Algorytmy i Struktury Danych, 5. ćwiczenia

2019-10-30

Spis treści

1 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:

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

(mnożymy obie strony przez n)

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

dla $n-1$ otrzymujemy:

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

po odjęciu stronami otrzymujemy:

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

co możemy przekształcić do:

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

po przesumowaniu obu stron dla $n' = 1..n$

$$\frac{T(n)}{n+1} = \frac{T(0)}{1} + \sum_{i=1}^{n} \frac{2}{i+1}$$

Stosując przekształcenia możemy otrzymać:

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

2 Klasówka 2009 (1), zadanie 1

Dana jest tablica $n \times n$, $n > 1$, w której w każde pole wpisano liczbę całkowitą. Chcemy przejść z dolnego lewego rogu (z $(1,1)$) do górnego prawego rogu (do (n,n)) i wrócić, idąc w drodze z (1,1) zawsze w prawo lub w górę, a z powrotem - w lewo lub w dół. Z danego pola można przejść tylko na pola sąsiednie (współrzędne różnią się o 1 na dokładnie jednej pozycji). Żadne pole nie może się pojawić na całej trasie (czyli tam i z powrotem) więcej niż raz, poza polem (1,1), które pojawia się na początku i na końcu trasy. Zaprojektuj algorytm znajdowania najtańszej trasy, czyli takiej, na której suma wartości pól jest najmniejsza. **Rozwiązanie:** dynamik po przekątnych

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 $O(1)$.