Jednoczesne wyszukiwanie maksimum i minimum

Algorytm

Zadanie jest następujące:

> W n elementowym zbiorze danych znaleźć element maksymalny i minimalny.

Jeśli podejdziemy do tego zadania w sposób tradycyjny, to wyszukanie elementu maksymalnego będzie wymagało wykonania n - 1 porównań (porusz poprzedni rozdział). Podobnie jest dla elementu minimalnego. Zatem złożoność obliczeniowa problemu wyszukania elementu maksymalnego i minimalnego będzie równa:

$$T(n) = 2(n - 1) = 2n - 2$$

Czy można ten wynik poprawić? Okazuje się, że tak. Musimy zastosować zasadę **Dziel i Zwyciężaj** (ang. Divide & Conquer). Polega ona na podziale problemu na mniejsze problemy i wyznaczeniu rozwiązania z rozwiązań problemów mniejszych. W naszym przypadku zbiór rozdzielimy na dwa podzbiory - w jednym będą elementy mniejsze a w drugim elementy większe. W podzbiorze z elementami mniejszymi wyznaczymy element minimalny, a w podzbiorze z elementami większymi wyznaczymy element maksymalny.

Najpierw sprawdzimy, czy n jest liczbą parzystą. Jeśli nie, to na końcu zbioru dublujemy ostatni element. Parzysta liczba elementów jest nam potrzebna do zgrabnego podziału zbioru na dwie połówki. Zdublowanie ostatniego elementu zbioru nie wpłynie na wyznaczenie elementu minimalnego i maksymalnego.

Ze zbioru pobieramy kolejne dwa elementy i porównujemy ze sobą. Element mniejszy jest kandydatem na element minimalny. Element większy jest kandydatem na element maksymalny. Zatem po porównaniu dokonujemy dla mniejszego elementu testu na element minimalny, a dla większego dokonujemy testu na element maksymalny. Po przeglądnięciu wszystkich par elementów zbioru znajdujemy element minimalny i maksymalny.

Policzmy ilość niezbędnych porównań:

Ponieważ ze zbioru pobieramy po dwa elementy, to podziały na elementy mniejsze i większe wymagają $^{n}/_{2}$ porównań. Po wykonaniu każdego takiego porównania musimy sprawdzić, czy wyznaczone elementy są odpowiednio elementem minimalnym i maksymalnym. Czyli otrzymujemy $^{n}/_{2}$ porównań przy teście na element minimalny oraz $^{n}/_{2}$ porównań przy teście na element maksymalny. Zatem złożoność obliczeniowa naszego algorytmu wyrazi się wzorem:

$$T(n) = \frac{3}{2} \quad n < 2n - 2$$

Otrzymany wynik jest dużo korzystniejszy niż wynik poprzedni. Zysk powstaje stąd, iż elementu minimalnego i maksymalnego poszukujemy w podzbiorach o liczebności dwa razy mniejszej niż zbiór wyjściowy. Klasa złożoności obliczeniowej nie ulega zmianie i wynosi $O(n)$.

Specyfikacja problemu

Dane wejściowe

n - liczba elementów w zbiorze; $n \in N$, $n > 0$

d[ ] - zbiór, w którym poszukujemy elementu maksymalnego. Indeksy elementów rozpoczynają się od 1. Jeśli liczba elementów jest nieparzysta, to w zbiorze powinno być miejsce na jeden dodatkowy element.

Dane wyjściowe

w_{min} - wartość wyznaczonego elementu minimalnego;

p_{min} - pozycja elementu minimalnego; $p_{min} \in N$, $1 \leq p_{min} \leq n$

w_{max} - wartość wyznaczonego elementu maksymalnego;

p_{max} - pozycja elementu maksymalnego; $p_{max} \in N$, $1 \leq p_{max} \leq n$

Zmienne pomocnicze

i - $i \in N$, $i \in \{2,3,...,10\}$

Lista kroków

K01: Jeśli n **mod** $2 \neq 0$, to $d[n + 1] \leftarrow d[n]$

K02: $w_{min} \leftarrow d[1]$; $w_{max} \leftarrow d[1]$

K03: $p_{min} \leftarrow 1$; $p_{max} \leftarrow 1$

K04: $i \leftarrow 1$

K05: Dopóki $i \leq n$: wykonuj kroki 6...12

K06: Jeśli $d[i] < d[i + 1]$, to idź do kroku K07. Inaczej idź do kroku K10

K07: Jeśli $d[i] < w_{min}$, to $w_{min} \leftarrow d[i]$; $p_{min} \leftarrow i$

K08: Jeśli $d[i + 1] > w_{max}$, to $w_{max} \leftarrow d[i + 1]$; $p_{max} \leftarrow i + 1$

K09: Idź do kroku K12

K10: Jeśli $d[i + 1] < w_{min}$, to $w_{min} \leftarrow d[i + 1]$; $p_{min} \leftarrow i + 1$

K11: Jeśli $d[i] > w_{max}$, to $w_{max} \leftarrow d[i]$; $p_{max} \leftarrow i$

K12: $i \leftarrow i + 2$ i wróć do kroku K05

K13: Jeśli $p_{min} > n$, to $p_{min} \leftarrow p_{min} - 1$

K14: Jeśli $p_{max} > n$, to $p_{max} \leftarrow p_{max} - 1$

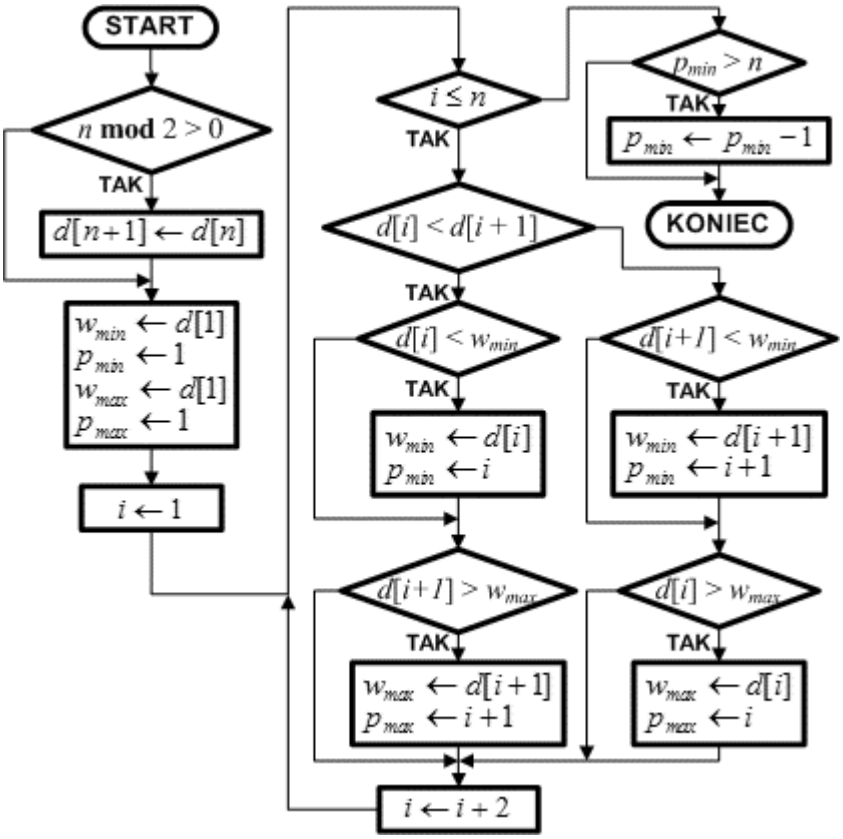
K15: Zakończ algorytm

Schemat blokowy

Pierwszy test w algorytmie sprawdza, czy zbiór *d*[ ] składa się z parzystej liczby elementów (parzysta liczba elementów jest niezbędna do równego podziału na dwie połówki). Jeśli nie, to wymuszamy parzystą liczbę elementów przez zdublowanie ostatniego elementu zbioru.

Następnie inicjujemy zmienne dla elementu minimalnego i maksymalnego. Tymczasowo wpisujemy do nich wartość pierwszego elementu zbioru oraz pozycję pierwszego elementu.

Rozpoczynamy pętlę przeszukującą zbiór. W pętli porównujemy ze sobą kolejne pary elementów zbioru - *i*-ty z (*i*+1)-szym. Porównanie to pozwala nam określić, który z tych elementów jest kandydatem na element minimalny a który na maksymalny. W zależności od wyniku porównania idziemy jedną z dwóch ścieżek.



Ścieżką **TAK** podążamy, gdy element *i*-ty jest mniejszy od elementu (*i*+1)-szego. W takim przypadku sprawdzamy, czy *i*-ty element jest nowym elementem minimalnym, a (*i*+1)-szy jest nowym elementem maksymalnym. Jeśli któryś z tych warunków jest spełniony, to odpowiednio modyfikujemy zmienne dla elementu minimalnego i maksymalnego.. W przypadku ścieżki **NIE** mamy sytuację odwrotną. Element *i*-ty jest testowany na element maksymalny, a element (*i*+1)-szy na element minimalny (zobacz na uwagi końcowe).

Po wykonaniu testów zwiększamy indeks *i* o 2, czyli przechodzimy do kolejnej pary elementów w zbiorze *d*[ ]. Pętla jest wykonywana dotąd, aż wszystkie pary zostaną przetworzone. Na wyjściu w zmiennych w_{min}, p_{min} mamy wartość i pozycję elementu minimalnego, a w w_{max} i p_{max} wartość i pozycję elementu maksymalnego.

Na koniec musimy jeszcze sprawdzić, czy wyznaczona pozycja elementu minimalnego p_{min} nie leży poza zbiorem. Może się tak zdarzyć, gdy najmniejszy element znajduje się na ostatniej pozycji w zbiorze, a zbiór posiada nieparzystą liczbę elementów. W takiej sytuacji dublowaliśmy ostatni element. Zatem porównanie $d[i] < d[i + 1]$ da wynik negatywny, ponieważ oba elementy są równe. Algorytm zaliczy jako w_{min} element na pozycji $i + 1$, a to jest poza ostatnim elementem wejściowego zbioru. Wystarczy jedynie zmniejszyć o 1 wartość pozycji p_{min} i wszystko będzie w porządku. Kończymy algorytm.