Algorytmy i Struktury Danych, 5. ćwiczenia

2018-10-31

Spis treści

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

SORT(A)
1: posortuj stabilnie ciąg A wg $A[i] \mod n$
2: posortuj stabilnie ciąg A wg $\lfloor A[i]/n \rfloor \mod n$
3: posortuj stabilnie ciąg A wg $\lfloor A[i]/n^2 \rfloor \mod n$

2 Sortowanie słów o różnych długościach

2.1 Sortowanie słów o różnych długościach, pierwsza próba

Niech $S = \{W_1, \ldots, W_n\}$ 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 $n = max\{n_i : i \in 1, \ldots, n\}$,
6: $S = L[n]$
7: **for all** $i \in n, \ldots, 1$ **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 $n = max\{n_i : i \in 1,\ldots,n\}$,
8: $S = L[n]$
9: **for all** $i \in n,\ldots,1$ **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 $1,\ldots,n$

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 $c1, \ldots, c_k$ 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 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: $t_1, \ldots, t_k$
- $czas = max\{i + t_i : 1 \leq i \leq k\}$

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

- $Q = \{$ 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 $O(1)$.