Metody Programowania (2) - modyfikacje sortowań prostych. Zadanie domowe.

Wojciech Szlosek

March 2020

1 Bubble Sort.

1.1 Przypadek, dla którego sortowanie Bubble Sort przy każdym porównaniu elementów wykonuje ich zamiane:

1.1.1

Przyjmijmy, że nasz kod sortowania babelkowego zaczyna sortowanie od lewej strony (tablicy). Ponadto niech kod obejmuje usprawnienie, że gdy nie ma takiej potrzeby, nie sprawdza dalej tablicy. ¡punkt 2 z wykładu¿

1.1.2 Pseudokod

```
do
  swapped = false
  for i = 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap(leftElement, rightElement)
      swapped = true
while swapped
```

1.1.3 Przykład - odpowiedź

Przypadkiem z pytania bedzie po prostu przypadek pesymistyczny, rozważmy taki przykład:

$$4, 3, 2, 1$$

$$\underline{4}, \underline{3}, 2, 1$$

$$3, \underline{4}, \underline{2}, 1$$

$$3, 2, \underline{4}, \underline{1}$$

$$\underline{3}, \underline{2}, 1, 4$$

$$2, \underline{3}, \underline{1}, 4$$

$$2, 1, \underline{3}, \underline{4}$$

$$\underline{2}, \underline{1}, 3, 4$$

$$1, 2, 3, 4$$

Koniec, wynik: uporzadkowany ciag liczbowy. Zauważmy, że musimy "swapować" cały czas; takim przypadkiem bedzie wiec tablica liczb uporzadkowana nierosnaco (podczas gdy chcemy posortować ja niemalejaco).

1.2 Bubble Sort jako sortowanie stabilne.

1.2.1

Obok prostoty, zaleta sortowania babelkowego jest to, że jest ono stabilne. Oznacza to, że elementy o równej wartości beda wystepowały po posortowaniu, w takiej samej kolejności jaka miały w zbiorze nieposortowanym.

1.2.2 Przykład:

$$\underline{3}, \underline{2_A}, 4, 2_B$$

$$2_A, \underline{3}, \underline{4}, 2_B$$

$$2_A, 3, \underline{4}, \underline{2_B}$$

$$\underline{2_A}, \underline{3}, 2_B, 4$$

$$2_A, \underline{3}, \underline{2_B}, 4$$

$$2_A, 2_B, 3, 4$$

2 Selection Sort

2.1 Selection Sort jako stabilny algorytm sortowania.

2.1.1

W podstawowej wersji kodu, Selection Sort nie jest stabilnym sortowaniem.

2.1.2 Modyfikacje i kod stabilnej wersji Selection Sorta:

```
void stabilnySelectionSort(int a[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        int min = i;

        for (int j = i + 1; j < n; j++) //szukamy minimum
            if (a[min] > a[j])
                min = j;

        //wlozenie elementu min na dobra pozycje
        int key = a[min];
        while (min > i)
        {
            a[min] = a[min - 1];
            min--; //przesuwamy
        }
        a[i] = key;
    }
}
```

3 Insertion Sort.

3.1

Ten rodzaj sortowania możemy porównać do układania kart pokerzysty. Pierwsza karte wstawiamy w odpowiednie miejsce przesuwajac pozostałe, nastepna także wstawiamy miedzy odpowiednie karty i tak układamy zestaw kart.

3.2 Kod

```
void sortowaniePrzezWstawianie(int n, int tab[]){
     int pom, j;
     for(int i=1; i<n; i++){
             //wstawienie elementu w odpowiednie miejsce
             pom = tab[i]; //ten element b dzie wstawiony w odpowiednie miejsce
             j = i-1;

             //przesuwanie element w wi kszych od pom
             while(j>=0 && tab[j]>pom)
             {
                        tab[j+1] = tab[j]; //przesuwanie element w
                        --j;
             }
```

```
                tab[j+1] = pom; //wstawienie pom w odpowiednie miejsce
        }
}
```

3.3 Przykład działania i stabilności:

$$2_A, \underline{1}, 3, 2_B$$

$$1, 2_A, \underline{3}, 2_B$$

$$1, 2_A, 3, \underline{2_B}$$

$$1, 2_A, 2_B, 3$$

Co sie tutaj wydarzyło? W pierwszej linii zauważamy, że $2 > 1$, zatem zamieniamy je miejscami, idziemy dalej, i tak dalej...

Widzimy m.in. w tym przykładzie, że sortowanie przez wstawianie jest sortowaniem stabilnym. Dlaczego? W Insertion Sorcie po prostu wybieramy element i umieszczamy go we właściwym miejscu; zamieniamy elementy tylko wtedy, gdy element jest wiekszy niż klucz, tzn. nie zamieniamy elementu na klucz, gdy spełnia on warunek równości.

4 Kolejka LIFO (stos).

4.1 Wykorzystanie stosu do odwracania słowa

4.1.1

Stos można z powodzeniem wykorzystać np. do odwracania słowa. Jak wiemy, po "wrzuceniu" jakiejś liczby/litery do pustego stosu, idzie ona na samo "dno" - w ten sposób wrzucajac kolejne liczby/litery beda one szły od dołu w góre. W ten sposób na wierzchu do wyjecia bedzie obiekt wrzucony najpóźniej. Zatem, by odwrócić słowo, wystarczy literka po literce (od przodu - lewej strony), dodawać je do LIFO, a na koniec po kolei (od góry stosu) wyjmować. W ten sposób otrzymamy słowo wspak - a o to nam właśnie chodziło.

4.1.2 Implementacja w Javie

Dla uproszczenia i zwiezłości kodu, użyto stosu "wbudowanego" w Jave (nie chodzi w tym zadaniu o "tworzenie koła od nowa".

```
import java.util.Scanner;
import java.util.Stack;
public class Main {

        public static Scanner inScan = new Scanner(System.in);

        public static void main(String[] args) {
```

```
                Stack s = new Stack();
                String slowo = inScan.nextLine();

                for(int i=0;i<slowo.length();i++){
                        s.push(slowo.charAt(i));     //wkladamy literka po literc
                }

                for(int i=0;i<slowo.length();i++){
                        System.out.print(s.pop()); //wyjmujemy "od gory"
                }
        }
}
```

Przykład: in: kotek, out: ketok.