Laboratorium 5: „Sortowanie tablic”

dr inż. Arkadiusz Chrobot

7 grudnia 2025

1. Wprowadzenie

Sortowanie i wyszukiwanie informacji są według prof. Donalda Knutha najczęściej wykonywanymi operacjami przez komputer. Powstało wiele różniących się poszczególnymi cechami algorytmów dla obu tych operacji. W pierwszej części tej instrukcji znajduje się opis i przykładowa implementacja algorytmu sortowania przez wybór (ang. *selection sort*). Druga część traktuje o algorytmie wyszukiwania binarnego (ang. *binary search*). Trzecia część zawiera treści zadań do zrealizowania.

2. Sortowanie tablic jednowymiarowych

Jedną ze struktur danych, dla których może być przeprowadzana operacja sortowania jest jednowymiarowa tablica. Sortowanie może być klasyfikowane jako wewnętrzne, czyli takie, które nie wymaga bezpośredniego użycia pamięci masowej lub zewnętrzne, które całkowicie lub częściowo wykonywane jest w urządzeniu pamięci masowej. Wyróżniamy także sortowanie stabilne i niestabilne. Pierwsze, w przypadku gdy w tablicy są dwie takie same wartości zachowuje ich początkowe względne położenie, a drugie nie. Aby lepiej wyjaśnić tę cechę sortowania przyjmijmy, że w tablicy są dwie liczby `5`, które oznaczymy kolejno jako `5'` i `5''`. Jeśli sortowanie jest stabilne, to po jego przeprowadzeniu `5'` nadal będzie w tablicy przed `5''`, a jeśli nie, to zamienią się one kolejnością. Sortowanie może być także przeprowadzane w miejscu (łac. *in situ*), tzn. w trakcie jego przeprowadzania nie będzie rosło zapotrzebowanie na miejsce w pamięci operacyjnej niezbędne do jego wykonania.

2.1. Sortowanie przez wybór

Algorytm sortowania przez wybór (ang. *selection sort*) nie należy do najefektywniejszych algorytmów tego typu, ale jest stosunkowo łatwy w implementacji. Przeprowadza on sortowanie niestabilne, wewnętrzne i w miejscu. Konstrukcja tego algorytmu jest dosyć prosta. Na wejście otrzymuje on tablicę nieuporządkowaną[1]. Ponieważ po uporządkowaniu tej tablicy w jej pierwszym elemencie powinna znaleźć się wartość najmniejsza, to algorytm ten szuka wartości minimalnej w całej tablicy, i jeśli znajdzie ją w innym elemencie niż pierwszy, to zamienia miejscami wartości tych elementów. Następnie powtarza to działanie dla drugiego elementu, ale wyłączając z tych czynności pierwszy element. Algorytm kończy swoje działanie po uporządkowaniu dwóch ostatnich elementów tablicy. Listing 1 zawiera funkcję implementującą ten algorytm dla jednowymiarowej tablicy liczb całkowitych z zakresu typu `int`. Kod funkcji `swap()` użytej w `selection_sort()` został podany w materiałach do piątego wykładu.

```
void selection_sort(int array[], const unsigned int number_of_elements)
{
    int i,j;

    for(i=0; i<number_of_elements-1; i++) {
        int min = i;
        for(j=i+1; j<number_of_elements; j++)
            if(array[min]>array[j])
                min = j;
        if(min!=i)
            swap(&array[min],&array[i]);
    }
}
```

Listing 1: Funkcja implementująca sortowanie przez wybór

[1] Nie zawsze tak jest. Sortowanie może być wykonane na tablicy już uporządkowanej, choć w takim wypadku jest zbędne. Tablica wejściowa może być również odwrotnie uporządkowana.

3. Wyszukiwanie w posortowanej tablicy

Wyszukiwanie określonej wartości w tablicy można znacznie przyspieszyć, jeśli ta tablica jest posortowana rosnąco (lub niemalejąco, jeśli wartości w niej się powtarzają) poprzez zastosowanie algorytmu wyszukiwania binarnego (ang. *binary search*). Algorytm ten wyznacza najpierw element środkowy tablicy [2] i porównuje jego wartość z wartością poszukiwaną. Jeśli są one sobie równe, to algorytm zwraca indeks tego elementu i kończy działanie. Jeśli jest ona mniejsza od szukanej, to oznacza to, że wartość poszukiwana, jeśli znajduje się w tablicy, jest w jej obszarze leżącym na prawo od elementu środkowego. Jeśli zaś jest ona większa, to wartości należy szukać w obszarze tablicy położonym na lewo od elementu środkowego. W drugim lub trzecim przypadku algorytm kontynuuje poszukiwania tą samą metodą co dla całej tablicy, ale tym razem stosuje ją tylko dla wyznaczonego jej obszaru. Algorytm kontynuuje zawężanie tych obszarów tak długo, aż znajdzie szukaną wartość lub gdy otrzyma obszar zawierający zero elementów tablicy, co oznacza, że szukanej wartości nie ma w tablicy. W porównaniu do algorytmu przeszukiwania tablicy nieposortowanej w przypadku wartości powtarzającej się w tablicy algorytm wyszukiwania binarnego znajduje *dowolne* jej wystąpienie, niekoniecznie pierwsze. Listing 2 zawiera funkcję, która implementuje algorytm wyszukiwania binarnego dla jednowymiarowej tablicy liczb całkowitych z zakresu typu `int`.

```
int binary_search(int array[], int value, const unsigned int number_of_elements)
{
    int down=0, up=number_of_elements-1;

    while(down<=up) {
        int middle = down+((up-down)/2);
        if(array[middle]==value)
            return middle;
        if(array[middle]<value) {
            down = middle + 1;
            continue;
        }
        if(array[middle]>value)
            up = middle - 1;
    }

    return -1;
}
```

Listing 2: Funkcja implementująca wyszukiwanie binarne

4. Zadania

Wszystkie programy należy napisać z podziałem na funkcje z parametrami!

1. Napisz program, który wypełni tablicę o 100 elementach typu `int` liczbami całkowitymi wylosowanymi z przedziału $[-10; 10]$. Następnie program powinien wypisać zawartość tablicy na ekranie, posortować ją używając funkcji `selection_sort()` i ponownie wypisać jej zawartość. Prześledź działanie tego programu przy pomocy debuggera.

2. Napisz taki sam program jak w zadaniu nr 1, ale dodaj do niego wyszukiwanie liczby w tablicy przy pomocy funkcji `binary_search()`. Wyszukiwana liczbę powinien określać użytkownik. Prześledź działanie programu przy pomocy debuggera. Sprawdź jak zmienia się zawartość zmiennych `down` i `up` w funkcji wyszukiwania.

[2] W przypadku tablic o parzystej liczbie elementów jest to element najbliższy środkowemu.

3. Napisz program, który wypełni tablicę o 30 elementach typu `int` liczbami naturalnymi losowanymi z zakresu od `0` do `10`, a następnie wypisze na ekranie zawartość tę tablicę oraz informację ile razy każda z liczb z podanego zakresu w tej tablicy występuje. *Słowo-wskazówka: histogram.*

4. Napisz program, który posortuje tablicę o 30 elementach wypełnioną liczbami naturalnymi losowanymi z przedziału od `0` do `8`. Zawartość tablicy należy wypisać przed i po posortowaniu.

5. Tablicę liczb można uporządkować dużo szybciej niż robią to algorytmy sortowania ogólnego przeznaczenia. Bazując na rozwiązaniu pierwszego zadania znajdź ten sposób i zademonstruj go dokonując zmian w programie z zadania drugiego.

6. Napisz program, w którym wykorzystasz algorytm przestawiania zaprezentowany na piątym wykładzie do utworzenia tablicy o 30 elementach typu `int`. W tym samym programie postaraj się uporządkować tę tablicę nie używając żadnego z algorytmów sortowania ogólnego przeznaczenia. Zmodyfikuj algorytm z zadania trzeciego, tak aby uwzględniał fakt, że liczby w tablicy się nie powtarzają.

7. Powtórzy zadanie pierwsze dla liczby całkowitych losowanych z przedziału `-5` do `5`.

8. Napisz program, który wypełni tablicę o 21 elementach liczbami całkowitymi losowanymi z zakresu od `-4` do `4`, a następnie posortuje tę tablicę, wyznaczy medianę[3] oraz wartości pierwszego i trzeciego kwartyla[4].

[3] W przypadku tablicy o parzystej liczbie elementów wartość mediany jest średnią arytmetyczną wartości dwóch elementów położonych najbliżej środka tej tablicy.

[4] W przypadku tego zadania przyjmij, że są to wartości odpowiednio położone w $\frac{1}{4}$ i $\frac{3}{4}$ odległości od początku posortowanej tablicy.