Lista 3, zadanie 2 - Tomasz Woszczyński

Treść: Przeanalizuj następujący algorytm oparty na strategii dziel i zwyciężaj jednoczesnego znajdowania maksimum i minimum w zbiorze $S = \{a_1, \ldots, a_n\}$:

function MAXMIN(S: Set)
  if $|S| = 1$ **then**
    return $\{a_1, a_1\}$
  else if $|S| = 2$ **then**
    return $(\max(a_1, a_2), \min(a_1, a_2))$
  else
    Podziel S na dwa równoliczne podzbiory S_1, S_2
    (max1, min1) $\leftarrow$ MaxMin(S_1)
    (max2, min2) $\leftarrow$ MaxMin(S_2)
    return (max(max1, max2), min(min1, min2))
  end if
end function

Uwaga: Operacja **return** $(\max(a_1, a_2), \min(a_1, a_2))$ wykonuje jedno porównanie.

1. Jak pokażemy na jednym z wykładów każdy algorytm dla tego problemu, który na elementach zbioru wykonuje jedynie operacje porównania, musi wykonać co najmniej $\left\lceil \frac{3}{2}n - 2 \right\rceil$ porównania. Dla jakich danych powyższy algorytm wykonuje tyle porównań? Podej wzorem wszystkie takie wartości.

2. Jak bardzo może różnić się liczba porównań wykonywanych przez algorytm od dolnej granicy?

3. Popraw algorytm, aby osiągał on tę granicę dla każdej wartości n.

Rozwiązanie (1): Oznaczmy optymalną ilość porównań przez $Opt(n) = \left\lceil \frac{3}{2}n - 2 \right\rceil$. Rozpatrzmy sposób działania algorytmu przedstawionego w treści zadania: poza przypadkami brzegowymi, zbiór S jest dzielony na dwie równe części (z dokładnością do jednego elementu), a więc jedna część ma $\left\lfloor \frac{n}{2} \right\rfloor$ elementów, a druga $\left\lceil \frac{n}{2} \right\rceil$. Procedura wykonuje później dwa porównania, a więc zbiór rekurencyjny wyraża się przez:

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 2$$

Warunki początkowe możemy odczytać z dwóch pierwszych instrukcji warunkowych, a więc $T(1) = 0$, gdyż zwracana jest para powstała z jednego elementu oraz $T(2) = 1$, ponieważ wykonywane jest tylko jedno porównanie (co wiemy z uwagi).

Dowód (1): Pokażemy zgodność powyższego wzoru rekurencyjnego ze wzorem jawnym $T'(n)$, a więc należy indukcyjnie udowodnić, że $\forall n T(n) = T'(n)$ względem k, gdzie $n = 2^{k+1} + r$.

$$T(n) = \begin{cases} 0, & \text{dla } n = 1 \\ 3 \cdot 2^k + 2r - 2, & \text{dla } 0 \leqslant r \leqslant 2^k \\ 4 \cdot 2^k + r - 2, & \text{dla } 2^k < r < 2^{k+1} \end{cases}$$

Podstawa indukcyjna dla $k = 0$ oraz $r \in \{0, 1\}$ jest oczywista, wynika to bezpośrednio z definicji, wiemy również, że $T'(1) = T(1)$ z definicji. Przejdźmy do kroku indukcyjnego: załóżmy, że nasza teza zachodzi dla wszystkich $k' \leqslant k - 1$, a więc dla $n < 2^{k+1}$. Musimy rozpatrzyć następujące przypadki:

- $0 \leqslant r \leqslant 2^k$: wzór jawny jest wtedy postaci $T'(n) = 3 \cdot 2^k + 2r - 2$. Po rozwinięciu wzoru rekurencyjnego łatwo zauważyć, że będziemy korzystać z drugiego przypadku wzoru jawnego, czyli $\left\lfloor \frac{r}{2} \right\rfloor$ oraz $\left\lceil \frac{r}{2} \right\rceil$ mieszczą się w przedziale $\left[0; 2^{k-1}\right]$.

$$\begin{aligned}
T(2^{k+1} + r) &= T\left(2^k + \left\lfloor \frac{r}{2} \right\rfloor\right) + T\left(2^k + \left\lceil \frac{r}{2} \right\rceil\right) + 2 = \\
&= \left(3 \cdot 2^{k-1} + 2 \cdot \left\lfloor \frac{r}{2} \right\rfloor - 2\right) + \left(3 \cdot 2^{k-1} + 2 \cdot \left\lceil \frac{r}{2} \right\rceil - 2\right) = \\
&= 3 \cdot 2^k + 2\left(\left\lfloor \frac{r}{2} \right\rfloor + \left\lceil \frac{r}{2} \right\rceil\right) - 2 = \\
&= 3 \cdot 2^k + 2r - 2 = \\
&= T'(2^{k+1} + r)
\end{aligned}$$

- $2^k < r < 2^{k+1}$: tutaj musimy rozpatrzyć kolejne wartości, które będziemy podstawiać do wzoru:

$$T(2^{k+1} + r) = T\left(2^k + \left\lfloor \frac{r}{2} \right\rfloor\right) + T\left(2^k + \left\lceil \frac{r}{2} \right\rceil\right) + 2 = *$$

  - dla $r = 2^k + 1$:

$$\begin{aligned}
* &= \left(3 \cdot 2^{k-1} + 2 \cdot 2^{k-1} - 2\right) + \left(4 \cdot 2^{k-1} + 2^{k-1} + 1 - 2\right) + 2 = \\
&= 4 \cdot 2^k + \left(2^k + 1\right) - 2 = Opt(n)
\end{aligned}$$

  - dla $r = 2^{k+1} - 1$:

$$\begin{aligned}
* &= \left(4 \cdot 2^{k-1} + 2^k - 1 - 2\right) + \left(3 \cdot 2^k + 2 \cdot 0 - 2\right) + 2 = \\
&= 4 \cdot 2^k + \left(2^{k+1} - 1\right) - 2 = Opt(n)
\end{aligned}$$

  - dla $r \in \left(2^k + 1; 2^{k+1} - 1\right)$:

$$\begin{aligned}
* &= \left(4 \cdot 2^{k-1} + \left\lfloor \frac{r}{2} \right\rfloor - 2\right) + \left(4 \cdot 2^{k-1} + \left\lceil \frac{r}{2} \right\rceil - 2\right) + 2 = \\
&= 4 \cdot 2^k + r - 2 = Opt(n)
\end{aligned}$$

Teraz należy sprawdzić, dla jakich wartości algorytm ten jest optymalny, a więc kiedy zachodzi $T(n) = Opt(n)$. Po prostym przekształceniu otrzymujemy $Opt(n) = \left\lceil \frac{3}{2}n - 2 \right\rceil = 3 \cdot 2^k + \left\lceil \frac{3}{2}r \right\rceil - 2$. Musimy rozpatrzyć więc dwa przedziały naszego wzoru:

- $0 \leqslant r \leqslant 2^k$: $3 \cdot 2^k + 2r - 2 = 3 \cdot 2^k + \left\lceil \frac{3}{2}r \right\rceil - 2 \Longrightarrow r = \left\lceil \frac{r}{2} \right\rceil$, czyli $r \in \{0, 1\}$;

- $2^k < r < 2^{k+1}$: $4 \cdot 2^k + r - 2 = 3 \cdot 2^k + \left\lceil \frac{3}{2}r \right\rceil - 2 \Longrightarrow 2^k = \left\lceil \frac{r}{2} \right\rceil$. Rozwiązanie to pasuje dla $r = 2^{k+1} - 1$ oraz dla $r = 2^{k+1}$, jednak drugi przypadek wykracza poza dziedzinę.

Algorytm jest optymalny więc dla takich n, gdzie $r \in \{0, 1, 2^{k+1} - 1\}$.

Rozwiązanie (2): W drugim podpunkcie musimy sprawdzić jak bardzo algorytm może odbiegać od optymalnego rozwiązania, zdefiniujmy więc funkcję $R(n) = T(n) - Opt(n)$. Po przeprowadzonym wcześniej rozumowaniu możemy rozbić ją na funkcję złożoną dla dwóch przedziałów r:

$$R(n) = \begin{cases} \left\lfloor \frac{r}{2} \right\rfloor, & \text{dla } 0 \leqslant r \leqslant 2^k \\ 2^k - \left\lceil \frac{r}{2} \right\rceil, & \text{dla } 2^k < r < 2^{k+1} \end{cases}$$

Funkcja $\left\lfloor \frac{r}{2} \right\rfloor$ jest funkcją niemalejącą, a $2^k - \left\lceil \frac{r}{2} \right\rceil$ nierosnącą, skąd możemy wywnioskować, że ich maksymalna różnica powinna znajdować się gdzieś po środku, a więc przy $r = 2^k$. Rozpatrzmy więc wartości różniące się niewiele od takiego r:

- $R(2^{k+1} + 2^k - 1) = 2^{k-1} - 1$
- $R(2^{k+1} + 2^k) = 2^{k-1}$
- $R(2^{k+1} + 2^k + 1) = 2^{k-1} - 1$

Stąd wiemy, że algorytm działa najgorzej dla $n = 2^{k+1} + 2^k = 3 \cdot 2^k$ i różni się wtedy o $2^{k-1} = \frac{n}{6}$ porównań od optymalnego.

Rozwiązanie (3): Poprawienie działania algorytmu:

```
function MaxMin(S: Set)
    if |S| = 1 then
        return {a_1, a_1}
    else if |S| = 2 then
        return (max(a_1, a_2), min(a_1, a_2))
    else if |S| jest potęgą 2 then
        Podziel S na dwa równoliczne podzbiory S_1, S_2
        (max1, min1) ← MaxMin(S_1)
        (max2, min2) ← MaxMin(S_2)
        return (max(max1, max2), min(min1, min2))
    else
        k ← ⌊log_2 |S|⌋
        S_1 ← [a_1 ... a_{2^k}]
        S_1 ← [a_{2^k+1} ... a_{|S|}]
        (max1, min1) ← MaxMin(S_1)
        (max2, min2) ← MaxMin(S_2)
        return (max(max1, max2), min(min1, min2))
    end if
end function
```

Weźmy $P(n)$ określające ilość porównań ulepszonego algorytmu szukania minimum i maksimum:

$$P(n) = \begin{cases} 0, & \text{dla } n = 1 \\ 1, & \text{dla } n = 2 \\ 2 \cdot P(\frac{n}{2}) + 2, & \text{dla } n = 2^k \\ P(2^k) + P(r) + 2 & \text{w p.p., czyli dla } n = 2^k + r \end{cases}$$

Udowodnimy to również indukcyjnie po k, gdzie $n = 2^k + r$. Dla podstawy indukcji, a więc dla $n = 1$ oraz $n = 2$ jest to oczywiste, załóżmy że algorytm działa dla $k - 1$. Niech $r = 0$, wtedy:

$$P(2^k) = 2 \cdot P(\frac{n}{2}) + 2 = 2 \cdot \left\lceil \frac{3}{2} \cdot \frac{n}{2} - 2 \right\rceil + 2 = \left\lceil \frac{3}{2} 2^k - 2 \right\rceil = Opt(2^k)$$

Niech $r \neq 0$, wtedy:

$$P(2^k + r) = P(2^k) + P(r) + 2 = \left\lceil \frac{3}{2} 2^k - 2 \right\rceil + \left\lceil \frac{3}{2} r - 2 \right\rceil + 2 = 3 \cdot 2^{k-1} + \left\lceil \frac{3}{2} r \right\rceil - 2 = Opt(2^k + r)$$

Stąd mamy, że $P(n) = Opt(n)$, a więc algorytm jest w każdym przypadku optymalny, a więc doszliśmy do końca rozwiązania.