Metody sortowania: sortowanie bąbelkowe

Najprostszym ze znanych algorytmów sortowania jest tzw. sortowanie bąbelkowe (ang. *bubble sort*). Skąd pochodzi nazwa tego algorytmu? Wyobraźmy sobie kolumnę z wodą, do której wpuszcza się w pobliżu dna powietrze. Pęcherzyki powietrza, mające mniejszą gęstość od wody, unoszone są do góry wskutek działania siły wyporu. Jeśli kolumna z wodą jest zamknięta u góry, tak iż powietrze nie może się z niej wydostać, powietrze stopniowo będzie wypełniać jej górną część. Sortowanie bąbelkowe jest analogią tej sytuacji. W tablicy sortowanej algorytmem „bubble sort" przemieszczamy pojedyncze wartości w stronę jednego z końców tablicy tak, aby stopniowo uszeregować wartości w pobliżu tego końca w żądanym porządku, a następnie stopniowo rozszerzyć uporządkowanie na całą tablicę.

Algorytm „bubble sort" opiera się na wielokrotnym przeglądaniu tablicy w wybranym kierunku. Podczas każdego przeglądu sprawdza się element po elemencie, czy dwie sąsiednie wartości: wartość właśnie rozważana oraz kolejna są ustawione w tablicy we właściwym porządku. Jeśli tak jest, kontynuuje się analizę, począwszy od następnego elementu tablicy. Jeśli jednak rozważane elementy nie są ustawione we właściwej kolejności, należy przed przejściem do kolejnego kroku analizy zamienić je miejscami. Proces przeglądu tablicy prześledzimy na przykładzie sortowania kart. Początkowy układ kart jest następujący:

10	J	K	9	A	Q

Przeglądanie rozpoczniemy od ostatniej pozycji w tablicy oraz przedostatniej. Są to karty: dama oraz as.

10	J	K	9	A	Q

Widzimy, że jeśli chcemy uporządkować karty rosnąco, to te karty nie są ustawione w prawidłowej kolejności, a zatem należy zamienić je miejscami. Po zamianie miejscami kart tablica wygląda następująco:

10	J	K	9	Q	A

Kolejnym krokiem przeglądu jest sprawdzenie relacji zachodzącej pomiędzy przedostatnią kartą (damą) oraz drugą kartą od końca („9"). Karty te ustawione są w prawidłowej kolejności, a zatem nie zamieniamy ich miejscami, lecz od razu przechodzimy do analizy kolejnej pary kart. Rozważamy obecnie kartę na pozycji trzeciej od końca, oraz jej poprzedniczkę.

10	J	K	9	Q	A

Tę parę kart należy zamienić miejscami, po zamianie otrzymamy następujące ułożenie kart:

10	J	9	K	Q	A

W kolejnym kroku przeglądu również należy zamienić karty miejscami:

10	9	J	K	Q	A

Po dokonaniu koniecznej zamiany dostajemy następujące ułożenie kart:

9	10	J	K	Q	A

Proces przeglądu zorganizowany jest w sposób, który gwarantuje umieszczenie na końcu tablicy, do którego dążymy w trakcie przeglądania, dokładnie jednej wartości, która ma się tam znaleźć – w naszym przykładzie była to najmniejsza z kart.

Skoro wiadomo, że po wykonaniu jednego przeglądu dokładnie jeden element zostaje „uniesiony” na odpowiednie miejsce, nasuwa się wniosek, że najpóźniej po wykonaniu *N-1* takich przeglądów, gdzie *N* to ilość elementów tablicy, uzyskamy uporządkowanie całej tablicy. Warto również zauważyć, że jedynie podczas pierwszego przeglądu należy analizować wszystkie sortowane elementy – każdy kolejny przegląd może obejmować o jeden element tablicy mniej (gdyż po każdym z przeglądów jeden element zajmuje odpowiednią pozycję w tablicy).

Przykładowa implementacja algorytmu wygląda następująco (plik *b-sort.cpp*):

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

void wypisz(float tablica[], unsigned ilosc);
void bubble(float tablica[], unsigned ilosc);

int main()
{
float dane[] = {0, 12.5, 7, 13.33, 25.23, 10, 9.99, 0.25, 0.98, 1.01};

   cout<<"Program posortuje tablice z danymi metoda \"bubble sort\""<<endl;
   cout<<"Przed sortowaniem tablica zawiera dane:"<<endl;
   wypisz(dane, 10);
   cout<<endl;

   cout<<endl<<"Sortujemy tablice. Tablica zawiera obecnie dane:"<<endl;
   bubble(dane, 10);
   wypisz(dane, 10);
   cout<<endl;
}

void wypisz(float tablica[], unsigned ilosc)
{
unsigned indeks;

   for(indeks=0; indeks<ilosc; indeks++) cout<< tablica[indeks] <<" ";
}

void bubble(float tablica[], unsigned ilosc)
{
unsigned i, j;

   // i stanowi dolna granice obszaru "przestawianego"
   for(i = 0; i < ilosc-1; i++)

         // j indeksuje przestawiane wartosci (od pozycji ostatniej do "i+1")
         for(j = ilosc-1; j > i; j--)

            // Jesli element rozwazany (j) jest mniejszy od poprzedniego (j-1)
            // dokonaj zamiany elementow miejscami - element j "plynie w gore"
            if( tablica[j] < tablica[j-1] )
            {
            float tymczasowy;

                  // zapamietaj element "poprzedni"
                  tymczasowy = tablica[j-1];

                  // wpisz do elementu "poprzedniego" element "aktualny"
                  tablica[j-1] = tablica[j];

                  // wpisz do elementu "aktualnego" zapamietana wartosc "poprzedniego"
                  tablica[j] = tymczasowy;
            }
}
```

Funkcja *bubble()* jest skonstruowana w oparciu o dwie pętle, zagnieżdżone jedna w drugiej. Zmienna *i* (sterująca zewnętrzną pętlę) wskazuje komórkę tablicy, na której należy zakończyć dany przegląd, zaś zmienna *j* (sterująca pętlą zagnieżdżoną) zawiera indeks aktualnie analizowanego elementu tablicy (drugi z analizowanych elementów posiada indeks *j-1*). Krok pętli wewnętrznej sprowadza się do warunkowego przeprowadzenia zamiany miejscami wartości dwóch komórek tablicy.

Algorytm w podanej dotychczas postaci ma pewną wadę. Spójrzmy jeszcze raz na ostatni rysunek przykładu z kartami. W następnym przeglądzie dokonana zostanie zamiana miejscami króla z damą, co spowoduje że tablica będzie już posortowana. Algorytm mówi jednak, że należy dokonać jeszcze trzech przeglądów tablicy! Aby wyeliminować tę niedogodność, można rejestrować, czy w danym przeglądzie została dokonana choć jedna zamiana pary elementów miejscami, jeśli zamian nie było, to tablica jest już posortowana i można pominąć dalsze przeglądy. Przykładowa implementacja zmodyfikowanej metody sortowania bąbelkowego przedstawia się następująco (plik *ibsort.cpp*):

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

void wypisz(float tablica[], unsigned ilosc);
void i_bubble(float tablica[], unsigned ilosc);

int main()
{
float dane[] = {0, 12.5, 7, 13.33, 25.23, 10, 9.99, 0.25, 0.98, 1.01};

   cout<<"Program posortuje tablice z metoda \"bubble sort\" - udoskonalona"<<endl;
   cout<<"Przed sortowaniem tablica zawiera dane:"<<endl;
   wypisz(dane, 10);
   cout<<endl;

   cout<<endl<<"Sortujemy tablice. Tablica zawiera obecnie dane:"<<endl;
   i_bubble(dane, 10);
   wypisz(dane, 10);
   cout<<endl;
}

void wypisz(float tablica[], unsigned ilosc)
{
unsigned indeks;

   for(indeks=0; indeks<ilosc; indeks++) cout<< tablica[indeks] <<" ";
}

void i_bubble(float tablica[], unsigned ilosc)
{
unsigned i, j;

   // Pierwszego przegladu musimy dokonac
   bool kontynuowac = true;

   for(i = 0; (i < ilosc-1) && kontynuowac; i++)
   {
         //zakladamy optymistycznie, ze kolejnego przegladania NIE BEDZIE
         kontynuowac = false;

         // dokonujemy przegladu tablicy
         for(j = ilosc-1; j > i; j--)
            if( tablica[j] < tablica[j-1] )
            {
            float tymczasowy;
```

```
                    // dokonaj zamiany miejscami elementow
                    tymczasowy = tablica[j-1];
                    tablica[j-1] = tablica[j];
                    tablica[j] = tymczasowy;

                    // Jesli trzeba bylo dokonac choc jednej zamiany, kontynuuj przegladanie
                    kontynuowac = true;
                }
    }
}
```

Powyższy program wyświetla na ekranie następujące informacje:

```
Program posortuje tablice z metoda "bubble sort" - udoskonalona
Przed sortowaniem tablica zawiera dane:
0 12.5 7 13.33 25.23 10 9.99 0.25 0.98 1.01

Sortujemy tablice. Tablica zawiera obecnie dane:
0 0.25 0.98 1.01 7 9.99 10 12.5 13.33 25.23
```

Zadanie

Proszę napisać program, który posortuje malejąco algorytmem „bubble sort“ tablicę 20 liczb całkowitych i wypisze jej elementy na ekran.