Algorytmy i Struktury Danych, 4. ćwiczenia

2018-10-24

Spis treści

1 Analiza algorytmu QuickSort

(AiSD, strony 56-58)
Dla losowej permutacji, oczekiwany czas sortowania przy użyciu algorytmu QuickSort wynosi:

$$T(n) = \begin{cases} 1 & \text{dla } n \leq 1 \\ (n+1) + \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) & \text{dla } n > 1 \end{cases}$$

Dla $n > 1$ możemy równanie przekształcić do:

$$T(n) = (n+1) + \frac{2}{n}\sum_{i=1}^{n} T(i-1)$$

(mnożymy obie strony przez n)

$$nT(n) = n(n+1) + 2\sum_{i=1}^{n} T(i-1)$$

dla $n-1$ otrzymujemy:

$$(n-1)T(n-1) = (n-1)n + 2\sum_{i=1}^{n-1} T(i-1)$$

po odjęciu stronami otrzymujemy:

$$T(n) = \frac{n+1}{n}T(n-1) + 2$$

co możemy przekształcić do:

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2}{n+1}$$

po przesumowaniu obu stron dla $n' = 1..n$

$$\frac{T(n)}{n+1} = \frac{T(0)}{1} + \sum_{i=1}^{n} \frac{2}{i+1}$$

Stosując przekształcenia możemy otrzymać:

$$T(n) = 2(n+1)(H_n + \frac{1}{n+1} - \frac{3}{2})$$

2 Dowód, że $n-1$ porównań jest potrzebne do znajdowania minimum

Weźmy algorytm, A, powiedzmy, za każdym razem, gdy porównuje on dwa elementy, to łączymy je krawędzią. Jeśli A użył mniej niż $n-1$ porównań, to istnieją dwa elementy, które nie są ze sobą porównywalne.

3 Optymalne znajdowanie drugiego co wielkości elementu

- budujemy drzewo turniejowe (porównujemy sąsiednie elementy, dalej przechodzi wygrany) — ten krok zabiera $n-1$ porównań,
- niech S zbiór elementów które przegrały z liderem, $|S| = \lceil \log n \rceil$
- wybierz lidera wśród elementów S — ten krok zabiera $|S|-1 = \lceil \log n \rceil - 1$ porównań.
- razem $n + \lceil \log n \rceil - 2$

Dowód, że algorytm jest optymalny. Knuth, tom III, 5.3.3. strona 221.

4 Optymalny algorytm do znajdowania min i max jednocześnie

Algorytm dziel i rządź. ($3\lceil n/2 \rceil - 2$ porównań)

- $Q = \emptyset$
- for $i = 1$ to $\lceil n/2 \rceil$ do $Q.push(pair(min(A[2i-1], A[2i]), max(A[2i-1], A[2i])))$
- while $|Q| > 1$ do
  - $(a_1, b_1) = Q.POP$,

- $(a_2, b_2) = Q.POP$,
- $Q.PUSH(min(a_1, a_2), max(b_1, b_2))$

- return $Q.POP$

Jest to również optymalna liczba porównań. (Knuth, Tom 3, ćwiczenie 16, strona 231). Niech (a, b, c, d) oznacza stan obliczeń algorytmu,

- a — liczba elementów, które nie były jeszcze porównywane,
- b — liczba elementów, które były porównywane i nie przegrały żadnego porównania,
- c — liczba elementów, które były porównywane i przegrały wszystkie porównania,
- d — liczba elementów, które wygrały co najmniej jedno porównanie, i przegrały co najmniej jedno porównanie.

Dowolny algorytm zaczyna obliczenia w stanie $(n, 0, 0, 0)$ i powinien kończyć w $(0, 1, 1, n-2)$ (jeśli kończy w innym to łatwo podać kontprzykład).

Konstrukcja przeciwnika dla algorytmu. Dla zapytania (x, y) postaci:

- (a_1, a_2) — odpowiada $a_1 < a_2$, zmiana $(-2, +1, +1, 0)$
- (b_1, b_2) — odpowiada $b_1 < b_2$, zmiana $(0, -1, 0, +1)$
- (c_1, c_2) — odpowiada $c_1 < c_2$, zmiana $(0, 0, -1, +1)$
- (a_1, b_1) — odpowiada $a_1 < b_1$, zmiana $(-1, 0, +1, 0)$
- (a_1, c_1) — odpowiada $a_1 > c_1$, zmiana $(-1, +1, 0, 0)$
- (a_1, d_1) — odpowiada $a_1 > d_1$, zmiana $(-1, +1, 0, 0)$
- (d_1, d_2) — odpowiada $d_1 < d_2$, zmiana $(0, 0, 0, 0)$
- (b_1, c_1) — odpowiada $b_1 > c_1$, zmiana $(0, 0, 0, 0)$
- (b_1, d_1) — odpowiada $b_1 > d_1$, zmiana $(0, 0, 0, 0)$
- (c_1, d_1) — odpowiada $c_1 < d_1$, zmiana $(0, 0, 0, 0)$

Dowolny algorytm musi zadać co najmniej $\lceil n/2 \rceil$ zapytań typu $(a, *)$, aby zmniejszyć licznik a o n, a co za tym idzie liczniki b i c zostaną sumarycznie zwiększone o n. Żeby zmniejszyć liczniki b i c o $n-2$ należy wykonać co najmniej $n-2$ zapytań typu $(b|c, *)$.