Algorytmy i programowanie

Marcin Żurowski

12 kwiecień 2021

Plan zajęć

Sortowanie

Mając dany ciąg liczb rzeczywistych a_n musimy utworzyć poprzez przestawienie elementów ciągu a_n ciąg a'_n o następującej własności: dla każdego elementu ciągu a'_n jeśli $i < j$ to $a'_i \leq a'_j$ dla sortowania w porządku niemalejącym ($a'_i \geq a'_j$ dla porządku nierosnącego). Ponieważ algorytmy sortowane nierosnąco działają podobnie jak algorytmy sortowane niemalejąco, więc będziemy zajmować się sortowaniem w porządku niemalejącym.

Sortowanie przez wybór

Idea sortowania przez wybór jest następująca. Ponieważ wynikowy ciąg ma mieć porządek niemalejący to najmniejszy element ciągu musi się znaleźć na pierwszej pozycji. Algorytm sortowania przez wybór polega na znalezieniu najmniejszego elementu w tablicy, umieszczeniu go na pierwszej pozycji, a następnie uruchomienia algorytmu dla tablicy począwszy od drugiej pozycji. Do implementacji tego algorytmu może przydać się zmodyfikowana funkcja z zadania 3 poprzednich zajęć.

Przykład

5 2 4 6 1 3 2 6

1 jest szukanym minimum więc zamieniamy ten element z pierwszym elementem.

1 2 4 6 5 3 2 6

Dalej szukamy minimum od drugiej pozycji.

Sortowanie bąbelkowe

Innym podejściem do sortowania jest skorzystanie z faktu, że między dwiema sąsiadującymi ze sobą wartościami w ciągu niemalejącym zachodzi następująca relacja $a_i \leq a_{i+1}$ dla $1 \leq i < n$. Możemy wykorzystując ten fakt przejść przez tablicę zamieniając sąsiednie elementy w przypadku gdy powyższa relacja miedzy sąsiednimi elementami nie zachodzi. Po jednym przejściu takiej tablicy największy element będzie się znajdował pod indeksem n. Wystarczy czynność "przejścia przez tablicę" wykonać $n-1$ razy pomijając za każdym razem k ostatnich elementów, a otrzymamy posortowaną tablicę ($n-1$ ponieważ pierwszy element po $n-1$ przejściach jest elementem najmniejszym).

Przykład

5 2 4 6 1 3 2 6
2 5 4 6 1 3 2 6
2 4 5 6 1 3 2 6
2 4 5 6 1 3 2 6
2 4 5 1 6 3 2 6
2 4 5 1 3 6 2 6
2 4 5 1 3 2 6 6

Na ostatniej pozycji mamy największy element w całej tablicy, po $n-1$ takich operacjach na tablicach mniejszych o 1 w każdym kroku mamy posortowaną tablicę.

2 4 5 1 3 2 6 6

Sortowanie przez wstawianie

Jeszcze innym podejściem do sortowania jest sposób, jakiego używamy, kiedy gramy w karty. Załóżmy, że już $k = 4$ z kart mamy na właściwej pozycji i rozpatrujemy kartę o numerze 1. **Przykład:**

2 4 5 6 1 3 2 6

Gdybyśmy grali w karty po prostu przełożyli byśmy kartę z numerem 1 na pierwsza pozycję w ciągu zielonych posortowanych kart. W algorytmie musimy znaleźć na 1 miejsce w tablicy. Zapamiętujemy więc 1 i liczby w zielonym ciągu przestawiamy dopóki nie znajdziemy miejsca na 1

Przykład

2 4 5 6 6 3 2 6
2 4 5 5 6 3 2 6
2 4 4 5 6 3 2 6
2 2 4 5 6 3 2 6
1 2 4 5 6 3 2 6
1 2 4 5 6 3 2 6

W ten sposób mamy posortowany ciąg o długości $k + 1 = 5$.

Zadania

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

void randTable(int t[], int n);
// losuje n elementów tablicy t z przedziału od 1 do 10000
void copyTable(int source[], int target[] int n);
//kopiuje elementy tablicy source do tablicy target
void insertSort(int t[], int n);
// sortuje tablicę sortowaniem przez wstawianie
void bubleSort(int t[], int n);
// sortuje tablicę sortowaniem bąbelkowym
void selectSort(int t[], int n);
// sortuje tablicę sortowaniem przez wybór
```

Zadania

```
voit printTable(int t[], int n);
// wypisuje tablicę
int main() {
    srand(time(NULL));
    const int N = 100;
    int t[N];
    int c[N];
    randTable(t, N);

    copyTable(t, c, N);
    printTable(c, N);
    insertSort(c, N);
    printTable(c, N);
```

Zadania

```
    copyTable(t, c, N);
    printTable(c, N);
    bubleSort(c, N);
    printTable(c, N);

    copyTable(t, c, N);
    printTable(c, N);
    selectSort(c, N);
    printTable(c, N);

    return 0;
}
```