

Algorytmy i Struktury Danych, 4. ćwiczenia

2016-10-28

Spis treści

1	Sortowanie liczb z zakresu $0..n^3$	1
2	Sortowanie słów o różnych długościach	1
2.1	Sortowanie słów o różnych długościach, pierwsza próba	1
2.2	Sortowanie słów o różnych długościach	2
3	Izomorfizm drzew	2
4	Izomorfizm drzew — algorytm dla drzew nieskierowanych	3
5	Dowód, że $n - 1$ porównań jest potrzebne do znajdowania minimum	3
6	Optymalne znajdowanie drugiego co wielkości elementu	3
7	Rozgłaszanie komunikatów	4

1 Sortowanie liczb z zakresu $0..n^3$

`Sort(A)`

- 1: posortuj stabilnie ciąg A wg $A[i] \bmod n$
- 2: posortuj stabilnie ciąg A wg $\lfloor A[i]/n \rfloor \bmod n$
- 3: posortuj stabilnie ciąg A wg $\lfloor A[i]/n^2 \rfloor \bmod n$

2 Sortowanie słów o różnych długościach

2.1 Sortowanie słów o różnych długościach, pierwsza próba

Niech $S = \{W_1, \dots, W_n\}$ zbiór słów do posortowania, niech n_i oznacza długość słów W_i .

`RADIXSORT2(G)`

- 1: wyzeruj tablicę L ,
- 2: **for all** $W \in S$ **do**
- 3: dodaj W na koniec listy $L[n_i]$.
- 4: **end for**
- 5: niech $n = \max\{n_i : i \in 1, \dots, n\}$,

```

6:  $S = L[n]$ 
7: for all  $i \in n, \dots, 1$  do
8:   wyzeruj tablicę  $A$ ,
9:   for all  $W \in S$  do
10:    dodaj  $W$  na koniec listy  $A[W[i]]$ 
11:   end for
12:    $S =$  złączenie listy  $L[i - 1]$  i list z tablicy  $A$  (w tej kolejności)
13: end for

```

Niestety powyższy algorytm ma złożoność $O(|\Sigma|n)$, a my potrzebujemy $O(|\Sigma| + n)$.

2.2 Sortowanie słów o różnych długościach

RADIXSORT3(G)

```

1: przygotuj zbiór par  $P = \{(i, x) : W[j][i] = x\}$ ,
2: posortuj  $P$ 
3: wyzeruj tablicę  $L$ ,
4: for all  $W \in S$  do
5:   dodaj  $W$  na koniec listy  $L[n_i]$ .
6: end for
7: niech  $n = \max\{n_i : i \in 1, \dots, n\}$ ,
8:  $S = L[n]$ 
9: for all  $i \in n, \dots, 1$  do
10:  for all  $(i, x) \in P$  do
11:    $A[x] = nil$ 
12:  end for
13:  for all  $W \in S$  do
14:   dodaj  $W$  na koniec listy  $A[W[i]]$ 
15:  end for
16:   $S = L[i - 1]$ 
17:  for all  $(i, x) \in P$  do
18:    $S = S \cup A[x]$ 
19:  end for
20: end for

```

3 Izomorfizm drzew

Algorytm:

TREEISOMORPHISM(T1, T2, DEPTH)

```

1: if  $T1.height > depth$  then
2:   return  $(T1.height = T2.height)$ ;
3: end if
4: if not TREEISOMORPHISM(T1, T2, DEPTH+1) then
5:   return false;
6: end if
7: for  $v \in T1.nodes[depth + 1] \cup T2.nodes[depth + 1]$  do
8:   {w porządku rosnących etykiet}
9:   dodaj  $value(v)$  do listy wierzchołka  $parent(v)$ 

```

10: **end for**
 11: posortuj leksykograficznie listy $value(v)$ dla $v \in T1.nodes[depth]$
 12: posortuj leksykograficznie listy $value(v)$ dla $v \in T2.nodes[depth]$
 13: porównaj czy listy są identyczne, jeśli nie to **return false**
 14: zamień etykiety $value(v)$ na liczby z zakresu $1, \dots, n$
 15: **return true**

4 Izomorfizm drzew — algorytm dla drzew nieskierowanych

Znajdź w drzewach centroidy (każde drzewo zawiera co najwyżej 2 centroidy), dla każdej kombinacji ukorzeń drzewa w centroidach i uruchom poprzedni algorytm.

Niech $w(x) = \max\{|subtree(t_i)| : t_i \in adj(x)\}$. *Centroid* — wierzchołek o minimalnej wadze $w(x)$.

FIND(v)

1: niech c_1, \dots, c_k synowie wierzchołka v ,
 2: jeśli $subtree(c_i) \leq n/2$ dla $1 \leq i \leq k$, to **return v**,
 3: wpp. niech c_j wierzchołek, taki, że $subtree(c_j) > n/2$ (jest tylko jeden o tej własności),
 4: **return** FIND(c_j)

FINDCENTROID(v)

1: ukorzeń drzew w dowolnym wierzchołku r ,
 2: oblicz wartości $subtree(v)$ dla wszystkich wierzchołków,
 3: **return** FIND(r)

5 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.

6 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.

7 Rozgłaszanie komunikatów

Dane drzewo T , należy obliczyć czas potrzebny na przesłanie komunikatów do wszystkich węzłów drzewa. Przesłanie komunikatu po jednej krawędzi zajmuje 1 jednostkę czasu.

Algorytm $O(n \log n)$:

- jeśli wierzchołek jest liściem to $czas = 0$,
- wpp. rekurencyjnie oblicz czas potrzebny na rozgłoszenie w poddrzewach,
- posortuj malejąco otrzymane czasy: t_1, \dots, t_k
- $czas = \max\{i + t_i : 1 \leq i \leq k\}$

Aby otrzymać algorytm $O(n)$ trzeba sprytnie obliczać wartości atrybutu $czas$.

- $Q = \{ \text{liście } T \}$,
- while $root \notin Q$ do
 - $x = Q.extractMin()$
 - dodaj $x.czas$ do kolejki $parent(x)$,
 - jeśli $parent(x)$ ma już pełną listę poddrzew, to policz $parent(x).czas$ i dodaj $parent(x)$ do kolejki.

Kolejkę Q można zaimplementować w tablicy (i -ty element tablicy zawiera listę wierzchołków o wartości $x.czas = i$). Sumarycznie operacje $extractMin$ zajmą czas $O(n)$. Dodawanie do kolejki zajmuje czas $O(1)$.