

Sortowanie bąbelkowe i przez scalanie

Referat z przedmiotu "Analiza i synteza algorytmów"

Adam Sawicki, gr. II

26 lutego 2006

Sortowanie bąbelkowe

Sortowanie bąbelkowe to najprostsza metoda sortowania i pierwsza, jaka zwykle przychodzi na myśl. Polega na zamienianiu miejscami dwóch sąsiadujących elementów tak długo, aż uzyska się zbiór posortowany.

Twierdzenie 1: Algorytm sortowania bąbelkowego wymaga około $\frac{n^2}{2}$

porównań i $\frac{n^2}{2}$ zamian w przypadku

przeciętnym i najgorszym. Jego asymptotyczna złożoność jest więc kwadratowa - $\Theta(n^2)$.

Niech n będzie wielkością tablicy: $n = \text{Right} - \text{Left} + 1$. Pętla zewnętrzna wykonuje się zawsze $n-1$ razy. Pojedynczy, i -ty przebieg tej pętli wymaga $n-i$ operacji porównania/zamiany, więc otrzymujemy w sumie $(n-1) + (n-2) + \dots + 2 + 1 = \frac{n(n-1)}{2}$ operacji. Od uporządkowania

danych wejściowych zależy jedynie spełnianie lub niespełnianie warunku w pętli wewnętrznej, a co za tym idzie wykonanie lub niewykonanie pojedynczej zamiany, co nie zmienia złożoności algorytmu.

Inna, gorsza wersja algorytmu sortowania bąbelkowego przebiega w pętli wewnętrznej za każdym razem po wszystkich elementach, także już posortowanych. Algorytm można też udoskonalić w taki sposób, by pętla zewnętrzna kończyła się, kiedy cała tablica jest już posortowana.

Sortowanie bąbelkowe ze względu na dużą złożoność nie jest dobrą metodą sortowania i dlatego ma zastosowanie głównie edukacyjne.

```
template <typename Item>
void BubbleSort(Item *Array, int Left, int Right)
{
    int i, j;
    for (i = Left; i < Right; i++)
    {
        for (j = Right; j > i; j--)
        {
            if (Array[j-1] > Array[j])
                std::swap(Array[j-1], Array[j]);
        }
    }
}
```

Sortowanie przez wybieranie

Sortowanie przez wybieranie (selekcję) to jedna z najprostszych metod sortowania. Działa w następujący sposób: najpierw należy znaleźć najmniejszy element tablicy i zamienić go miejscem z pierwszym. Następnie znaleźć drugi w kolejności i zamienić go miejscem z drugim. Kontynuując takie postępowanie uzyskujemy w końcu posortowaną tablicę.

Innymi słowy, lewa część posortowanej tablicy (która początkowo jest pusta i stopniowo się powiększa) zawiera posortowane, najmniejsze elementy tablicy, a z pozostałej (prawej) nieposortowanej części za każdym razem wybieramy najmniejszy element i przenosimy go

```
template <typename Item>
void SelectionSort(Item *Array, int Left, int Right)
{
    int i, j, min;
    for (i = Left; i < Right; i++)
    {
        min = i;
        for (j = i+1; j <= Right; j++)
        {
            if (Array[j] < Array[min])
                min = j;
            std::swap(Array[i], Array[min]);
        }
    }
}
```

na początek tej części uznając go za nowy element części posortowanej.

Twierdzenie 2: Algorytm sortowania przez selekcję wymaga około $\frac{n^2}{2}$ porównań i n zamian. Jego złożoność jest więc kwadratowa - $\Theta(n^2)$.

Dla każdego i z przedziału od 1 do $n-1$ wykonywana jest tylko jedna zamiana, ale $n-i$ porównań, więc otrzymujemy w sumie $n-1$ zamian i $(n-1)+(n-2)+\dots+2+1=\frac{n(n-1)}{2}$ porównań. Wynik taki uzyskuje się niezależnie od rodzaju danych wejściowych. Jedynym członem algorytmu, który zależy od wejścia, jest liczba aktualizacji zmiennej *min*. W najgorszym przypadku wielkość tak może być również kwadratowa, jednak w przypadku przeciętnym jest ona tylko rzędu $O(n \log n)$. Można więc oczekiwać, że czas działania sortowania przez wybieranie będzie niezależny od wejścia.

Czas działania tego algorytmu w bardzo małym stopniu zależy od stopnia uporządkowania zbioru. Mimo prostoty i ewidentnie siłowego podejścia do problemu, sortowanie przez selekcję ma jedną wielką zaletę – jest to najlepszy możliwy algorytm sortowania pod względem ilości przestawień elementów tablicy. Toteż opłaca się go stosować wszędzie tam, gdzie koszt zamiany elementów jest wysoki i dominuje nad kosztem porównań.

Podsumowanie

Obydwa przedstawione w referacie algorytmy sortowania – sortowanie bąbelkowe (ang. *bubble sort*) i sortowanie przez wybieranie (ang. *selection sort*) mają złożoność asymptotyczną klasy $\Theta(n^2)$, co nie jest najlepszą złożonością, jaką można uzyskać. Dlatego w praktycznych zastosowaniach lepiej sprawdzają się zazwyczaj algorytmy o średniej złożoności klasy $O(n \log n)$, jak np. sortowane szybkie (ang. *quick sort*).

Bibliografia

1. Sedgewick Robert, "Algorytmy w C++". Wydawnictwo RM, Warszawa 1999