Lista 3, zadanie 2

Algorytmy i struktury danych

Rafał Łasocha

3 kwietnia 2014

1 Zadanie

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

```
Procedure MaxMin(S : Set)
    if |S| = 1 then return (S[0], S[0]);
    else if |S| = 2 then return (max(S[0], S[1]), min(S[0], S[1]));
    else
         podziel S na dwa rownoliczne podzbiory S1, S2
           (z dokladnoscia do jednego elementu)
         (max1, min1) <- MaxMin(S1)
         (max2, min2) <- MaxMin(S2)
         return (max(max1,max2), min(min1,min2));
```

Operacja `return (max(S[0], S[1]), min(S[0], S[1]))` wykonuje jedno porównanie.

- Jak pokażemy na jednym z wykładów każdy algorytm dla tego problemu, który na elementach zbioru wykonuje jedynie operacje porównania, musi wykonać co najmniej $\lceil \frac{3}{2}n - 2 \rceil$ porównania. Dla jakich danych powyższy algorytm wykonuje tyle porównań? Podaj wzorem wszystkie takie wartości.
- Jak bardzo może różnić się liczba porównań wykonywanych przez algorytm od dolnej granicy?
- Popraw algorytm, tak by osiągał on tę granicę dla każdej wartości n?

2 Rozwiązanie

2.1 Wzór optymalny

Optymalną ilość porównań będziemy oznaczać jako $Opt(n)$.

$$Opt(n) = \lceil \frac{3}{2}n - 2 \rceil \tag{1}$$

2.2 Wzór jawny

Na początku, aby odpowiedzieć na zadane pytania, dobrze będzie mieć wzór nierekurencyjny na ilość porównań w procedurze `MaxMin`. Poza przypadkami brzegowymi, zbiór S jest dzielony na dwie równe części, z dokładnością do jednego elementu, więc jedna część ma $\lfloor \frac{n}{2} \rfloor$ elementów, a druga $\lceil \frac{n}{2} \rceil$. Ponadto procedura wykonuje potem 2 porównania. Wzór rekurencyjny przedstawia się więc następująco:

$$T(1) = 0; \quad T(2) = 1; \quad T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2 \tag{2}$$

Chcemy udowodnić, że następujący wzór jawny opisuje powyższy wzór rekurencyjny. Na czas dowodu, oznaczmy wzór jawny przez $T'(n)$, czyli pokażemy, że $\forall n \quad T(n) = T'(n)$. Dowód będzie indukcyjny względem k, gdzie $n = 2^{k+1} + r$.

$$T'(n = 2^{k+1} + r) = \begin{cases} 0, & \text{dla } n = 1, \\ 3 \cdot 2^k + 2r - 2, & \text{dla } 0 \leqslant r \leqslant 2^k, \\ 4 \cdot 2^k + r - 2, & \text{dla } 2^k < r < 2^{k+1}. \end{cases} \tag{3}$$

2.2.1 Podstawa indukcyjna

Weźmy $k = 0$. $r \in 0, 1$ - przypadki oczywiste. Zauważmy też, że $T'(1) = T(1)$ (z definicji).

2.2.2 Krok indukcyjny

Załóżmy że teza zachodzi $\forall k' \leqslant k - 1$, czyli dla $n < 2^{k+1}$. Rozpatrzmy $n = 2^{k+1} + r$.

Niech $0 \leqslant r \leqslant 2^k$. Wtedy wzór jawny ma postać

$$T'(n) = 3 \cdot 2^k + 2r - 2 \tag{4}$$

Rozwińmy wzór rekurencyjny i zauważmy, że będziemy korzystać z drugiego przypadku wzoru jawnego, tzn. zarówno $\lfloor \frac{r}{2} \rfloor$ jak i $\lceil \frac{r}{2} \rceil$ mieszczą się w przedziale $[0; 2^{k-1}]$.

$$\begin{align} T(2^{k+1} + r) &= T(2^k + \lfloor \frac{r}{2} \rfloor) + T(2^k + \lceil \frac{r}{2} \rceil) + 2 \tag{5} \\ &= (3 \cdot 2^{k-1} + 2\lfloor \frac{r}{2} \rfloor - 2) + (3 \cdot 2^{k-1} + 2\lceil \frac{r}{2} \rceil - 2) + \tag{6} \\ &= 3 \cdot 2^k + 2(\lfloor \frac{r}{2} \rfloor + \lceil \frac{r}{2} \rceil) - 2 = 3 \cdot 2^k + 2r - 2 \tag{7} \\ &= T'(2^{k+1} + r) \tag{8} \end{align}$$

Teraz rozpatrzmy drugi przypadek, gdy $2^k < r < 2^{k+1}$.

$$T(2^{k+1} + r) = T(2^k + \lfloor \frac{r}{2} \rfloor) + T(2^k + \lceil \frac{r}{2} \rceil) + 2 = * \tag{9}$$

- Niech $r = 2^k + 1$.

$$\begin{align} * &= (3 \cdot 2^{k-1} + 2 \cdot 2^{k-1} - 2) + (4 \cdot 2^{k-1} + 2^{k-1} + 1 - 2) + 2 \tag{10} \\ &= 4 \cdot 2^k + (2^k + 1) - 2 = Opt(n) \tag{11} \end{align}$$

- Niech $r = 2^{k+1} - 1$

$$* = (4 \cdot 2^{k-1} + 2^k - 1 - 2) + (3 \cdot 2^k + 2 \cdot 0 - 2) + 2 \tag{12}$$
$$= 4 \cdot 2^k + (2^{k+1} - 1) - 2 = Opt(n) \tag{13}$$

- Niech $r \in (2^k + 1; 2^{k+1} - 1)$

$$* = (4 \cdot 2^{k-1} + \lfloor \frac{r}{2} \rfloor - 2) + (4 \cdot 2^{k-1} + \lceil \frac{r}{2} \rceil - 2) + 2 \tag{14}$$
$$= 4 \cdot 2^k + r - 2 = Opt(n) \tag{15}$$

2.3 Wartości, dla których algorytm jest optymalny

Chcemy znaleźć dla jakich wartości n, algorytm zachowuje się w sposób optymalny, tzn. $T(n) = Opt(n)$. Jako że nasz wzór jawny jest podzielony na dwa przypadki, musimy je rozważyć oddzielnie. Przekształćmy najpierw jeszcze wzór na $Opt(n)$:

$$Opt(n) = \lceil \frac{3}{2} n - 2 \rceil = 3 \cdot 2^k + \lceil \frac{3}{2} r \rceil - 2 \tag{16}$$

Najpierw niech $0 \leqslant r \leqslant 2^k$.

$$3 \cdot 2^k + 2r - 2 = 3 \cdot 2^k + \lceil \frac{3}{2} r \rceil - 2 \qquad \Longrightarrow \qquad r = \lceil \frac{r}{2} \rceil \tag{17}$$

Stąd, $r \in 0, 1$.
Teraz niech $2^k < r < 2^{k+1}$.

$$4 \cdot 2^k + r - 2 = 3 \cdot 2^k + \lceil \frac{3}{2} r \rceil - 2 \qquad \Longrightarrow \qquad 2^k = \lceil \frac{r}{2} \rceil \tag{18}$$

Jako rozwiązanie równania pasuje zarówno $r = 2^{k+1} - 1$ jak i $r = 2^{k+1}$, jednak ten drugi przypadek wykracza poza dziedzinę r.

Podsumowując, algorytm zachowuje się optymalnie dla takich n, gdzie $r \in 0, 1, 2^{k+1} - 1$

2.4 Wartości, dla których algorytm najbardziej odbiega od optymalnego

Pomyślmy dla jakich wartości algorytm odbiega od optymalnego rozwiązania, zdefiniujmy $R(n) = T(n) - Opt(n)$. Po podstawowych rachunkach mamy:

$$R(n) = \begin{cases} \lfloor \frac{r}{2} \rfloor, & \text{dla } 0 \leqslant r \leqslant 2^k, \\ 2^k - \lceil \frac{r}{2} \rceil, & \text{dla } 2^k < r < 2^{k+1}. \end{cases} \tag{19}$$

Zauważmy, że pierwszy przypadek opisuje funkcję niemalejącą, a drugi nierosnącą. Stąd, maksymalna różnica musi być gdzieś po środku, tj. w okolicach $r = 2^k$.

$$R(2^{k+1} + 2^k - 1) = 2^{k-1} - 1 \tag{20}$$
$$R(2^{k+1} + 2^k) = 2^{k-1} \tag{21}$$
$$R(2^{k+1} + 2^k + 1) = 2^{k-1} - 1 \tag{22}$$
$$\tag{23}$$

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

2.5 Nowy algorytm

```
Procedure MaxMin(S : Set)
   if |S| = 1 then return (S[0], S[0]);
   else if |S| = 2 then return (max(S[0], S[1]), min(S[0], S[1]));
   else if jest_potega_dwojki(|S|) then
       S1, S2 <- podziel S na polowy
   else
       k = floor(log(2,|S|))
       S1 <- S[0 .. 2^k - 1]
       S2 <- S[2^k .. |S| - 1 ]
   (max1, min1) <- MaxMin(S1)
   (max2, min2) <- MaxMin(S2)
   return (max(max1,max2), min(min1,min2));
```

2.5.1 Dowód optymalności

Niech $U(n)$ określa ilość wykonywanych porównań.

$$U(n) = \begin{cases} 0 & \text{dla } n = 1 \\ 1 & \text{dla } n = 2 \\ 2 \cdot U(\frac{n}{2}) + 2 & \text{dla } n = 2^k \\ U(2^k) + U(r) + 2 & \text{w.p.p., } n = 2^k + r \end{cases} \tag{24}$$

Dowód indukcyjny po k, gdzie $n = 2^k + r$. Dla $n = 1, 2$ - oczywisty. Załóżmy że działa dla $k - 1$.

Niech $r = 0$.

$$U(2^k) = 2 \cdot U(\frac{n}{2}) + 2 = 2 \cdot \lceil \frac{3}{2}\frac{n}{2} - 2 \rceil + 2 \tag{25}$$

$$= \lceil \frac{3}{2} 2^k - 2 \rceil = Opt(2^k) \tag{26}$$

Niech $r \neq 0$.

$$U(2^k + r) = U(2^k) + U(r) + 2 = \lceil \frac{3}{2} 2^k - 2 \rceil + \lceil \frac{3}{2} r - 2 \rceil + 2 \tag{27}$$

$$= 3 \cdot 2^{k-1} + \lceil \frac{3}{2} r \rceil - 2 = Opt(2^k + r) \tag{28}$$

Stąd, $U(n) = Opt(n)$.