Algorytmy i Struktury Danych, 2. ćwiczenia

2019-10-09

Spis treści

1 Analiza algorytmu InsertionSort

Algorithm 1: InsertionSort(A)

Input: array $A[1..n]$
Output: sorted array $A[1..n]$
for $i := 2$ **to** n **do**
  $j := i$
  while $j > 1$ ***and*** $A[j-1] > A[j]$ **do**
    swap $A[j-1]$ and $A[j]$
    $j := j - 1$

- złożoność optymistyczna: $\Theta(n)$ (dla ciągu $A = 1, 2, \ldots, n$),
- złożoność pesymistyczna: $\Theta(n^2)$ (dla ciągu $A = n, n-1, \ldots, 1$),
- średnia złożoność: koszt algorytmu to $O(n)$ + liczba inwersji w A, dla losowej permutacji oczekiwania liczba inwersji wynosi $f(n) = \Theta(n^2)$:

$$f(n) = \frac{1}{n} \cdot \sum_{i=0}^{n-1} i + f(n-1) = \frac{n-1}{/} 2 + f(n-1)$$

$$f(n) = \frac{n(n-1)}{4}$$

czyli średnia złożoność algorytmy InsertionSort to $\Theta(n^2)$.

2 Sortowanie metodą Shella

Lemat 1 *Niech m, n, r będą nieujemnymi liczba całkowitymi i niech $(x_1, \ldots, x_{m+r})$ oraz $(y_1, \ldots, y_{n+r})$ będą dowolnymi ciągami liczbowymi takimi, że $y_i \leq x_{m+i}$ dla $1 \leq i \leq r$. Jeśli elementy x oraz y posortujemy niezależnie tak, że $x_1 \leq \ldots \leq x_{m+r}$ oraz $y_1 \leq \ldots \leq y_{n+r}$ to nadal będziemy mieli $y_i \leq x_{m+i}$ dla $1 \leq i \leq r$.*

Po posortowaniu element x_{m+i} jest większy bądź równy od co najmniej $m+i$ elementów z x, wśród nich jest co najmniej i elementów które przed sortowaniem były na pozycjach $m, \ldots, m+r$, każdy z tych elementów ma wśród y element od którego jest większy, stąd x_{m+i} jest większy bądź równy od i najmniejszych elementów y.

(pełny dowód jest w Knuth, tom III, strona 94)

Lemat 2 *Jeśli tablica jest h posortowana i k posortujemy, to nadal będzie h posortowana.*

Niech a_i i a_{i+h} elementy które po sortowaniu nie są h posortowane. Niech Y ciąg zawierający a_i, a_{i+k}, a_{i+2k}, ... Niech X ciąg zawierający a_{i+h}, a_{i+h+k}, a_{i+h+2k}, ... Po k posortowaniu ciągi Y i X są uporządkowane, z poprzedniego lematu mamy jednak, że $a_i \leq a_{i+h}$ — sprzeczność.

Lemat 3 *Liczba porównań wymagana przy h posortowaniu tablicy rozmiaru n wynosi $O(n^2/h)$.*

Mamy h ciągów, każdy o długości n/h — stąd całkowity czas wynosi $h \cdot n^2/h^2 = n^2/h$.

Lemat 4 *Liczba porównań wymagana przy h_i posortowaniu tablicy rozmiaru n, która jest h_{i+1} i h_{i+2} posortowana wynosi $O(nh_{i+1}h_{i+2}/h_i)$ (przy założeniu, że h_{i+1} i h_{i+2} są względnie pierwsze)*

Trzeba pokazać, że jeśli ciąg jest h_{i+1} i h_{i+2} posortowany, to jeśli $k \geq h_{i+1}h_{i+2}$, to $a_i \leq a_{i+k}$.

Lemat 5 *Dla ciągu $h = \{2^i - 1 : i \in N\}$ algorytm ShellSort ma złożoność $O(n\sqrt{n})$.*

(Knuth, tom III, strona 95)

Niech B_i koszt i–tej fazy, $t = \lceil \log n \rceil$. Dla pierwszy $t/2$ przebiegów $h \geq \sqrt{n}$, ponieważ koszt jednej fazy jest ograniczona przez $O(n^2/h)$ stąd sumaryczny koszt jest rzędu $O(n^{1.5})$. Dla pozostałych przebiegów możemy skorzystać z poprzedniego lematu, koszt pojedynczej fazy jest równy $B_i = O(nh_{i+2}h_{i+1}/h_i)$, więc sumaryczny koszt tych faz jest również rzędu $O(n^{1.5})$.

Lemat 6 *Dla ciągu $h = \{2^i 3^j : i, j \in N\}$ algorytm ShellSort ma złożoność $O(n \log^2 n)$.*

Wszystkich faz algorytmu jest $O(\log^2 n)$. Trzeba pokazać, że każda z faz zajmuje czas $O(n)$.

Obserwacja: jeśli ciąg jest 2 i 3–uporządkowany, to jego 1–posortowanie wymaga czasu $O(n)$.

Analogicznie jeśli ciąg jest $2i$ oraz $3i$–uporządkowany to jego i–posortowanie wymaga czasu $O(n)$.

3 Przesunięcie cykliczne tablicy

Function CyclicLeftShift(a, k)

$n := len(a)$
Reverse(a, 1, $n - k$)
Reverse(a, $n - k + 1$, n)
Reverse(a, 1, n)

4 Stabilne i w miejscu sortowanie ciągów 0/1

Algorithm 2: StableMergeSort01(A)

Input: array $A[1..n]$
Output: number of elements with value 0
if $n = 1$ **then**
  return (1 if $A[1] = 0$ else 0)
else
  $m = \lfloor \frac{n}{2} \rfloor$
  c_1 = StableMergeSort01$(A[1, \ldots, m])$
  c_2 = StableMergeSort01$(A[m + 1, \ldots, n])$
  if $c_1 < m$ **and** $c_2 > 0$ **then**
    Swap$(A[c_1 + 1, \ldots, m]$, $A[m + 1, \ldots, m + c_2])$ – przy użyciu CyclicShift

Złożoność czasowa: $T(n) = 2T(n/2) + O(n) = O(n \log n)$. Złożoność pamięciowa: $O(\log n)$ (ale można to też zapisać w pamieci $O(1)$).

5 Kodowanie permutacji

Zadanie: dla zadanego wektora inwersji permutacji $w(\pi)$, odkoduj oryginalną permutację π.
Gdzie:

$$w(\pi)[i] = |\{j : 1 \leq j < i \text{ oraz } \pi[j] > \pi[i]\}|$$

Rozwiązanie: `https://www.mimuw.edu.pl/~jrad/asd/inwersje.pdf`