

Sortowanie

Rozważamy porządek sortowania malejący.

Pierwsze sformułowanie problemu

Dana jest tablica (lista) A z granicami indeksu $1..n$ i składowymi skalarnymi. Należy dokonać permutacji elementów tak, żeby zachodziło:

$A[1] \leq A[2] \leq \dots \leq A[n]$ - tablica (lista) posortowana.

Drugie sformułowanie problemu

Mamy dany ciąg rekordów: $r_1, \dots, r_n$ typu

```
type rek = rekord
  klucz : typ_klucza;
  pole_1: typ_pola_1;
  ...
  pole_m : typ_pola_m
end
```

Zakładamy, że `typ_klucza` jest uporządkowany liniowo (np. tak jak jest dla typów prostych).

Naszym zadaniem jest takie przestawienie elementów ciągu tak, by otrzymany ciąg $r_{p1}, \dots, r_{pn}$ miał własność:
 $r_{p1}.klucz \leq \dots \leq r_{pn}.klucz$

STABILNOŚĆ

Czasami wymaga się dodatkowo zachowania warunku **stabilności**,

tzn. zachowania początkowego ustawienia względem siebie elementów równych (rekordów o takich samych kluczach);

Przykład:

Sortujemy alfabetyczną listę studentów względem wyników egzaminu. Wymagamy, aby w wypadku tej samej oceny nazwiska studentów były podawane w porządku alfabetycznym.

Ocenianie złożoności

Operacja dominująca - porównania elementów w ciągu.

Złożoność pamięciową **S(n)** - ilość dodatkowej pamięci (oprócz n miejsc pamięci dla elementów w ciągu), potrzebnej do wykonania algorytmu.

Dalej zakładamy, że:

elementy sortowane są liczbami całkowitymi i że znajdują się w tablicy A, $A[i] = a_i$, dla $1 \leq i \leq n$.

$$A[0] = -\infty,$$

$$A[n + 1] = +\infty,$$

Przy analizie złożoności przyjmujemy, że danymi wejściowymi są permutacje liczb 1, 2, ... , n oraz że każda taka permutacja jest jednakowo prawdopodobna.

Selectionsort - sortowanie przez selekcję

Idea algorytmu polega na tym, by tablicę A podzielić na części $A[1..i-1]$ złożoną z elementów $A[1], \dots, A[i-1]$ i część $A[i..n]$ złożoną z elementów $A[i], \dots, A[n]$.

Na początku część $A[1..i-1]$ jest pusta.

Dla $i=1$ część $A[1..i-1]$ jest posortowana i w każdym kroku tę część rozszerza się o kolejny jeden element.

Na końcu algorytmu część $A[i..n]$ jest pusta.

Czyli:

Wyznaczamy najmniejszy element w ciągu; zamieniamy go miejscami z pierwszym elementem w ciągu,

wyznaczamy najmniejszy element w $A[2..n]$ i zamieniamy go z drugim elementem w ciągu itd., aż cała tablica zostanie posortowana.

```

procedure selectionsort;
var i, j, min: integer;
begin
  for i := 1 to n - 1 do
    begin {A[1] <= ... <= A[i - 1] <= A[i ... n]}
      min := 1;
      for j := i + 1 to n do
        if A[j] < A[min] then min := j;.
      A[min] < - > A[i];
      {zamiana miejscami A[min] z A[i]}
    end
  end selectionsort;

```

$$T(n) = A(n) = (n-1) + (n-2) + \dots + 1 = \frac{1}{2} n^2 - O(n)$$

$$S(n) = O(1) \quad (\text{algorytm sortuje w miejscu})$$

Algorytm nie jest stabilny

Insertionsort - sortowanie przez wstawianie

Sortowanie przez wstawianie odbywa się w następujący sposób: dla każdego $i = 2, 3, \dots, n$ powtarzamy wstawianie $A[i]$ do już uporządkowanej części listy:

$$A[1] \leq \dots \leq A[i-1].$$

```
procedure insertionsort;  
{ $A[0] = -\infty$ , aby uniknąć testu " $j > 1$ " }  
var i, j, v: integer;  
begin  
  for i := 2 to n do  
    begin { $A[0] \leq A[1] \leq \dots \leq A[i-1]$ }  
      j := i; v := A[i];  
      while A[j - 1] > v do  
        begin  
          { $A[0] \leq \dots \leq A[j-1] \leq A[j+1] \leq \dots \leq A[i]$ ,  
             $j < i \Rightarrow v < A[j+1]$ , A[j] - wolne miejsce}  
          A[j] := A[j - 1]; j := j - 1  
        end;  
        A[j] := v  
      end  
    end  
end insertionsort;
```

$$T(n) = 2 + 3 + \dots + n = \frac{1}{2} n^2 + O(n)$$

$$A(n) = \frac{1}{4} n^2 + O(n)$$

$$S(n) = O(1) \quad (\text{algorytm sortuje w miejscu})$$

Algorytm jest stabilny

Quicksort – sortowanie szybkie

Dziel: Tablica $A[l..r]$ jest dzielona (jej elementy są przestawiane) na dwie niepuste podtablice $A[l..j]$ i $A[j+1..r]$ takie, że każdy element $A[l..j]$ jest niewiększy niż każdy element $A[j+1..r]$. Indeks j jest obliczany przez procedurę dzielącą.

Zwyciężaj: Dwie podtablice $A[l..j]$ i $A[j+1..r]$ są sortowane za pomocą rekurencyjnych wywołań algorytmu quicksort.

Połącz: Ponieważ tablice są posortowane „w miejscu”, to nie potrzeba nic robić, żeby je połączyć: cała tablica $A[l..r]$ jest już posortowana.

```
procedure quicksort(l,r: integer);
var j: integer;
begin
  if l < r then
    begin
      j := partition(l,r);
      if j - 1 > l then quicksort(l,j-1);
      if r > j+1 then quicksort(j+1,r)
    end;
  end quicksort;
```

Jako element v , rozdzielający ciąg $A[l..r]$, wybieramy $v = A[l]$.

Przeglądamy ciąg $A[l+1], \dots, A[r]$ od lewej strony, dopóki nie znajdziemy elementu nie mniejszego niż $A[l]$.

Przeglądamy $A[l+1], \dots, A[r]$, od strony prawej, dopóki nie napotkamy elementu nie większego niż $A[l]$.

Dwa elementy, na których się zatrzymujemy, zamieniamy miejscami.

Powtarzając opisane działania, zapewniamy, że elementy na lewo od wskaźnika wędrującego ze strony lewej na prawą są nie większe niż $A[l]$, a elementy na prawo od wskaźnika wędrującego ze strony prawej na lewą są nie mniejsze niż $A[l]$.

Kiedy oba wskaźniki się spotykają, proces dzielenia jest zakończony - wystarczy element rozdzielający $v = A[l]$ zamienić miejscami z ostatnim elementem lewej części.

```

function partition(l,r):integer;
var v,i,j: integer;
begin
    v := A[l];
    i := l;
    j := r + 1;
while i < j do
    begin
        repeat i := i + 1
        until (A[i] >= v) {nierówność może być <}
        repeat j := j - 1;
        until (A[j] <= v) {nierówność może być <}
        if i < j then A[i] < - > A[j];
    end;
    A[l] := A[j]; A[j] := v;
    partition := j
end;

```

$T(n) \leq n^2 - O(1)$,
 gdzie n liczba elementów tablicy.
 $A(n) \sim \Theta(n \log n)$

Algorytm jest niestabilny

Przykłady:

$$\mathbf{v} = \mathbf{A}[\mathbf{1}] = 7$$
$$\begin{array}{ccccccccccc} 7, & 1, & 5, & 8, & 9, & 6, & 10, & 2, & +\omega \\ & \mathbf{i} & & & & & & \mathbf{j} & & & \end{array}$$

7, 1, 5, 2, 9, 6, 10, 8, +ω
i j

7, 1, 5, 2, 6, 9, 10, 8, +ω
j i j ≤ i

6, 1, 5, 2, 7, 9, 10, 8, + ω
lewy | prawy

5, 3, 2, 6, 4, 1, 3, 7, +∞ **v=A[1] = 5**

i **j**

$$5, 3, 2, 6, 4, 1, 3, 7, +\omega$$

i
j

5, 3, 2, 3, 4, 1, 6, 7, + ω
 j i j <= i

1, 3, 2, 3, 4, 5, 6, 7, +ω
Lewy | prawy