

# Algorytmy i Struktury Danych, 4. ćwiczenia

2008-10-27

## 1 Plan zajęć

- Pokazać, że Partition w QuickSortcie zachowuje losowość.
- Zanalizować QuickSort randomizowany. Pokazać, że oczekiwany czas wynosi  $O(n \log n)$ .
- QS bez stosu przy sortowaniu dodatnich liczb całkowitych,
- Załóżmy, że mamy heurystykę na wybór elementu do podziału, jako mediana z 3 elementów w ciągu - pierwszy, środkowy i ostatni. Jakie są złośliwe dane w tym przypadku.

## 2 Funkcja Partition w QuickSortcie

Dana jest permutacja  $\pi$  liczb z zakresu  $1, \dots, n$ . Niech  $s$  oznacza element dzielący ( $\pi[1]$ ) wybrany przez funkcję partition dla permutacji  $\pi$ . W wyniku działania partition, permutacja  $\pi$  zostaje podzielona na listy  $\pi_1$  (permutacja liczb  $1, \dots, (s-1)$ ),  $\{s\}$ ,  $\pi_2$  (permutacja liczb  $(s+1), \dots, n$ ). Udowodnij, że dla ustalonego  $s$ , jeśli  $\pi$  jest permutacją losową, to permutacje  $\pi_1$  i  $\pi_2$  również są losowe.

**Dowód:** Jeśli pominiemy w permutacji  $\pi$  element  $s$ , to dostajemy  $(n-1)!$  różnych permutacji. Musimy pokazać, że każdą parę permutacji  $\pi_1, \pi_2$  można uzyskać z tej samej liczby permutacji wejściowych. Wystarczy zauważyć, że każdą parę permutacji  $\pi_1, \pi_2$  można otrzymać z  $\binom{n-1}{s-1}$  permutacji wejściowych (dla każdego możliwego wyboru  $s-1$  pozycji wśród  $n-1$  elementów, które zajmują elementy mające znaleźć się w  $\pi_1$  możemy skonstruować permutację wejściową dającą podział  $\pi_1, \pi_2$ ).  $\square$

## 3 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:

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

Stosując przekształcenia możemy otrzymać:

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

## 4 QS bez stosu przy sortowaniu dodatnich liczb całkowitych

Zakładamy, że w tablicy możemy przechowywać liczby ujemne, i znak liczby będziemy traktować jako dodatkowy bit informacji.

```

l=0; r=n ;
while l < n do
    // niezmienniki: A[l... (r - 1)] > 0, ;
    // liczby z ujemny znakiem są już posortowane ;
    while l < r do
        m=Partition(l, r) ;
        A[m] = -A[m] ;
        r = m
    end
    while l < n and A[l] < 0 do l = l + 1 ;
    r = l + 1 ;
    while r < n and A[r] > 0 do r = r + 1
end

```

## 5 Scalanie w miejscu

Knuth, Tom III, strona 698.

- podziel tablicę na bloki rozmiaru  $\lceil \sqrt{n} \rceil$ , —  $Z_1, Z_2, \dots, Z_{m+2}$ , (blok  $Z_{m+2}$  może być mniejszy,
- zamień blok leżący na połączeniu dwóch ciągów, z blokiem  $Z_{m+1}$ , teraz każdy z bloków  $Z_1, \dots, Z_m$  jest uporządkowany,
- posortuj używając selection-sort bloki, wg. pierwszego elementu z bloków (jeśli dwa bloki mają ten sam element początkowy, to porównaj elementy końcowe)
- scal  $Z_1, \dots, Z_m$  używając  $Z_{m+1}$  jako bufora pomocniczego,
 

```

for i ∈ 1, ..., m - 1 do
    SimpleMerge( $Z_i, Z_{i+1}, Z_{m+1}$ )
end for

```

 (należy jeszcze pokazać, że taka procedura daje dobre uporządkowanie)
- dzielimy tablicę na trzy części:  $A, B, C$ ,  $|B| = |C| = 2\lceil \sqrt{n} \rceil$

- posortuj ostatnie  $4 \cdot \lceil \sqrt{n} \rceil$  elementów (bloki  $B, C$ ) używając InsertionSort (w rezultacie w bloku  $C$  znajdują się największe elementy w tablicy)
- scal bloki  $A$  i  $B$  używając  $C$  jako bufora
- posortuj blok  $C$  używając InsertionSort

Ćwiczenie: dlaczego używając selection-sort trzeba uwzględniać początki i końce bloków? Rozwiązanie: np. dla ciągów (111,123),(111,145) (rozmiar bloku 3), sortując jedynie po początkach moglibyśmy otrzymać: (123,145,111,111), który przy scalaniu metodą opisaną w algorytmie nie da uporządkowanego ciągu.

## 6 Budowa kopca w algorytmie HeapSort

Kopiec możemy budować idąc od dołu do góry. Zauważmy, że ostatnie  $n/2$  elementów spełnia warunki kopca, pozostaje jedynie poprawić porządek w pierwszych  $n/2$  elementach.

```

for  $i = \lfloor n/2 \rfloor$  downto 1 do
  DownHeap( $i$ )
end for

```

Koszt budowy kopca możemy opisać wzorem:

$$\frac{n}{4} \cdot 2 + \frac{n}{8} \cdot 4 + \frac{n}{16} \cdot 6 + \dots = n \cdot \sum_{i=1}^{\infty} \frac{i}{2^i} =$$

$$n \cdot \left( \sum_{i=1}^{\infty} \frac{1}{2^i} + \sum_{i=2}^{\infty} \frac{1}{2^i} + \sum_{i=3}^{\infty} \frac{1}{2^i} + \dots \right) = n \cdot \left( 1 + \frac{1}{2} + \frac{1}{4} + \dots \right) = 2n$$