Algorytmy i Struktury Danych, 2. ćwiczenia

2008-10-13

1 Plan zajęć

- ShellSort
- udowodnić, że ciąg k-posortowany po h-sortowaniu pozostaje k-posortowany
- zaprojektować algorytm generowania kroków postaci $2^p 3^q$ on-line w porządku rosnącym
- Scalanie w miejscu dla ciągów długości $\sqrt{n}$ i $n - \sqrt{n}$
- Optymalne sortowanie dla 5 elementów
- Sortowanie ciągów 0-1 w miejscu i stabilnie
- Sortowanie permutacji

2 Optymalne sortowanie 5–ciu elementów

Niech $A = (a, b, c, d, e)$.

- $compare(a, b)$, (niech $a < b$)
- $compare(c, d)$, (niech $c < d$)
- $compare(a, c)$, (niech $a < c$)

- teraz wkładamy, e pomiędzy a, c, d,
  if $(e > c)$ then $compare(e, a)$ else $compare(e, d)$
- możemy otrzymać jeden z następujących posetów:

  każdy z nich można posortować używając 2 porównań.

3 Sortowanie metodą Shella, dowody

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 $h \cdot n^2/h^2 = n^2/h$. qued

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)*

TODO

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 $a_i \leq a_{i+k}$. qued

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 $h \geq sqrtn$, 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 $O((\log n)^2)$. Trzeba pokazać, że każda z nich zajmuje $O(n)$ czasu. Obserwacja — jeśli ciąg jest 2 i 3 uporządkowany, to jego 1–posortowanie wymaga $O(n)$ czasu. Analogicznie jeśli ciąg jest $2i$ i $3i$ posortowany to jego i posortowanie wymaga $O(n)$.

4 Algorytm generowania kroków postaci $2^p 3^q$ online

Algorytm 1: GenSequence()

```
1 h = emptyHeap(); seq = ∅ ;
2 h.add(1); ;
3 while true do
4 |   x = h.extractMin() ;
5 |   seq.add(x) ;
6 |   h.add(3 · x) ;
7 |   if h mod 3 ≠ 0 then
8 |   |   h.add(2 · x) ;
```

5 Sortowanie ciągu 0–1

Stabilne sortowanie tablicy 0/1 w miejscu.

Rozwiązanie $O(n \log n)$:

Algorytm 2: Sort01()

```
   /* merge(l, m, r) - scala dwa ciągi (l..m − 1) i (m..r) postaci
      0*1* w jeden ciag (l, r)                                  */
1  for i = 0 to ⌈log n⌉ do
2  |   b = 2^i; j = 1;
3  |   while j < n do
4  |   |   merge(j, j + b, j + 2 · b − 1);
5  |   |   j = j + 2 · b;
```

6 Scalanie w miejscu dla ciągów długości $\sqrt{n}$ i $n - \sqrt{n}$

Algorytm 3: Merge(A)

1 Dana jest tablica A zawierająca dwa uporządkowane rosnąco ciągi: $1..n - \sqrt{n}$ i $n - \sqrt{n} + 1..n$. ;
2 Posortuj (używając alg. insertion sort) ciąg $n - 2\sqrt{n} + 1..n$;
3 Scal ciąg $1..n - 2\sqrt{n}$ i $n - 2\sqrt{n} + 1..n - \sqrt{n}$ używając obszaru $n - \sqrt{n} + 1..n$ jako bufor ;
4 Posortuj (używając alg. insertion sort) ciąg $n - \sqrt{n} + 1..n$;