Laboratorium nr 7

Sortowanie

1. Sortowanie bąbelkowe (BbS)
2. Sortowanie przez wstawianie (IS)
3. Sortowanie przez wybieranie (SS)

Materiały

Wyróżniamy następujące metody sortowania:

1. Przez prostą zamianę – bąbelkowe (Bubble Sort) – BbS

Sprawdzamy całą tablicę od dołu do góry (od prawej do lewej strony). Analizowane są zawsze dwa sąsiadujące ze sobą elementy. Jeżeli uporządkowane są one tak, że większy poprzedza mniejszy to zamieniane są one miejscami. Czynność powtarzana jest tak długo, aż podczas sprawdzania całej tablicy, nie zajdzie ani jedna zamiana elementów.

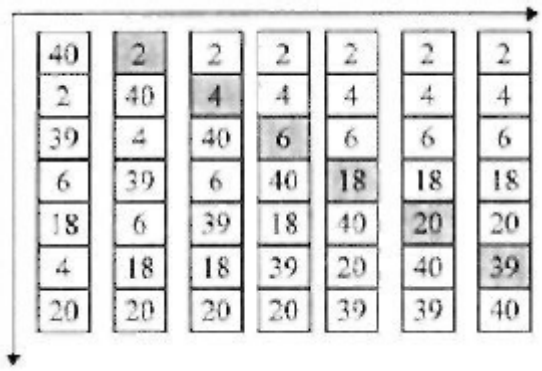
W trakcie pierwszego przebiegu na pierwszą pozycję tablicy (indeks *0)* przesuwa się element „najlżejszy", w trakcie drugiego przebiegu drugi najlżejszy wędruje na drugą pozycję tablicy (indeks 1) i tak dalej, aż do posortowania tablicy. Strefa pracy algorytmu zmniejsza się zatem o 1 w kolejnym przejściu dużej pętli - analizowanie za każdym razem całej tablicy byłoby niepotrzebne.

Algorytm nosi nazwę bąbelkowego przez analogię do pęcherzyków powietrza ulatujących w górę tuby wypełnionej wodą – o ile postawioną pionowo tablicę potraktować jako pojemnik z wodą a liczby jako pęcherzyki powietrza. Najszybciej ulatują do góry „bąbelki" najlżejsze - liczby o najmniejszej wartości (przyjmując sortowanie w kierunku wartości niemalejących).

Idea działania algorytmu, inaczej:

- Algorytm składa się z tzw. fal „bąbli" (powietrza). W każdej fali bierze się po kolei dwa sąsiednie elementy, tzn. 0 i 1, 1 i 2, 2 i 3, ... i zamienia się je miejscami („bąbel"), jeśli element o mniejszym indeksie ma większą wartość.
- Po każdej fali największy element zostaje przesunięty na koniec zbioru. Dlatego każda następna „fala" jest o jeden element krótsza.
- Jeśli w jakiejś „fali" nie wystąpił żaden „bąbel" to znaczy, że wszystkie elementy są uporządkowane od najmniejszego do największego.
- Kolejną „falę" przeprowadza się tylko wtedy kiedy w poprzedniej wystąpił „bąbel" oraz nie wszystkie „fale" zostały wykonane (dla zbioru N-elementowego wykonuje się N-1 fal).
- Przed dojściem do elementu największego „bąblowanie" prowadzi do wstępnego porządkowania elementów mniejszych. Potem największy przesuwa się na koniec zbioru.
- Jeśli zbiór jest uporządkowany, to zostaje wykonana tylko jedna fala „bąbli".

Poniżej znajduje się **przykład** dla nieuporządkowanego ciągu liczb *<40, 2, 39, 6, 18, 4, 20>*.



Inny przykład dla ciągu liczb *<4, 2, 5, 1, 7>*. Kolejne przebiegi algorytmu:

$$\underbrace{[4,2}_{4>2},5,1,7] \rightarrow [2,\underbrace{4,5}_{4<5},1,7] \rightarrow [2,4,\underbrace{5,1}_{5>1},7] \rightarrow [2,4,1,\underbrace{5,7}_{5<7}]$$

$$\underbrace{[2,4}_{2<4},1,5,7] \rightarrow [2,\underbrace{4,1}_{4>1},5,7] \rightarrow [2,1,\underbrace{4,5}_{4<5},7]$$

$$\underbrace{[2,1}_{2>1},4,5,7] \rightarrow [1,\underbrace{2,4}_{2<4},5,7]$$

$$\underbrace{[2,1}_{2>1},4,5,7] \rightarrow [1,\underbrace{2,4}_{2<4},5,7]$$

Tablica wejściowa A w poniższym pseudokodzie zawiera ciąg, który należy posortować. Po zakończeniu procedury tablica A zawiera posortowany ciąg wyjściowy.

PSEUDOKOD

Bubble-Sort (*A*)

1 **for** *i* ←1 **to** length[A]

3 **for** j ← length[A] –1 **to** *i*

4 **if** *A*[*j*] < *A*[*j-1*]

5 **then do** zamień *A*[*j*] ↔ *A*[*j-1*]

Algorytm wykonuje n przejść (fale), przy czym w każdej dokonuje n-1 porównań. **Jego złożoność pesymistyczna to $O(n^2)$.**

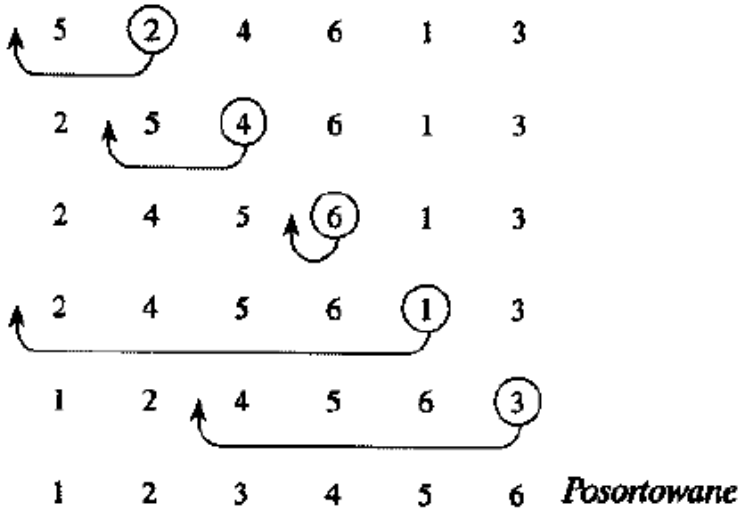
2. Przez proste wstawianie (Insertion Sort) – IS

Algorytm ten działa w taki sposób w jaki ludzie często porządkują talię kart. Zaczynamy od „pustej" lewej ręki, po czym bierzemy ze stołu kolejne karty i wstawiamy je we właściwe miejsca w talii kart, trzymanej w lewej ręce. Aby znaleźć właściwe miejsce dla danej karty, porównujemy ją z kartami, które już mamy w ręce, przesuwając się od strony prawej do lewej.

Opis słowny algorytmu:

1. Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
2. Weź dowolny element ze zbioru nieposortowanego:
    a. Wyciągnięty element porównujemy z kolejnymi elementami zbioru posortowanego póki nie napotkamy elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
    b. Wyciągnięty element wstaw w miejsce gdzie zakończono porównywanie.
3. Jeśli zbiór elementów nieuporządkowanych jest niepusty wróć do punkt 2 (pobieranie elementu nieposortowanego).

Na poniższym rysunku widać **działanie algorytmu dla tablicy** A=<*5, 2, 4, 6, 1, 3*>. Indeks *j* wskazuje „kartę bieżącą", która jest właśnie wstawiana do talii kart w ręce. Elementy tablicy A[1...j-1] reprezentują karty trzymane w ręce, a elementy A[j+1..n] odpowiadają stosowi kart na stole. Indeks *j* przesuwa się od strony lewej do prawej. W każdej iteracji zewnętrznej pętli **for** element A[j] jest pobierany z tablicy (wiersz 2 w pseudokodzie). Następnie, począwszy od pozycji *j-1*, elementy są sukcesywnie przesuwane o jedną pozycję w prawo, aż zostanie znaleziona właściwa pozycja dla A[j] (wiersze 4-7 w pseudokodzie) i wtedy element ten zostaje tam wstawiony (wiersz 8 w pseudokodzie).



Rys. 1.2. Działanie procedury INSERTION-SORT dla tablicy $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Pozycja o indeksie j jest oznaczona kółkiem

Inny przykład:

Zbiór	Opis operacji
7 3 8 5 2	Ostatni element jest zalążkiem listy uporządkowanej.
5 / 7 3 8 _ 2	Ze zbioru wybieramy element leżący tuż przed listą uporządkowaną.
5 / 7 3 8 _ 2	Wybrany element porównujemy z elementem listy.
5 / 7 3 8 2 <<<	Ponieważ element listy jest mniejszy od elementu wybranego, to przesuwamy go na puste miejsce.
7 3 8 2 5	Na liście nie ma już więcej elementów do porównania, więc element wybrany wstawiamy na puste miejsce. Lista uporządkowana zawiera już dwa elementy.
8 / 7 3 _ 2 5	Ze zbioru wybieramy 8
8 / 7 3 _ 2 5	8 porównujemy z 2
8 / 7 3 2 <<< 5	2 jest mniejsze, zatem przesuwamy je na puste miejsce.
8 / 7 3 2 _ 5	8 porównujemy z 5
8 / 7 3 2 5 <<<	5 jest mniejsze, przesuwamy je na puste miejsce
7 3 2 5 8	Lista nie zawiera więcej elementów, zatem 8 wstawiamy na puste miejsce. Na liście uporządkowanej mamy już trzy elementy.

Tablica wejściowa A w poniższym pseudokodzie zawiera ciąg, który należy posortować. Po zakończeniu procedury tablica A zawiera posortowany ciąg wyjściowy.

PSEUDOKOD

Insertion-Sort (*A*)

1 **for** *i* ←1 **to** length[A]

2 **do** key ← A[j]

3 **do** wstaw *A*[*j*] w posortowany ciąg *A*[1..*j-1*]

4 $i \leftarrow$ j –1

5 **while** i>0 i $A[i] >$ key

6 **do** $A[i+1] \leftarrow$ A[i]

7 $i \leftarrow$ i –1

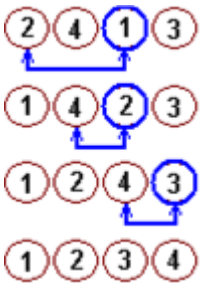
8 A[i+1] $\leftarrow$ key

Jego złożoność pesymistyczna to O(n^2).

3. Przez proste wybieranie/wymianę (Selection Sort) – SS

Na początku szukany jest najmniejszy element. Po jego znalezieniu jest on zamieniany z pierwszym elementem tablicy. Następnie szukany jest ponownie najmniejszy element, ale począwszy od elementu drugiego (pierwszy – najmniejszy jest już wstawiony na odpowiednie miejsce). Po jego znalezieniu jest on zamieniany z drugim elementem. Czynność ta jest powtarzana kolejno na elementach od trzeciego, czwartego, itd., aż do *n*-tego.

Poniżej znajduje się przykład zastosowania dla nieuporządkowanego ciągu liczb <*2, 4, 1, 3*>.




```
procedure prostewybieranie;
    var i, j, k: indeks; x: obiekt;
begin for i := 1 to n-1 do
        begin k := i; x := a[i];
            for j := j+1 to n do
                if a[j].klucz < x.klucz then
                begin k := j; x := a[j]
                end;
            a[k] := a[i]; a[i] := x;
            end
end
```

Literatura:

Thomas H. Cormen, Charles E. Leiserson, Ronald L.Rivest
Wprowadzenie do algorytmów
Wydanie czwarte, Wydawnictwo Naukowo-Techniczne, Warszawa, 2001

N. Wirth
Algorytmy + struktury danych = programy
Wydawnictwo Naukowo-Techniczne, Warszawa, 2004

http://edu.pjwstk.edu.pl/wyklady/asd/scb/index00.html

http://edu.i-lo.tarnow.pl/inf/alg/003_sort/index.php

http://pl.wikipedia.org/wiki/Sortowanie