Min & Max (sheadovas/algorytmy/min-max/)

Lut 20, 2016 / Algorytmy (sheadovas/category/algorytmy/)

Kontynuacja serii artykułów o algorytmach, tym razem na tapetę trafiają algorytmy służące do znajdowania elementu minimalnego i maksymalnego w zbiorze.

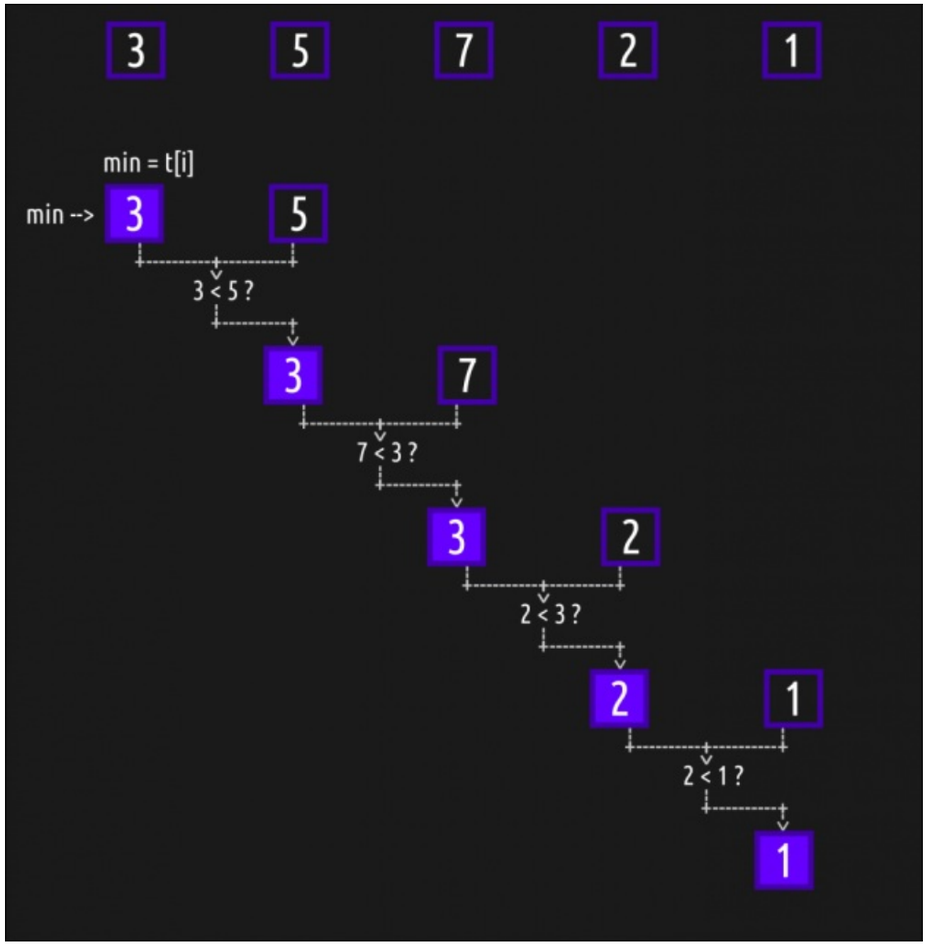
/* Ten artykuł by nie powstał gdyby nie mój znajomy ze studiów, Michale dzięki za dostarczenie „rozrywki” */

Minimum | Maximum

Powszechnie wiadomo, że napianie implementacji wyszukiwania najmniejszego/największego elementu nie jest trudne, ta metoda polega po „przejechaniu” po wszystkich elementach tablicy i porównaniu ich z podejrzanym elementem o bycie minimalnym/maksymalnym.

Lista kroków (szukanie minimum)

1. *i = 1*; *min = t[0]*
2. Jeżeli *t*[i] < *min*, to *min = t[i]*
3. Jeżeli *i < N*, to idź do kroku *(2)*.



(https://i1.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/02/minmax-linear.png)

W analogiczny sposób działa wyszukiwanie elementu maksymalnego, w obu przypadkach mamy do czynienia ze złożonością *O(n)* (ponieważ przechodzimy po wszystkich elementach tablicy) oraz *n-1* porównań (na początku przypisujemy wartość, pętla zaczyna się od i=1). Implementacje tych algorytmów mogą wyglądać następująco:

Przykładowa implementacja funkcji Minimum C++

```
int Minimum(int array[], int size)
{
    int min = array[0];

    for(int i=1; i<size; i++)
    {
        if(min > array[i])
            min = array[i];
    }

    return min;
}
```

Analogicznie wygląda funkcja `int Maximum(int array[], int size)` .

Przykładowa implementacja funkcji Maximum C++

```cpp
int Maximum(int array[], int size)
{
    int max = array[0];

    for(int i=1; i<size; i++)
    {
        if(max < array[i])
            max = array[i];
    }

    return max;
}
```

Minimum & Maximum (v1)

Aby łatwiej zrozumieć kolejną (tą bardziej poprawną) wersję wyszukiwania minimim i maximum prześledźmy jeszcze dodatkowy krok pośredni, który będzie niczym innym jak „wrzuceniem” obu wrześniej napisanych funkcji do jednej pętli for.

Przykładowa implementacja (naiwnej wersji) C++

```cpp
void naive_minmax(int t[], int size, int &min, int &max)
{
    min = max = t[0];
    for(int i=1; i<size; ++i)
    {
        if(t[i] < min)
            min = t[i];
        if(t[i] > max)
            max = t[i];
    }
}
```

Jak widzimy przechodzimy po wszystkich elementach, ale dzięki zastosowaniu pojedynczej pętli uzyskaliśmy 2n-2 porównań, całość można jeszcze znacznie poprawić zauważając ciekawy fakt.

Minimum & Maximum (v2)

Dość łatwo jest zauważyć, że cały proces można nieco poprawić w dość prosty sposób (który swego czasu pokazywałem na blogu, lecz już nie jest dostępny). Mianowicie zamiast sprawdzania elementów tablicy pojedynczo, możemy sprawdzać jednocześnie 2. Wynika to z faktu, że jeżeli A < *B*, to *A* nie może być elementem maksymalnym, może być co najwyżej elementem minimalnym.

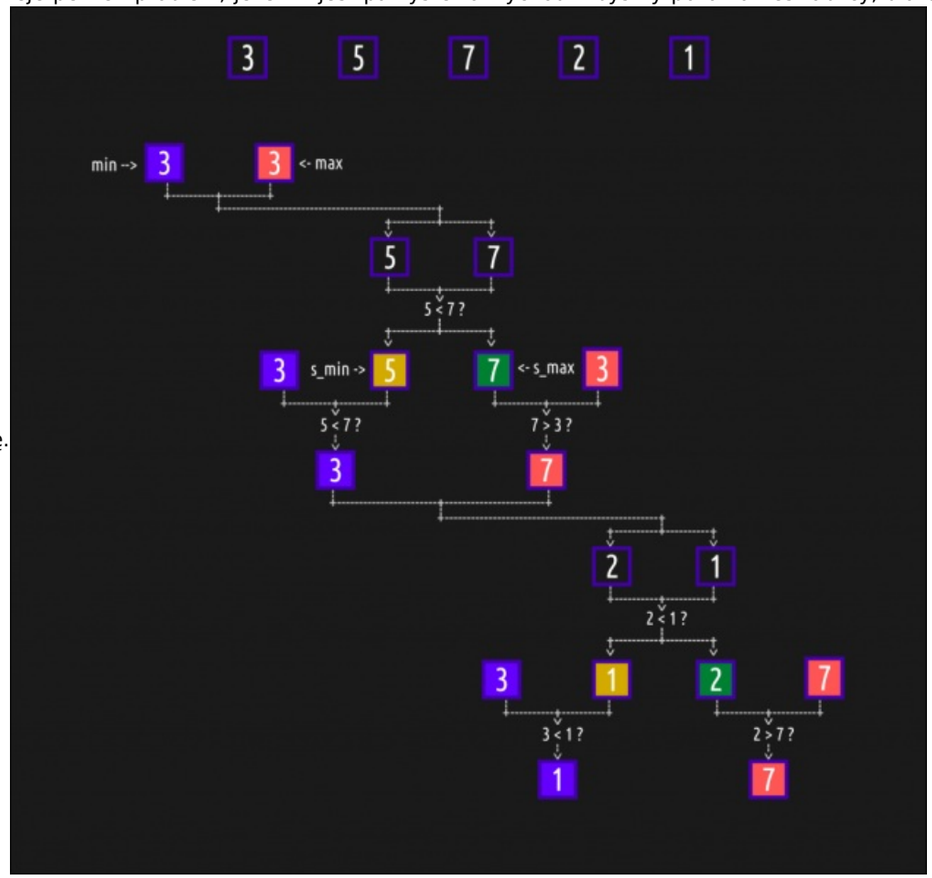
Zmienne

- *t*[*N*] = tablica o rozmiarze *N*;
- *i* – index tablicy;
- *min*, *max* – element minimalny/maksymalny;
- *s_min*, *s_max* – element potencjalnie minimalny/maksymalny.

Lista kroków

1. Ustaw *min* i *max* na pierwszy element tablicy; *i = 1*;
2. Sprawdź, czy *t*[*i*] < *t*[*i+1*]

    1. Jeżeli tak, to *t*[*i*] jest podejrzane o bycie nowym elementem minimalnym, a *t*[*i+1*] jest podejrzany o bycie elementem maksymalnym (*s_min* = *t*[*i*]; *s_max* = *t*[*i+1*]).
    2. Jeżeli nie, to mamy sytuację odwrotną (*s_min* = *t*[*i+1*]; *s_max* = *t*[*i*])
3. Jeżeli:
    1. *s_min* < *min*, to znaleźliśmy nowy element minimalny: *min* = *s_min*.
    2. *s_max* > *max*, to znaleźliśmy nowy element maksymalny: *max* = *s_max*.
4. *i* = *i+2*; jeżeli *i* < *N* to idź do kroku *(2)*.

Oczywiście istnieje pewien problem, jeżeli *N* jest parzyste to wychodzilibyśmy poza zakres tablicy, dlatego należy na początku obiegu pętli sprawdzić czy jesteśmy w stanie odwołać się do t[i+1]. Poniżej przedstawiam przykładową implementację. 

(https://i2.wp.com/www.shead.ayz.pl/wp-content/uploads/2016/02/minmax.png)

```
Przykładowa implementacja                                                    C++
```

```
void minmax(int t[], int size, int &min, int &max)
{
    int i = 1;
    if(!(size & 1))
        i = 0;

    min = max = t[0];
    for(; i<size; i+=2)
    {
        int s_min, s_max; // mozna sobie darowac tworzenie dodatkowych zmiennych,
        if(t[i] < t[i+1]) // tutaj pojawia sie jedynie dla skrocenia zapisu i lepszej czytelnosci
        {
            s_min = t[i];
            s_max = t[i+1];
        }
        else
        {
            s_min = t[i+1];
            s_max = t[i];
        }


        if(s_min < min)
            min = s_min;

        if(s_max > max)
            max = s_max;
    }
}
```

Złożoność powyższego algorytmu to wciąż *O(n)*, ale mamy stosunkowo mniej porównań, bo zeszliśmy do *3n/2*. Szybka (i nieformalna) analiza: wykonujemy 3 porównania, które powtarzamy n/2 razy (przy każdym obiegu pętli poruszamy się o 2 indeksy).

MinMax & Sortowanie przez Wybieranie (Selection Sort)

---

coming soon...

Code ON!