Sortowanie przez wybieranie w języku Java

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

Źródło: Brienne Hong, domena publiczna.

Potrafimy już scharakteryzować algorytm sortowania przez wybieranie. Za jego pomocą możemy sortować np. karty, ale też oceny z kartkówek, czy wyniki uczestników zawodów sportowych. W tym e-materiale poznamy implementację tego algorytmu w języku Java.

Implementacje algorytmu sortowania przez wybieranie w innych językach programowania zostały omówione w e-materiałach:

- Sortowanie przez wybieranie w języku C++,
- Sortowanie przez wybieranie w języku Python.

Więcej zadań? Przejdź do: Sortowanie przez wybieranie – zadania maturalne.

Twoje cele

- Prześledzisz działanie algorytmu sortowania przez wybieranie.
- Przeanalizujesz algorytm sortowania przez wybieranie pod kątem złożoności czasowej.
- Zaimplementujesz w języku Java algorytm sortowania przez wybieranie.

Przeczytaj

Problem 1

Pracujesz w banku. Twoim przełożonym zależy na usprawnieniu procesu przeszukiwania bazy klientów. Szef działu technicznego zasugerował, by zastosować algorytm wyszukiwania binarnego (omówiony w e-materiale Wstęp do algorytmów sortowania). Zanim będzie można to zrobić, trzeba posortować identyfikatory klientów.

- Napisz program sortujący niemalejąco n-elementową tablicę, zawierającą identyfikatory klientów. Każdy z klientów ma przyporządkowany do siebie numer identyfikatora.

- Swój program przetestuj dla tablicy `zbior = [5, 7, 3, 4, 1, 6, 8, 10, 24, 14].`

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `tab`

- `zbior` – n-elementowa tablica liczb naturalnych

Wynik:

- `zbior` – posortowana niemalejąco n-elementowa tablica liczb naturalnych

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

```
1
```

Polecenie 1

Porównaj swoje rozwiązanie z przedstawionym w prezentacji.

1

Przykładowe liczby do posortowania znajdują się w tablicy `zbior`. Po napisaniu programu możesz przetestować program dla dowolnych danych.

2

```
1 int zbior[] = {5, 7, 3, 4,
  1, 6, 8, 10, 24, 14};
```

Stwórzmy pętlę `for`, która będzie przeglądać elementy tablicy `zbior`. W pierwszym wykonaniu pętli przejdziemy liniowo po całej tablicy i znajdziemy najmniejszy element. Następnie zamienimy go z elementem znajdującym się na początku tablicy. W ten sposób mamy zapewnione, że pierwszy element tablicy jest już posortowany. W drugim wykonaniu postąpimy podobnie – tym razem w nieposortowanym jeszcze zbiorze znajdziemy najmniejszy element i ustawimy go na drugim miejscu w tablicy. W każdym kolejnym wykonaniu pętli liczba posortowanych elementów na początku tablicy zwiększy się o 1. Pamiętamy, że ostatni z nich może zostać pominięty, ponieważ w wyniku działania algorytmu zostanie zastąpiony przez największy element z tablicy, co oznacza, że będzie prawidłowo posortowany.

3

```
1 int zbior[] = {5, 7, 3, 4,
  1, 6, 8, 10, 24, 14};
2
3 for (int i = 0; i <
  zbior.length - 1; i++) {
4   // Sortowanie tablicy
5 }
```

Naszym celem jest posortowanie tablicy niemalejąco. W każdym cyklu pętli musimy więc znaleźć najmniejszy element w nieposortowanej części zbioru. Zainicjujmy zmienną, która będzie przechowywać indeks tego elementu.

4

```
1 int zbior[] = {5, 7, 3, 4,
  1, 6, 8, 10, 24, 14};
2
```

```
3 for (int i = 0; i < zbior.length - 1; i++) {
4   int najmniejszyIndeks = i;
5   // Znajdź najmniejszy indeks
6 }
```

Wstępnie zakładamy, że najmniejszy element znajduje się pod indeksem `i`. Zmiennej `najmniejszyIndeks` przypisujemy więc wartość zmiennej sterującej `i`. Żeby sprawdzić, czy element ten naprawdę jest najmniejszy, musimy porównać go z pozostałymi elementami. W tym celu konstruujemy wewnętrzną pętlę `for`. Rozpocznie ona przeglądanie elementów od indeksu `i + 1` (chcemy bowiem analizować wartości, które nie zostały jeszcze posortowane).

5

```
1 int zbior[] = {5, 7, 3, 4, 1, 6, 8, 10, 24, 14};
2
3 for (int i = 0; i < zbior.length - 1; i++) {
4   int najmniejszyIndeks = i;
5   for (int j = i + 1; j < zbior.length; j++) {
6     // Znajdź najmniejszy indeks
7   }
8 }
```

Sprawdzamy, czy aktualny element o indeksie `j` jest mniejszy niż ten, który dotychczas uważaliśmy za najmniejszy. Jeżeli okaże się, że tak, to zmodyfikujemy wartość `najmniejszyIndeks`.

6

```
1  int zbior[] = {5, 7, 3, 4, 1, 6, 8, 10, 24, 14};
2
3  for (int i = 0; i < zbior.length - 1; i++) {
4    int najmniejszyIndeks = i;
5    for (int j = i + 1; j < zbior.length; j++) {
6      if (zbior[j] < zbior[najmniejszyIndeks]) {
7        najmniejszyIndeks = j;
8      }
9    }
10 }
```

Po znalezieniu najmniejszego elementu w części nieposortowanej zamieniamy go miejscami z liczbą wskazywaną przez indeks `i`. Będzie nam potrzebna zmienna pomocnicza. Nazwijmy ją `zmiennaTymczasowa`.

7

```
1  int zbior[] = {5, 7, 3, 4, 1, 6, 8, 10, 24, 14};
2
3  for (int i = 0; i < zbior.length - 1; i++) {
4    int najmniejszyIndeks = i;
5    for (int j = i + 1; j < zbior.length; j++) {
6      if (zbior[j] < zbior[najmniejszyIndeks]) {
7        najmniejszyIndeks = j;
8      }
```

```
9    }
10
11   int zmiennaTymczasowa = zbior[i];
12 }
```

Do zmiennej `zmiennaTymczasowa` kopiujemy element o indeksie `i`. Pod indeksem `i` w tablicy możemy umieścić teraz nową wartość oraz dokończyć zamianę, wstawiając utworzoną kopię w odpowiednie miejsce w tablicy.

8

```
1  int zbior[] = {5, 7, 3, 4, 1, 6, 8, 10, 24, 14};
2
3  for (int i = 0; i < zbior.length - 1; i++) {
4    int najmniejszyIndeks = i;
5    for (int j = i + 1; j < zbior.length; j++) {
6      if (zbior[j] < zbior[najmniejszyIndeks]) {
7        najmniejszyIndeks = j;
8      }
9    }
10
11   int zmiennaTymczasowa = zbior[i];
12   zbior[i] = zbior[najmniejszyIndeks];
13   zbior[najmniejszyIndeks] = zmiennaTymczasowa;
14 }
```

Zawartość posortowanej tablicy wyświetlimy na ekranie.

9

```
int zbior[] = {5, 7, 3, 4, 1, 6, 8, 10, 24, 14};

for (int i = 0; i < zbior.length - 1; i++) {
  int najmniejszyIndeks = i;
  for (int j = i + 1; j < zbior.length; j++) {
    if (zbior[j] < zbior[najmniejszyIndeks]) {
      najmniejszyIndeks = j;
    }
  }

  int zmiennaTymczasowa = zbior[i];
  zbior[i] = zbior[najmniejszyIndeks];
  zbior[najmniejszyIndeks] = zmiennaTymczasowa;
}

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

Na następnej karcie przedstawiamy cały kod programu.

10

```
public class Main {
  public static void main(String []args) {
    int zbior[] = {5, 7, 3, 4, 1, 6, 8, 10, 24,
```

```
14};

    for (int i = 0; i < zbior.length - 1; i++) {
        int najmniejszyIndeks = i;
        for (int j = i + 1; j < zbior.length; j++) {
            if (zbior[j] < zbior[najmniejszyIndeks]) {
                najmniejszyIndeks = j;
            }
        }

        int zmiennaTymczasowa = zbior[i];
        zbior[i] = zbior[najmniejszyIndeks];
        zbior[najmniejszyIndeks] = zmiennaTymczasowa;
    }

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

Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

> **Polecenie 2**
>
> Napisz program sortujący niemalejąco dane zawarte w pliku `identyfikatory.txt`.
>
> Plik `identyfikatory.txt` zawiera 100 liczb naturalnych należących do przedziału $< 424,\ 9882 >$. Każda liczba zapisana jest w osobnym wierszu. Zadbaj o prawidłowe wczytanie danych z pliku. Rozwiązanie zapisz do pliku `identyfikatory_posortowane.txt`.

Przykładowe dane

Plik `identyfikatory.txt`

Plik o rozmiarze 592.00 B w języku polskim

Rozwiązanie

Plik `identyfikatory_posortowane.txt`

Plik o rozmiarze 592.00 B w języku polskim

Złożoność czasowa

Złożoność czasowa algorytmu wynosi $O(n^2)$ i rośnie wykładniczo wraz ze wzrostem elementów w nieposortowanym zbiorze. Przeciętna złożoność czasowa jest taka sama, jak pesymistyczna, ponieważ algorytm sortowania przez wybieranie dla danej długości ciągu zawsze wykona taką samą liczbę porównań. Złożoność czasowa jest również taka sama dla ciągu wejściowego, który byłby już posortowany.

Stabilność

Algorytm sortowania przez wybieranie nie jest algorytmem stabilnym. Oznacza to, że nie utrzymuje kolejności występowania dla elementów o tym samym kluczu. W przypadku sortowania przez wybieranie, sortowanie odbywa się w miejscu.

Załóżmy, że mamy do posortowania nierosnąco tablicę: `[4,4,3,2,5,1]`.

Znajdują się w niej dwie liczby `4`. Żeby zobrazować, że algorytm nie jest stabilny, oznaczmy je odpowiednio `4a` i `4b`: `[4a, 4b, 3, 2, 5, 1]`.

W pierwszym wykonaniu głównej pętli algorytmu zamienimy wartość `5` z wartością `4a`: `[5, 4b, 3, 2, 4a, 1]`.

> **Ważne!**

> Zwróć uwagę na to, że przy sortowaniu **nierosnącym** szukamy **maksimum**, nie minimum (jak wcześniej).

W drugiej iteracji nic się nie zmieni – wartość `4b` pozostanie na swoim miejscu, ponieważ w nieposortowanej jeszcze części tablicy nie ma wartości większej niż `4`.

W kolejnej iteracji zamienimy wartość `3` z wartością `4a`: `[5, 4b, 4a, 2, 3, 1]`.

Po posortowaniu tablica będzie wyglądała następująco: `[5, 4b, 4a, 3, 2, 1]`. Widzimy, że w nieposortowanej tablicy `4a` znajdowało się przed `4b`. W posortowanej tablicy natomiast mamy sytuację odwrotną: `4b` jest przed `4a`. Na przykładzie pokazaliśmy, że algorytm sortowania przez wybieranie nie jest algorytmem stabilnym.

Słownik

> **algorytmy niestabilne**
>
> algorytmy, w których elementy o równej wartości klucza nie występują po posortowaniu w tej samej kolejności, jaką miały w zbiorze nieposortowanym
> **algorytmy stabilne**
>
> w przypadku algorytmów stabilnych w uporządkowanej tablicy nie zmienia się kolejność elementów o tych samych wartościach klucza, np. gdy w tablicy znajdą się dwie liczby o wartości 6, ta mająca niższy indeks przed posortowaniem tablicy, po zakończeniu działania algorytmu wciąż będzie miała mniejszy indeks
> **wyszukiwanie binarne**
>
> metoda odnajdowania elementów w uporządkowanych zbiorach; polega ona na wielokrotnym dzieleniu przeszukiwanego zbioru na dwie części i na porównywaniu wartości szukanej z elementem znajdującym się w środku dzielonego obszaru
> **zagnieżdżona pętla**
>
> pętla działająca wewnątrz innej pętli

Film samouczek

Polecenie 1

Napisz w języku Java program wykorzystujący algorytm sortowania przez wybieranie. Przetestuj jego działanie dla następującej tablicy liczb naturalnych: {5, 7, 3, 4, 1, 6, 8, 10, 24, 14}. Tablica powinna zostać posortowana niemalejąco.

Specyfikacja problemu:

Dane:

- `dane` – nieposortowana tablica liczb naturalnych

Wynik:

- `dane` – posortowana niemalejąco tablica liczb naturalnych

Twoje zadania

1. Napisz program sortujący daną tablicę przez wybieranie. Program powinien sortować tablicę niemalejąco.

Polecenie 2

Zapoznaj się z filmem przedstawiającym realizację algorytmu sortującego przez wybieranie tablicę liczb naturalnych niemalejąco w języku Java.

Film dostępny pod adresem /preview/resource/RQUXbQwsYUAIn

Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Film nawiązujący do sortowania tablicy przez wybór.

Plik o rozmiarze 980.00 B w języku polskim

Polecenie 3

Zmodyfikuj program zapisany w **Poleceniu 1** tak, by sortował tablicę nierosnąco.

Specyfikacja problemu:

Dane:

- `dane` – nieposortowana tablica liczb naturalnych

Wynik:

- `dane` – posortowana nierosnąco tablica liczb naturalnych

Twoje zadania

1. Napisz program sortujący daną tablicę przez wybieranie. Program powinien sortować tablicę nierosnąco.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program wypisujący na standardowe wyjście indeks najmniejszego elementu we wskazanym zakresie podanej tablicy, który zaczyna się od elementu o indeksie `poczatek`, a kończy wraz z końcem tablicy. Działanie programu przetestuj dla tablicy `dane = {423, 654, 423, 659, 345, 432, 765, 534, 469, 421, 6457, 856, 543, 645, 523, 576, 7645}` oraz wartości zmiennej `poczatek` równej 2.

Indeks sprawdzamy, biorąc pod uwagę całą tablicę, nie tylko jej fragment.

Specyfikacja problemu:

Dane:

- `dane` – tablica liczb naturalnych do przeszukania
- `poczatek` – liczba naturalna; indeks elementu tablicy `dane`, od którego należy zacząć przeszukiwanie

Wynik:

- `indeksMin` – liczba naturalna; indeks najmniejszego elementu we wskazanym zakresie tablicy `dane`, który zaczyna się od elementu o indeksie `poczatek`, a kończy wraz z końcem tablicy

Przykład:

Dla tablicy
`dane = {423, 654, 423, 659, 345, 432, 765, 534, 469, 421, 6457, 856, 543, 645, 523, 576, 7645}`
oraz zmiennej `poczatek = 2` przeszukiwany zakres składa się z następujących elementów:
`423, 659, 345, 432, 765, 534, 469, 421, 6457, 856, 543, 645, 523, 576, 7645`, a najmniejszy element (liczba 345) znajduje się na miejscu o indeksie 4.

Twoje zadania

1. Wypisz indeks najmniejszego elementu ze wskazanego zakresu tablicy `dane`.

Ćwiczenie 2

Napisz program, który posortuje niemalejąco podane tablice `dane1`, `dane2`, `dane3` z użyciem algorytmu sortowania przez wybieranie. Dla każdego sortowania podaj liczbę wykonanych porównań oraz przestawień. Dla każdej wypisz, na standardowe wyjście, liczbę porównań, następnie po spacji liczbę przestawień, a w kolejnej linijce elementy posortowanego zbioru, oddzielone spacją. Czy liczba porównań i przestawień zmienia się wraz ze zmianą kolejności elementów w tablicy?

Specyfikacja problemu:

Dane:

- `dane1` – nieposortowana tablica liczb naturalnych; jedna z tablic do posortowania
- `dane2` – nieposortowana tablica liczb naturalnych; jedna z tablic do posortowania
- `dane3` – nieposortowana tablica liczb naturalnych; jedna z tablic do posortowania

Wynik:

Dla każdej tablicy `dane1`, `dane2`, `dane3`: w jednej linijce, oddzielone znakiem odstępu:

- `liczbaPorownan` – liczba naturalna oznaczająca liczbę wykonanych porównań podczas sortowania
- `liczbaPrzestawien` – liczba naturalna oznaczająca liczbę wykonanych przestawień podczas sortowania

W następnej linijce, oddzielone pojedynczym znakiem odstępu, znajdują się posortowane niemalejąco elementy tablicy

Przykładowe wyjście:

```
91 13
111 257 325 342 364 432 435 541 543 654 746 764 765 845
91 13
111 257 325 342 364 432 435 541 543 654 746 764 765 845
91 13
111 257 325 342 364 432 435 541 543 654 746 764 765 845
```

Twoje zadania

1. Program sortuje podane tablice, zlicza liczbę porównań i przestawień wykonanych w trakcie sortowania oraz drukuje te informacje, na standardowe wyjście, wraz z posortowanymi danymi.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie przez wybieranie w języku Java

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy

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 podstawowy. Uczeń:

1) planuje kolejne kroki rozwiązywania problemu, z uwzględnieniem podstawowych etapów myślenia komputacyjnego (określenie problemu, definicja modeli i pojęć, znalezienie rozwiązania, zaprogramowanie i testowanie rozwiązania).

4) porównuje działanie różnych algorytmów dla wybranego problemu, analizuje algorytmy na podstawie ich gotowych implementacji;

5) sprawdza poprawność działania algorytmów dla przykładowych danych.

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

Zakres podstawowy. Uczeń:

1) projektuje i programuje rozwiązania problemów z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje

warunkowe, instrukcje iteracyjne, funkcje z parametrami i bez parametrów, testuje poprawność programów dla różnych danych; w szczególności programuje algorytmy z punktu I.2);

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):

- Prześledzisz działanie algorytmu sortowania przez wybieranie.
- Przeanalizujesz algorytm sortowania przez wybieranie pod kątem złożoności czasowej.
- Zaimplementujesz w języku Java algorytm sortowania przez wybieranie.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

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. Chętne lub wybrane osoby przygotowują krótkie wystąpienie na temat złożoności czasowej i stabilności omawianego algorytmu.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie przez wybieranie w języku Java". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Następnie wspólnie z uczniami ustala kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie zapoznają się z treściami w sekcji „Przeczytaj", jeśli nauczyciel – na podstawie raportu na platformie – uważa, że przygotowanie uczniów jest wystarczające, może pominąć tę czynność.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie w parach piszą w języku Java program wykorzystujący algorytm sortowania przez wybieranie. Nauczyciel sprawdza poprawność wykonania zadania.
3. Uczniowie zapoznają się z filmem samouczkiem. Na forum klasy omawiają przedstawione w nim informacje.
4. Wskazane osoby przedstawiają prezentacje przygotowane przed lekcją.
5. Chętne lub wybrane osoby zadają pytania. W razie potrzeby nauczyciel uzupełnia prezentacje.
6. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że będą rozwiązywać ćwiczenie nr 1 z sekcji „Sprawdź się". Każdy z uczniów robi to samodzielnie. Po ustalonym czasie następuje porównanie napisanych kodów podczas wspólnego omówienia rozwiązań.
7. Uczniowie wykonują ćwiczenie 2 z sekcji „Sprawdź się". Następnie nauczyciel na przykładzie ćwiczenia przedstawia, jak liczebność elementów w nieposortowanym zbiorze wpływa na złożoność obliczeniową algorytmu.

Faza podsumowująca:

1. Nauczyciel zadane pytania podsumowujące, np.
    - czym jest sortowanie?
    - na czym polega algorytm sortowania przez wybieranie?
    - jaką nazwę nosi algorytm sortowania, w którym elementy o tej samej wartości klucza nie zmieniają kolejności względem siebie?
2. 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.

3. Wybrany uczeń podsumowuje zajęcia z programowania w Javie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie realizują program opisany w sekcji „Przeczytaj” dla danych z pliku `identyfikatory.txt`
2. Uczniowie wykonują 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:

- Treści w sekcji „Przeczytaj” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.