Sortowanie pozycyjne słów w języku Java

Źródło: Markus Spiske, domena publiczna.

Z sortowaniem słów mamy do czynienia np. w słowniku, encyklopedii czy dzienniku szkolnym. W e-materiale Sortowanie pozycyjne słów dowiedzieliśmy się, jak do porządkowania słów użyć algorytmu sortowania pozycyjnego.

Teraz zaimplementujemy ten algorytm w języku Java.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Sortowanie pozycyjne słów w języku C++,
- Sortowanie pozycyjne słów w języku Python.

Więcej zadań? Sięgnij do: Sortowanie pozycyjne słów – zadania maturalne.

> **Twoje cele**
>
> - Zaimplementujesz algorytm sortowania pozycyjnego słów w języku Java.
> - Przeanalizujesz algorytm sortujący słowa w porządku leksykograficznym, zaimplementowany w języku Java.
> - Rozwiążesz zadania dotyczące algorytmu *radix sort*.

Film samouczek

W życiu codziennym często mamy do czynienia ze zbiorami posortowanych słów. Są to wszelkiego rodzaju katalogi czy listy, uporządkowane najczęściej zgodnie z kryterium alfabetycznym. Jak już wiesz, sortowanie pozycyjne słów (*radix sort*) służy do sortowania zbiorów danych według porządku leksykograficznego. Wymaga ono zastosowania algorytmu pomocniczego, który musi być stabilny. W tej sekcji posłużymy się sortowaniem przez zliczanie.

Polecenie 1

Napisz program sortujący niemalejąco tablicę imion, wykorzystując sortowanie pozycyjne słów (*radix sort*). Jako algorytm pomocniczy wykorzystaj sortowanie przez zliczanie (*counting sort*).

Specyfikacja problemu:

Dane:

- *dane* – jednowymiarowa tablica przechowująca łańcuchy znaków, która zawiera imiona do posortowania

Wynik:

- program wypisuje posortowane w kolejności niemalejącej imiona z tablicy `dane`, wykorzystując sortowanie pozycyjne słów; kolejne elementy powinny być oddzielone pojedynczym znakiem spacji

Twoje zadania

1. Posortuj niemalejąco tablicę `dane` algorytmem *radix sort*. Wypisz elementy posortowanej tablicy oddzielone pojedynczym znakiem spacji.

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/R1H5aggiQNEvV

Film nawiązujący do treści materiału

Kod programu zprezentowanego w filmie:

Plik o rozmiarze 1.68 KB w języku polskim

Polecenie 3

Dodaj do swojego programu komentarze tak, żeby był zrozumiały dla osoby, która nie potrafi programować.

Polecenie 4

Zapisz implementację programu z Polecenia 1 wykorzystującą inny stabilny algorytm sortowania.

Przeczytaj

W poprzedniej sekcji zaprezentowano, na czym polega sortowanie pozycyjne oraz jak przebiega jego implementacja w języku Java. W tej sekcji pokażemy, że nie zawsze algorytmem pomocniczym jest sortowanie przez zliczanie. Zaimplementujemy algorytm sortowania pozycyjnego, używając pomocniczo sortowania bąbelkowego. Pamiętajmy jednak, że możemy wykorzystać dowolny stabilny algorytm sortujący, np. sortowanie przez wstawianie lub sortowanie przez scalanie.

Algorytm sortowania pozycyjnego w języku Java

Zacznijmy od stworzenia funkcji, która będzie przechowywała tablicę zawierającą łańcuchy znaków – będą to przykładowe słowa.

```
// funkcja sortowania pozycyjnego
static void sortowaniePozycyjneSlow(String[] slowa) {

}
```

Ponieważ sortowane słowa mogą być różnej długości, zanim przystąpimy do właściwej implementacji algorytmu, znajdziemy najdłuższe słowo. W tym celu zapiszemy długość pierwszego słowa do zmiennej `maxDlugosc`. Następnie w pętli będziemy porównywali długości kolejnych słów ze zmienną `maxDlugosc`. Jeżeli sprawdzane słowo okaże się dłuższe, zmienimy wartość zmiennej `maxDlugosc` na długość obecnie sprawdzanego słowa. Pamiętajmy, że interesuje nas tylko długość najdłuższego słowa, a nie to, które słowo jest najdłuższe – dlatego wystarczy tylko jedna zmienna.

```
// funkcja sortowania pozycyjnego
static void sortowaniePozycyjneSlow(String[] slowa) {
    // znajdujemy najdłuższe słowo w tablicy
    int maxDlugosc = slowa[0].length();
    for (int i = 1; i < slowa.length - 1; i++) {
        if(slowa[i].length() > maxDlugosc) {
            maxDlugosc = slowa[i].length();
        }
    }
}
```

Teraz możemy napisać właściwy algorytm sortowania pozycyjnego. W pętli `for` będziemy wywoływać funkcję sortującą – w tym przypadku będzie to funkcja `sortowanieBabelkowe`. Funkcja ta będzie zawierać pomocniczy stabilny algorytm sortowania. Wykonywanie pętli rozpocznie się od znaku o indeksie `maxDlugosc - 1`, a zakończy na indeksie `0`.

```
// funkcja sortowania pozycyjnego
static void sortowaniePozycyjneSlow(String[] slowa) {
    // znajdujemy najdłuższe słowo w tablicy
    int maxDlugosc = slowa[0].length();
    for (int i = 1; i < slowa.length - 1; i++) {
        if(slowa[i].length() > maxDlugosc) {
            maxDlugosc = slowa[i].length();
        }
    }

    // sortujemy dla każdej kolejnej pozycji
    for(int i = maxDlugosc - 1; i >= 0; i--) {
        sortowanieBabelkowe(slowa, i);
    }
}
```

Przykładowy algorytm pomocniczy – sortowanie bąbelkowe

Po przygotowaniu pierwszej funkcji – wskazującej, po którym znaku będziemy układać elementy w tym wywołaniu, wybieramy stabilny algorytm sortowania i przechodzimy do implementacji algorytmu sortowania wewnętrznego. Funkcja realizująca ten algorytm będzie przyjmować dwa argumenty: jednowymiarową tablicę znaków `slowa` oraz liczbę całkowitą `indeksZnaku`. Pamiętajmy, że algorytm w niej zawarty musi być stabilny.

Algorytmem sortowania, który spełnia nasze wymagania jest między innymi sortowanie bąbelkowe.

Opis tego algorytmu możesz znaleźć w odpowiednim e-materiale. W tej sekcji potrzebujemy wprowadzić jedynie odpowiednią modyfikację, która zapobiegnie wyświetlaniu błędów w sytuacji, gdy sortowane słowa są różnej długości.

```
static void sortowanieBabelkowe(String[] slowa, int indeksZnaku)

}
```

Zapiszmy w zmiennej n długość tablicy *slowa*.

```
static void sortowanieBabelkowe(String[] slowa, int indeksZnaku)
    int n = slowa.length;
}
```

Stwórzmy również dwie pętle. Pierwsza z nich w każdym swoim obiegu umieści na odpowiednim miejscu jeden z elementów. Druga pętla, wewnętrzna, będzie odpowiedzialna za porównywanie elementów i ich zamianę.

```
static void sortowanieBabelkowe(String[] slowa, int indeksZnaku)
    int n = slowa.length;
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++) {

        }
}
```

Teraz wprowadzamy instrukcję warunkową, której nie ma w oryginalnym algorytmie. Sprawdzamy, czy słowa które porównujemy, mają na pozycji indeksZnaku jakikolwiek znak. Jeżeli nie mają (ponieważ np. są za krótkie), to porównywanie ich liter o danym indeksie jest bezcelowe.

```
static void sortowanieBabelkowe(String[] slowa, int indeksZnaku)
    int n = slowa.length;
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++) {
            if (slowa[j].length() > indeksZnaku && slowa[j + 1].l

                }
            }
        }
}
```

Zapisujemy instrukcje, które wykonają się, jeśli zostanie spełniony warunek zapisany w **linii 5**. Zadaniem tych instrukcji jest zamiana miejscami porównywanych elementów.

```
static void sortowanieBabelkowe(String[] slowa, int indeksZnaku)
```

```
    int n = slowa.length;
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++) {
            if (slowa[j].length() > indeksZnaku && slowa[j + 1].l
                if (slowa[j].charAt(indeksZnaku) > slowa[j + 1].c
                    String pom = slowa[j];
                    slowa[j] = slowa[j + 1];
                    slowa[j + 1] = pom;
                }
            }
        }
}
```

Mamy gotowy stabilny algorytm sortowania oraz algorytm sortowania pozycyjnego. Możemy teraz wywołać program dla przykładowych danych:

```
public static void main(String[] args) {
    String[] slowa = {
        "ziemniak",
        "kartofel",
        "pyra",
        "pomidor",
        "melon",
        "marchewka",
        "papryka",
        "arbuz",
        "ananas",
        "cebula",
        "czosnek"
    };

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

Na wyjściu otrzymamy zatem:

```
ananas
arbuz
```

```
cebula
czosnek
kartofel
marchewka
melon
papryka
pomidor
pyra
ziemniak
```

Cały kod programu:

```
import java.util.Arrays;

public class SortowaniePozycyjneSlow {

    static void sortowaniePozycyjneSlow(String[] slowa) {
        int maxDlugosc = slowa[0].length();
        for (int i = 1; i < slowa.length - 1; i++) {
            if(slowa[i].length() > maxDlugosc) {
                maxDlugosc = slowa[i].length();
            }
        }

        for (int i = maxDlugosc - 1; i >= 0; i--) {
            sortowanieBabelkowe(slowa, i);
        }

    }

    static void sortowanieBabelkowe(String[] slowa, int indeksZna
        int n = slowa.length;
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++) {
                if (slowa[j].length() > indeksZnaku && slowa[j +
                    if (slowa[j].charAt(indeksZnaku) > slowa[j +
                        String pom = slowa[j];
                        slowa[j] = slowa[j + 1];
                        slowa[j + 1] = pom;
                    }
                }
```

```
            }

    }

    public static void main(String[] args) {
        String[] slowa = {
                "ziemniak",
                "kartofel",
                "pyra",
                "pomidor",
                "melon",
                "marchewka",
                "papryka",
                "arbuz",
                "ananas",
                "cebula",
                "czosnek"
        };

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

Słownik

> **porządek leksykograficzny**
>
> sposób porządkowania elementów zbioru, analogiczny do kolejności alfabetycznej
> **stabilny algorytm sortowania**
>
> algorytm, który dla dwóch równych elementów w zbiorze danych wejściowych zachowuje ich kolejność

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Uzupełnij brakujące miejsca w kodzie tak, aby otrzymać działający program realizujący algorytm sortowania pozycyjnego słów. Dane powinny zostać posortowane leksykograficznie. Swój program przetestuj dla danych `magda`, `ala`, `adam`, `ewa`. Skorzystaj z wybranego przez siebie stabilnego algorytmu sortowania.

Specyfikacja problemu:

Dane:

- `dane` – jednowymiarowa tablica zawierająca imiona do posortowania

Wynik:

- program wypisuje posortowaną tablicę `dane` w porządku leksykograficznym; kolejne elementy tablicy wypisywane są w nowej linii

Twoje zadania

1. Program sortuje zbiór `dane` w porządku leksykograficznym z użyciem sortowania pozycyjnego.

```
public class Main {

    static void przygotujSlowa(String[] dane, int dlugoscNajdluzszegoSlowa) {
        for (int i = 0; i < dane.length; i++) {
            while (dane[i].length() < dlugoscNajdluzszegoSlowa) {
                dane[i] = dane[i] + "`";
            }
        }
    }

    static void wyczyscNapisy(String[] dane) {
        for (int i = 0; i < dane.length; i++) {
```

```

```

Ćwiczenie 2

Zmodyfikuj podany kod tak, aby sortował pozycyjnie ciągi znaków utworzone z cyfr. Dane powinny być posortowane w porządku odwrotnym do leksykograficznego. W porządku leksykograficznym ciąg „11” powinien znaleźć się przed ciągiem „9” (traktujemy liczby jakby to były słowa). W tym zadaniu jednak stosujemy porządek odwrotny do leksykograficznego, zatem „9” powinno stać przed „11”. W sortowaniu pozycyjnym zastosuj wybrany przez siebie stabilny algorytm sortowania.

Specyfikacja problemu:

Dane:

- `dane` – jednowymiarowa tablica przechowująca łańcuchy znaków będące ciągami cyfr do posortowania

Wynik:

- program wypisuje posortowaną w porządku odwrotnym do leksykograficznego tablicę `dane`; kolejne elementy tablicy wypisywane są w nowej linii

Twoje zadania

1. Program sortuje pozycyjnie, w porządku odwrotnym od leksykograficznego, ciągi znaków utworzone z cyfr od 0 do 9.

```
1  public class Main {
2
3      static void przygotujSlowa(String[] dane, int
   dlugoscNajdluzszegoSlowa) {
4          for (int i = 0; i < dane.length; i++) {
5              while (dane[i].length() <
   dlugoscNajdluzszegoSlowa) {
6                  dane[i] = dane[i] + "/";
7              }
8          }
9      }
10
11     static void wyczyscNapisy(String[] dane) {
```

```
12          for (int i = 0; i < dane.length; i++) {
```

```
1
```

Ćwiczenie 3

Organizatorzy debaty na podstawie zgłoszeń przygotowali listę dziennikarzy, którzy po kolei będą zadawać pytania politykom. Kolejność pytań miała zostać ustalona według porządku alfabetycznego. Zgłoszenia przychodziły jednak w losowej kolejności. Dziennikarka o nazwisku Kornas do ostatniej chwili nie potwierdziła swojej obecności na debacie – miała połączyć się ze studiem telefonicznie. Zespół odpowiedzialny za połączenie oczekuje na informację, która w kolejności powinna być ta osoba. Użyj sortowania pozycyjnego słów, aby wyznaczyć kolejność (indeks) tej dziennikarki w posortowanej liście.

W sortowaniu pozycyjnym skorzystaj z dowolnego stabilnego algorytmu sortowania. Zwróć uwagę, że stosujemy numerację od 0.

Dla ułatwienia nazwiska zapisano małymi literami oraz bez znaków diakrytycznych.

Specyfikacja problemu:

Dane:

- `dane` – jednowymiarowa tablica zawierająca nazwiska dziennikarek i dziennikarzy

Wynik:

- program wypisuje posortowaną w kolejności leksykograficznej tablicę `dane` oraz indeks elementu `kornas` w posortowanej tablicy `dane`

Twoje zadania

1. Kod sortuje tablicę nazwisk i wypisuje na standardowe wyjście jedną liczbę: indeks nazwiska "kornas".

```
1 public class Main {
2
3         // Tutaj uzupełnij kod
4
5     public static void main(String[] args) {
6         String[] dane = {
7             "kaluza",
```

```
            "wasniewski",
            "klasa",
            "fil",
            "leszko",
            "radtke",
            "stepniak",
            "falek",
```

```

```

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie pozycyjne słów w języku Java

Grupa docelowa:

Szkoła ponadpodstawowa, 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:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

i) struktury dynamiczne: stos, kolejka, lista (do realizacji algorytmu: ONP, symulacji problemu Flawiusza, sortowania leksykograficznego),

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 algorytm sortowania pozycyjnego słów w języku Java.
- Przeanalizujesz algorytm sortujący słowa w porządku leksykograficznym, zaimplementowany w języku Java.
- Rozwiążesz zadania dotyczące algorytmu *radix sort*.

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. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał „Sortowanie pozycyjne słów w języku Java". Prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj" dotyczącymi 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.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj". W kolejnym kroku uczniowie analizują przykład z sekcji „Przeczytaj" i powtarzają zaprezentowane rozwiązanie na swoim komputerze.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie zapoznają się z problemem 1 i w parach opracowują rozwiązanie. Następnie porównują je z przedstawionym w filmie.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1 z sekcji „Sprawdź się", a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. Liga zadaniowa – uczniowie pracując w parach, wykonują ćwiczenie nr 2 z sekcji „Sprawdź się", a następnie dzielą się swoimi wynikami przez porównywanie napisanego kodu z inną grupą, która również zakończyła zadanie.

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 nr 3 z sekcji „Sprawdź się".
2. Uczniowie wykonują polecenia nr 3–4 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 „Film samouczek” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.