
        }
    }

    return tab;
}
```

Cały kod przyjmuje postać:

```
import java.util.Arrays;

public class SortowanieKubelkowe {
    static int[] tabWejsciowa = {999, 888, 222, -444, 111};

    public static int[] znajdzMinMax(int[] tab){
        int minLiczba = tab[0];
        int maxLiczba = tab[0];
        for (int i = 1; i < tab.length; i++) {
            if (minLiczba > tab[i]) minLiczba = tab[i];
            if (maxLiczba < tab[i]) maxLiczba = tab[i];
        }
        return new int[]{minLiczba, maxLiczba};
    }

    public static int[] posortuj(int [] tab) {
        int[] kubelki;
```

```
        int[] MinMax = znajdzMinMax(tab);
        int min = MinMax[0];
        int max = MinMax[1];
        kubelki = new int[max - min + 1];

        for (int i = 0; i < tab.length; i++) {
            kubelki[tab[i] - min]++;
        }

        int j = 0;
        int i = 0;

        while (i < tab.length) {
            if (kubelki[j] != 0) {
                tab[i] = min + j;
                i++;
                kubelki[j]--;
            } else {
                j++;
            }
        }

        return tab;
    }

    public static void main(String[] args) {
        int[] tabWyjsc = tabWejsciowa.clone();
        System.out.println(Arrays.toString(tabWejsciowa));
        System.out.println(Arrays.toString(posortuj(tabWyjsc)));
    }
}
```

Metoda `Arrays.toString()` pozwala wypisać tablicę w postaci łańcucha znaków. Program po wykonaniu wyświetli następujące tablice:

```
[999,888,222,-444,111]
[-444,111,222,888,999]
```

> **Ważne!**
>
> Aby użyć klasy `Arrays` i metod w niej dostępnych, należy najpierw zaimportować tę klasę poprzez:
>
> ```
> 
> ```

```
1 import java.util.Arrays;
```

Sortowanie kubełkowe dla liczb rzeczywistych

Liczby rzeczywiste również można sortować za pomocą algorytmu sortowania kubełkowego, jednak różni się on od wcześniej przedstawionego.

W omawianych dotychczas przykładach sortowania liczb całkowitych algorytm przyjmował postać uproszczonego sortowania przez zliczanie.

W przypadku algorytmu dla liczb rzeczywistych tworzone są kubełki dla pewnych ściśle zdefiniowanych zakresów. Przykładowo, w każdym kubełku mogą znajdować się liczby z przedziału $< x,\ x\ +\ 1)$ dla liczb nieujemnych. Oznacza to, że jeżeli pod x podstawimy liczbę 3, to w danym kubełku znajdą się liczby z przedziału $< 3,\ 4)$. Dla liczb ujemnych w każdym kubełku znajdują się liczby z przedziału $(x - 1,\ x >$. Gdy pod x podstawimy -5, w kubełku znajdą się liczby z przedziału $(-6,\ -5 >$.

Wiedząc, do jakiego przedziału należą sortowane liczby, możemy łatwo określić przedziały poszczególnych kubełków. Dzielimy długość przedziału przez liczbę kubełków – w ten sposób powstaje długość jednego przedziału.

Zastosujemy przedział wielkości 1. Oznacza to, że w danym kubełku dla liczb nieujemnych znajdą się liczby z przedziału $< x,\ x\ +\ 1)$, natomiast dla liczb ujemnych przedział będzie prezentował się następująco: $(x - 1,\ x >$.

> **Przykład 5**
>
> Jeżeli pod x podstawimy liczbę 5, to przedział kubełka liczby znajdującej się w tym zakresie wyniesie $< 5,\ 6)$.
>
> Jeżeli pod x podstawimy liczbę -5, to przedział kubełka będzie miał zakres $(-6,\ -5 >$.

Zapoznaj się z kodem. Wszystkie elementy tablicy umieszczane są w kubełkach. Kubełki mają wcześniej określone przedziały. Program na końcu wypisuje zawartości kubełków.

```
1 import java.util.ArrayList;
2
3 public class Main {
4     static int[] znajdzMinMax(double[] tab){
5         int max = (int)tab[0];
6         int min = (int)tab[0];
7         for(int i = 1; i < tab.length; i++){
8             if (max < (int)tab[i] ) max = (int)tab[i];
9             if (min > (int)tab[i] ) min = (int)tab[i];
```

```
10          }
11          return new int[] {min, max};
12      }
13
14
15      public static void main(String[] args) {
16          double tabWejsciowa[] = { 3.5, 2, 3, 4, -4, 2, 3, 2, 0, -
17          int [] MinMax = znajdzMinMax(tabWejsciowa);
18          int max = MinMax[1];
19          int min = MinMax[0];
20          int dlugoscTab = max - min + 1;
21          ArrayList<ArrayList<Double>> tablicaTablic = new ArrayLis
22          for(int i = 0; i<dlugoscTab; i++){
23              tablicaTablic.add(new ArrayList<Double>());
24          }
25          for (int i = 0; i < tabWejsciowa.length; i++){
26              double wartosc = tabWejsciowa[i];
27              int ktoryKubelek = (int)wartosc - min;
28              tablicaTablic.get(ktoryKubelek).add(wartosc);
29          }
30          for (int i = 0; i < dlugoscTab; i++){
31              if (!(tablicaTablic.get(i).size() == 0)){
32                  System.out.print("Kubelek numer " + i + " : ");
33                  for (int j = 0; j < tablicaTablic.get(i).size();
34                      System.out.print(tablicaTablic.get(i).get(j)
35                  }
36                  System.out.println();
37              }
38          }
39      }
40  }
```

Zwróć uwagę na **21. linię** kodu.

Tworzymy pustą tablicę zawierającą wiele tablic, które z kolei zawierają obiekty klasy `Double`. Nie używamy w tym miejscu słowa kluczowego `double`, ponieważ interfejsy nie działają z typami prymitywnymi.

```
1  for(int i = 0; i<dlugoscTab; i++){
2      tablicaTablic.add(new ArrayList<Double>());
3  }
```

W przedstawionym fragmencie kodu implementujemy wypełnienie tablicy głównej tablicami przechowującymi obiekty klasy `Double`.

```
for (int i = 0; i < tabWejsciowa.length; i++){
        double wartosc = tabWejsciowa[i];
        int ktoryKubelek = (int)wartosc - min;
        tablicaTablic.get(ktoryKubelek).add(wartosc);
}
```

Następnie wszystkie komórki tablicy wejściowej wczytywane są do odpowiednich kubełków w nieuporządkowany sposób.

```
for (int i = 0; i < dlugoscTab; i++){
        if (!(tablicaTablic.get(i).size() == 0)){
            System.out.print("Kubelek numer " + i + " : ");
            for (int j = 0; j < tablicaTablic.get(i).size(); j++)
                System.out.print(tablicaTablic.get(i).get(j) + ";
            }
            System.out.println();
        }
}
```

Ostatni fragment odpowiada za wypisywanie zawartości tablicy tablic.

Przykładowy wynik wykonania kodu:

```
Kubelek numer 0 : -5.5;
Kubelek numer 1 : -4.0;
Kubelek numer 5 : 0.0;
Kubelek numer 7 : 2.0; 2.0; 2.0;
Kubelek numer 8 : 3.5; 3.0; 3.0;
Kubelek numer 9 : 4.0;
```

W kolejnym kroku należy posortować elementy poszczególnych niepustych kubełków. Elementy wypisujemy w kolejności od najmniejszego do największego. Możemy w tym celu użyć dowolnego algorytmu sortującego. W omawianym przykładzie posłużymy się algorytmem sortowania bąbelkowego.

Wyjaśnijmy najpierw, w jakim celu dzielimy elementy na kubełki, skoro i tak skorzystamy z innego algorytmu sortowania.

Przeanalizujmy sytuację, w której mamy 100 kubełków – w każdym z nich znajduje się 10 elementów, co oznacza, że mamy w sumie 1000 elementów do posortowania.

Gdybyśmy do posortowania tych elementów zastosowali jedynie sortowanie bąbelkowe, algorytm miałby złożoność obliczeniową rzędu n^2, czyli 1000^2, czyli 1000000.

W rozwiązaniu polegającym na osobnym sortowaniu każdego z kubełków złożoność obliczeniowa algorytmu sortowania bąbelkowego wyniesie $100 * 10^2$, czyli 10000

Porównajmy ze sobą dwie liczby:

$$1000^2 > 100 \cdot 10^2 \tag{1}$$

Możemy zauważyć, że zastosowanie sortowania bąbelkowego w tym przypadku wymaga większej liczby operacji.

Należy jednak pamiętać, że nie zawsze liczba elementów w każdym z kubełków będzie taka sama.

> **Ciekawostka**
>
> Dodatkowo (z racji tego, że kubełki znajdują się w oddzielnych obszarach pamięci) można wykorzystać wielowątkowość do sortowania poszczególnych kubełków, przyspieszając tym samym wykonanie całego programu, poprzez np. tworzenie wątku na każdy niepusty kubełek.
>
> Więcej informacji na temat wielowątkowości znajdziesz w e-materiałach Programowanie obiektowe – projekt, etap IV oraz Programowanie obiektowe – projekt, etap VI.

Do napisanego wcześniej programu dodajemy metodę sortującą z wykorzystaniem sortowania bąbelkowego.

> **Dla zainteresowanych**
>
> Spróbuj samodzielnie zaimplementować inny algorytm sortujący poszczególne kubełki.

```
1 static void sortowanieBabelkowe(ArrayList<Double> a) {
2     boolean sortuj = true;
3     while(sortuj){
4         sortuj = false;
5         for (int i = 0; i < a.size()-1; i++){
```

```
            if (a.get(i) > a.get(i+1)) {
                a.set(i, a.get(i) + a.get(i+1));
                a.set(i+1, a.get(i) - a.get(i+1));
                a.set(i, a.get(i)-a.get(i+1));
                sortuj = true;
            }
         }
    }
}
```

Po dodaniu sortowania poszczególnych kubełków cały kod wygląda następująco:

```
import java.util.ArrayList;

public class Main {
    static int[] znajdzMinMax(double[] tab){
        int max = (int)tab[0];
        int min = (int)tab[0];
        for(int i = 1; i < tab.length; i++){
            if (max < (int)tab[i] ) max = (int)tab[i];
            if (min > (int)tab[i] ) min = (int)tab[i];
        }
        return new int[] {min, max};
    }

    static void sortowanieBabelkowe(ArrayList<Double> a) {
        boolean sortuj = true;
        while(sortuj) {
            sortuj = false;
            for (int i = 0; i < a.size()-1; i++) {
                if (a.get(i) >a.get(i+1)) {
                    a.set(i, a.get(i) + a.get(i+1));
                    a.set(i+1, a.get(i) - a.get(i+1));
                    a.set(i, a.get(i)-a.get(i+1));
                    sortuj = true;
                }
            }
        }
    }


```

```
    public static void main(String[] args) {
        double tabWejsciowa[] = { 3.5, 2.5, 3, 4, -4, 2.2, 3, 2.6
        int [] MinMax = znajdzMinMax(tabWejsciowa);
        int max = MinMax[1];
        int min = MinMax[0];
        int dlugoscTab = max - min + 1;
        ArrayList<ArrayList<Double>> tablicaTablic = new ArrayLis
        for(int i = 0; i<dlugoscTab; i++){
            tablicaTablic.add(new ArrayList<Double>());
        }
        for (int i = 0; i < tabWejsciowa.length; i++) {
            double wartosc = tabWejsciowa[i];
            int ktoryKubelek = (int)wartosc - min;

            tablicaTablic.get(ktoryKubelek).add(wartosc);
        }
        //sortowanie wszystkich kubełków po kolei.
        for(int i = 0; i < dlugoscTab; i++){
            sortowanieBabelkowe(tablicaTablic.get(i));
        }
        for (int i = 0; i < dlugoscTab; i++) {
            if (!(tablicaTablic.get(i).size() == 0)) {
                System.out.print("Kubelek numer " + i + " : ");
                for (int j = 0; j < tablicaTablic.get(i).size();
                    System.out.print(tablicaTablic.get(i).get(j)
                }
                System.out.println();
            }
        }
    }
}
```

W kolejnym kroku należy zaimplementować wpisywanie elementów z kubełków do tablicy wynikowej, co można uzyskać za pomocą następującego kodu:

```
int g = 0;
double[] tabWynikowa = new double[tabWejsciowa.length];
for (int i = 0; i < dlugoscTab; i++) {
    if (!(tablicaTablic.get(i).size() == 0)) {
        for (int j = 0; j < tablicaTablic.get(i).size(); j++) {
            tabWynikowa[g++] = tablicaTablic.get(i).get(j);
```

```
        }
    }
}
for(int i = 0; i<tabWynikowa.length; i++){
    System.out.println(tabWynikowa[i]);
}
```

W przedstawionym fragmencie zaimplementowane zostało również wypisywanie zawartości tej tablicy.

Kod z tworzeniem tablicy wyjściowej wygląda następująco:

```
import java.util.ArrayList;

public class Main {
    static int[] znajdzMinMax(double[] tab){
        int max = (int)tab[0];
        int min = (int)tab[0];
        for(int i = 0; i< tab.length; i++){
            if (max < (int)tab[i] ) max = (int)tab[i];
            if (min > (int)tab[i] ) min = (int)tab[i];
        }
        return new int[] {min, max};
    }

    static void sortowanieBabelkowe(ArrayList<Double> a) {
        boolean sortuj = true;
        while(sortuj) {
            sortuj = false;
            for (int i = 0; i < a.size()-1; i++) {
                if (a.get(i) >a.get(i+1)) {
                    a.set(i, a.get(i) + a.get(i+1));
                    a.set(i+1, a.get(i) - a.get(i+1));
                    a.set(i, a.get(i)-a.get(i+1));
                    sortuj = true;
                }
            }
        }
    }


```

```
30      public static void main(String[] args) {
31          double tabWejsciowa[] = { 3.5, 2.5, 3, 4, -4, 2.2, 3, 2.6
32          int [] MinMax = znajdzMinMax(tabWejsciowa);
33          int max = MinMax[1];
34          int min = MinMax[0];
35          int dlugoscTab = max - min + 1;
36          ArrayList<ArrayList<Double>> tablicaTablic = new ArrayLis
37          for(int i = 0; i<dlugoscTab; i++){
38              tablicaTablic.add(new ArrayList<Double>());
39          }
40          for (int i = 0; i < tabWejsciowa.length; i++) {
41              double wartosc = tabWejsciowa[i];
42              int ktoryKubelek = (int)wartosc - min;
43
44              tablicaTablic.get(ktoryKubelek).add(wartosc);
45          }
46          for(int i = 0; i<dlugoscTab; i++){
47              sortowanieBabelkowe(tablicaTablic.get(i));
48          }
49          int g = 0;
50          double[] tabWynikowa = new double[tabWejsciowa.length];
51          for (int i = 0; i < dlugoscTab; i++) {
52              if (!(tablicaTablic.get(i).size() == 0)) {
53                  for (int j = 0; j < tablicaTablic.get(i).size();
54                      tabWynikowa[g++] = tablicaTablic.get(i).get(j
55                  }
56              }
57          }
58          for(int i = 0; i<tabWynikowa.length; i++){
59              System.out.println(tabWynikowa[i]);
60          }
61
62      }
63  }
```

Słownik

kontener

(ang. *container, collection*) struktura danych, która w sposób zorganizowany przechowuje pewien zbiór danych; jednym z najpowszechniejszych kontenerów jest

tablica, czyli uporządkowany zbiór o określonej z góry długości, który umożliwia dostęp do dowolnego elementu w dowolnym momencie poprzez indeks

stała

(ang. *const*) wartość, która w trakcie działania programu nie może zostać zmieniona; w języku Java stałe poprzedza się specyfikatorem `final`

typ całkowitoliczbowy

typ prymitywny, w którym nie ma wartości po przecinku; może przechowywać jedynie wartości całkowitoliczbowe, w tym wartości ujemne

wielowątkowość

cecha systemu operacyjnego, dzięki której w ramach jednego procesu może być wykonywanych kilka niezależnych wątków; wszystkie wątki w ramach tego samego procesu współdzielą tą samą przestrzeń adresową zawierającą kod programu i jego dane; wielowątkowość to także cecha procesorów, oznaczająca możliwość jednoczesnego wykonywania wielu wątków w sposób sprzętowy na pojedynczej jednostce wykonawczej – rdzeniu fizycznym

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Dokończ program wyszukujący wśród elementów tablicy wartość najmniejszą i największą.

Specyfikacja problemu:

Dane:

- `tab` – tablica zawierająca liczby całkowite z przedziału $< -99,\ 99 >$

Wynik:

- `minLiczba`, `maxLiczba` – liczby całkowite; najmniejsza oraz największa wartość spośród liczb zapisanych w tablicy `tab`

Wskazówka:

Pamiętaj o atrybucie `.length` (w implementacji tablic) oraz o stałych `MAX_VALUE`, `MIN_VALUE` z klasy `Integer`.

Swoje rozwiązanie przetestuj dla następującej tablicy:

```
1 tab = {76, 84, 79, 60, -75, 58, 94, 53, -84, 51, -4, -41, 49, -
```

Twoje zadania

1. Wypisz największą liczbę z tablicy.
2. Wypisz najmniejszą liczbę z tablicy.

Ćwiczenie 2

Napisz program, który dla zadanej tablicy zliczy wystąpienia każdego z elementów do odpowiedniego kubełka, a następnie wypisze zawartość tablicy z kubełkami.

Specyfikacja problemu:

Dane:

- `tab` – tablica zawierająca liczby całkowite z przedziału $< -99, 99 >$

Wynik:

- `kubelki` – tablica liczb całkowitych; zawartość tablicy z kubełkami

Wskazówka:

Pamiętaj o metodzie `toString()` z klasy `Arrays`.

Swoje rozwiązanie przetestuj dla następującej tablicy:

```
tab = {76, 84, 79, 60, -75, 58, 94, 53, -84, 51, -4, -41, 49, -
```

Twoje zadania

1. Poprawnie wypisz zawartość kubełków.

Ćwiczenie 3

Zaimplementuj algorytm sortowania kubełkowego i wypisz na standardowe wyjście wynik sortowania. Tablicę posortuj od największego do najmniejszego elementu.

Specyfikacja:

Dane:

- `tab` – tablica zawierająca liczby całkowite z przedziału $< -99,\ 99 >$

Wynik:

- `tab` – tablica zawierająca liczby całkowite z przedziału $< -99,\ 99 >$ posortowana nierosnąco

Wskazówka:

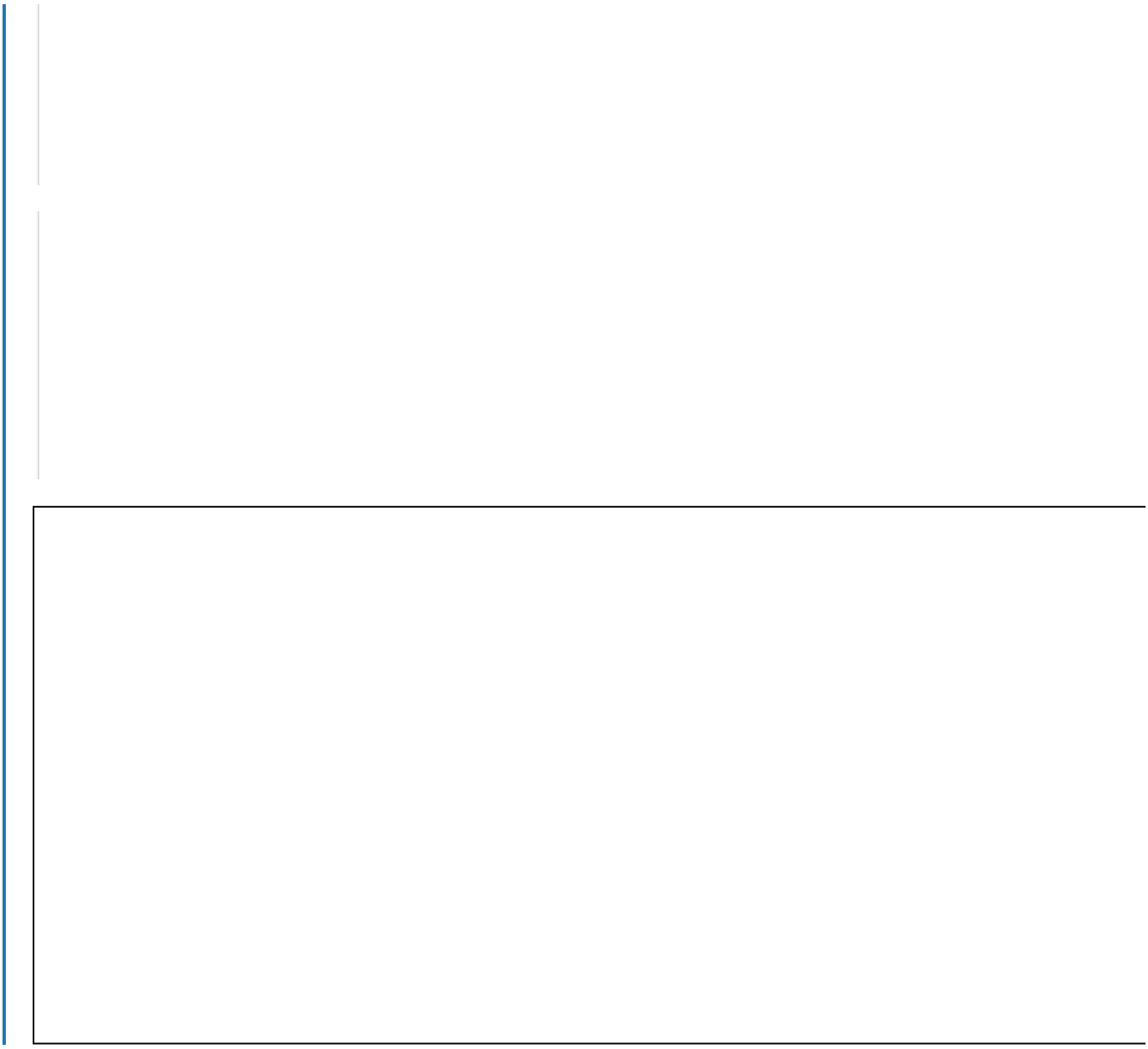
Pamiętaj o metodzie `toString()` z klasy `Arrays`.

Swoje rozwiązanie przetestuj dla następującej tablicy:

```
tab = {76, 84, 79, 60, -75, 58, 94, 53, -84, 51, -4, -41, 49, -
```

Twoje zadania

1. Posortuj tablicę nierosnąco.



Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie kubełkowe w języku Java

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

d) jednoczesnego wyszukiwania elementu najmniejszego i największego,

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Zaimplementujesz w języku Java algorytm sortowania kubełkowego.
- Przeanalizujesz krok po kroku program realizujący algorytm sortowania kubełkowego dla liczb należących do danego zakresu.
- Scharakteryzujesz złożoność pamięciową algorytmu sortowania kubełkowego.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- odwrócona klasa;
- ćwiczenia praktyczne;
- burza mózgów.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;

- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji).

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie kubełkowe w języku Java". Uczniowie mają zapoznać się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wyświetla i odczytuje temat lekcji oraz cele zajęć. Prosi uczniów o sformułowanie kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu i programowania, np.
   – na czym polega algorytm sortowania kubełkowego?
   – jak nazywamy strukturę danych, która w sposób zorganizowany przechowuje pewien zbiór danych?
   Chętni uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z tekstem.** Metodą burzy mózgów uczniowie referują najważniejsze informacje, jakie znaleźli w sekcji „Przeczytaj", przygotowując się do lekcji. W razie potrzeby nauczyciel wyjaśnia niezrozumiałe kwestie.
   W kolejnym kroku uczniowie na swoich komputerach odtwarzają omówione w tej sekcji algorytmy sortowania liczb całkowitych oraz rzeczywistych.
2. **Praca z multimedium.** Uczniowie zapoznają się z treścią polecenia 1 z sekcji „Film samouczek". W zespołach dwuosobowych opracowują rozwiązania, a następnie porównują je z zaprezentowanym w filmie.
3. **Ćwiczenie umiejętności.** Uczniowie indywidualnie wykonują ćwiczenia nr 1 i 2 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Wybrany uczeń podsumowuje zajęcia z programowania w Javie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się”.
2. Uczniowie realizują polecenie 3 z sekcji „Film samouczek”.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści zawarte w sekcji „Przeczytaj” do przygotowania się do lekcji powtórkowej.