Algorytmy i Struktury Danych, 13. ćwiczenia

2019–01–16

Spis treści

1 Zadanie Graf Inwersji

Dana permutacja π_n, definiujemy graf

$$G = (V = \{1, \ldots, n\}, E = \{(i, j) : i < j \text{ i } \pi_i > \pi_j\})$$

Należy wyznaczyć spójne składowe w grafie G.
Rozwiązanie (trickowe):

Lemat 1. *Niech π_k to permutacja liczb $\{1, \ldots, k\}$, taka, że:*

- *dla $1 \leq i < k$:* $\max\{\pi[1, \ldots, i]\} \neq i$,
- $\max\{\pi[1..k]\} = k$.

To graf inwersji permutacji π_k jest spójny.

Dowód. Załóżmy, że dla udowodniliśmy, że $\pi[i + 1, \ldots, k]$ tworzą jedną spójną składową.

Dla dowolnego wierzchołka $i < k$, niech $\pi[x] = \max(\pi[1, \ldots, i] > i$, $\pi[y] = \min(\pi[i + 1, \ldots, k] \leq i$. Mamy dwa przypadki:

- $x = i$, stąd istnieje inwersja (i, y) i wierzchołek i leży w tej samej spójnej składowej co $i + 1, \ldots, k$.
- $x < i$, stąd istnieją inwersje (x, i) i (x, y) więc również wierzchołek i leży w tej samej spójnej składowej co $i + 1, \ldots, k$.

□

$i = 1$; $CurrMax = 0$; **for** $j = 1, \ldots, n$ **do**
  $CurrMax = \max(CurrMax, \pi_j)$
  if $CurrMax = j$ **then**
    wierzchołki $\pi_i, \ldots, \pi_j$ tworzą spójną składową
    $i = j + 1$; $CurrMax = 0$;

2 Problem 21–1, Minimum "off–line"

Dany ciąg operacji INSERT(x) ($x \in 1, \ldots, n$, każda wartość jest dodawana co najwyżej 1 raz). oraz EXTRACT-MIN. Należy obliczyć rezultaty poszczególnych operacji EXTRACT-MIN (należy pamiętać, że cały ciąg operacji jest z góry dany).
Przykład:

$$4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5$$

Rozwiązanie: Rozbijamy ciąg wywołań na podciągi jednorodne:

$$I_1, E, I_2, \ldots, I_m, E, I_{m+1}$$

Gdzie każdy zbiór I_j to jakiś podzbiór kluczy (być może pusty!).

Algorithm 1: Off-Line-Minimum

for $i \in 1, \ldots, n$ **do**
  wyznacz j takie, że $i \in I_j$
  if $j \neq m + 1$ **then**
    $extracted[j]=i$
    niech l będzie najmniejszą wartością wiekszą niż j, dla której zbiór I_l istnieje
    $I_l = I_j \cup I_l$ (zbiór I_j zostaje zniszczony)

3 Problem wyznaczania głębokości

(w nowym wydaniu Cormena, problem na numer 21–2)
Dany jest las $\mathcal{F} = \{T_i\}$ ukorzenionych drzew z trzema operacjami:

- Make-Tree(v) tworzy drzewo składające się z węzła v,
- Find-Depth(v) zwraca głębokość węzła v w jego drzewie
- Graft(r,v) ustawia jako ojca węzła r węzeł v (zakładamy, że r jest korzeniem swojego drzewa T, oraz $v \notin T$)

Algorithm 2: Make-Tree(v)

Make-Set(v) (czyli $link[v] = nil$, $size[v] = 1$)
$parent[v]$=nil (ojciec wierzchołka v w lesie $\mathcal{F}$)
$d[v]$=0 (pseudo-głębokość v)

Algorithm 3: Find-Depth(v)

(symulujemy $Find(v)$ i sumujemy wartości $d[v]$ na ścieżce wyznaczonej przez wskaźniki $link$)
if $link[v] = nil$ **then**
  return $d[v]$
else
  niech $u = link[v]$
  $d_1 =$ Find-Depth(u)
  if $link[u] \neq nil$ **then**
    $d[v] += d[u]$
    $link[v] = link[u]$

Algorithm 4: Graft(r, v)

$parent[r] = v$
$h =$ Find-Depth(v)
$r' = Find(r)$
$v' = Find(v)$
$d[r'] += h + 1$
if $size[r'] \leq size[v']$ **then**
  $link[r'] = v'$
  $size[v'] += size[r']$
else
  (w Find-Union podłączamy węzły odwrotnie niż w lesie)
  $link[v'] = r'$
  $size[r'] += size[v']$
  $d[v'] = d[v'] - d[r']$

4 System różnych reprezentantów

Dana jest rodzina I, n niepustych podzbiorów zbioru $\{1, 2, \ldots, n\}$, z których każdy to całkowitoliczbowy przedział postaci $[i, j]$, $i \leq j$. Zaprojektuj efektywny algorytm sprawdzania, czy zadana rodzina posiada system różnych reprezentantów, a jeśli tak, to podaje jeden z nich.

Algorithm 5: SystemRóżnychReprezentantów(I)

```
for i ∈ 1, ..., n + 1 do
    Make-Set(i)
    Last[i] = i
posortuj przedziały I wg. drugiej i pierwszej współrzędnej
for [l, r] ∈ I do
    i = Last[Find-Set(l)]
    if i ≤ r then
        przypisz i jako reprezentanta [l, r]
        i' = Last[Find-Set(i + 1)]
        Union(i, i')
        Last[Find-Set(i')] = i'
    else
        BRAK ROZWIĄZANIA
```

5 Egzamin poprawkowy 2010/2011, Zadanie 3

Zaproponuj implementację struktury danych udostępniającej operacje struktury Find-Union dla elementów 1..n z przypisanymi całkowitoliczbowymi wartościami (początkowo same zera) oraz dwie nowe operacje:

Add(i, a) :: do wartości wszystkich elementów ze zbioru zawierającego element i dodaj wartość a

Value(i) :: podaj aktualną wartość przypisaną elementowi i

Rozwiązanie: Do każdego węzła drzewa find-union dodaj dodatkowy atrybut Δ początkowo wypełniony wartościami 0.

$Value(i)$ zaimplementowana jest jako zwrócenie sumy wartości Δ na ścieżce od węzła i do korzenia zbioru.

$Add(i, a)$ lokalizuje korzeń zbioru zawierający element i i dodaje do niego wartość a.

Dla standardowych operacji Find-Union, należy uważać na:

- kompresje ścieżek (trzeba aktualizować wartości Δ w węzłach),
- Union (trzeba zapewnić własność, że wartości elementów podłączanego drzewa nie zmienią się).

6 Klasówka 2012 (2), zadanie 1

Zaprojektuj strukturę danych, która umożliwia efektywne wykonywanie ciągu operacji Łącz(u,v) i Głębokość(u) na lesie drzew ukorzenionych o zbiorze wierzchołków $1, 2, \ldots, n$. Początkowo każde drzewo jest jednowierzchołkowe. Operacja Łącz(u,v) polega na połączeniu dwóch różnych drzew o korzeniach u i v w jedno drzewo o korzeniu v, poprzez uczynienie u synem v (podwiązanie u do v). Operacja Głębokość(u) polega na wyznaczeniu głębokości wierzchołka u w aktualnie zawierającym go drzewie w lesie. Podaj sposób i koszt inicjacji

swojej struktury danych, a następnie koszt wykonania każdej z operacji Łącz i Głębokość w zaprojektowanym przez siebie rozwiązaniu.

Rozwiązanie:

Do każdego węzła Find-Union dodajemy atrybut Δ (początkowo równy 0), który będzie zawierał względną odległość do ojca.

Algorithm 6: Głębokość(v)

```
v' = Find(v)
if v ≠ v' then
    return Δ(v) + Δ(v')
else
    return Δ(v)
```

Algorithm 7: Find(v)

```
if v = p(v) then
    return v
else
    p' = p(v)
    v' = Find(p')
    p(v) := v' - kompresja ścieżki
    if v' ≠ p' then
        Δ(v) := Delta(v) + Delta(p')
    return v'
```

Algorithm 8: Łącz(u, v)

```
u' := Find(u)
v' := Find(v)
łączymy według rozmiarów drzew (mniejsze do większego) if
 size(u') < size(v') then
    p(u') := v'
    Δ(u') := Δ(u') + 1
else
    p(v') := u'
    Δ(u') := Δ(u') + 1
    Δ(v') := Δ(v') − Δ(u')
```