

# Algorytmy i Struktury Danych, 14. ćwiczenia

2022-01-19

## Implementacja struktury Find-Union

---

**Algorithm 1:**  $\text{Init}(n)$

---

```
foreach  $i \in \{1..n\}$  do
   $p[i] = -1$ 
   $size[i] = 1$ 
```

---

---

**Algorithm 2:**  $\text{Find}(i)$

---

```
if  $p[i] = -1$  then
  return  $i$ 
else
   $p[i] := \text{Find}(p[i])$ 
  return  $p[i]$ 
```

---

---

**Algorithm 3:**  $\text{Union}(i, j)$

---

```
 $i = \text{Find}(i)$ 
 $j = \text{Find}(j)$ 
if  $i \neq j$  then
  if  $size[j] > size[i]$  then
     $(i, j) = (j, i)$ 
   $p[j] = i$ 
   $size[i] = size[i] + size[j]$ 
```

---

## Zadanie 12.1

(w nowym wydaniu Cormena, problem na numer 21-2)

Dany jest las  $\mathcal{F} = \{T_i\}$  ukorzenionych drzew z trzema operacjami:

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

---

**Algorithm 4: 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 5: 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  $u = link[v]$ 
  |  $d_1 = \text{Find-Depth}(u)$ 
  | if  $link[u] \neq nil$  then
  |   |  $d[v] += d[u]$ 
  |   |  $link[v] = link[u]$ 
```

---

---

**Algorithm 6: Graft( $r, v$ )**

---

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

---

## Zadanie 12.2

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  $\Delta$  początkowo wypełniony wartościami 0.

$Value(i)$  zaimplementowana jest jako zwrócenie sumy wartości  $\Delta$  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  $\Delta$  w węzłach),
- Union (trzeba zapewnić własność, że wartości elementów podłączanego drzewa nie zmieniają się).

### Zadanie 12.3

Mamy  $n$  kul ponumerowanych od 1 do  $n$ . Na początku wszystkie kule są zielone. Na kulach wykonujemy następujące operacje:

**Pokoloruj(a, b, kol):**  $1 \leq a \leq b \leq n$ ,  $kol \in \{\text{zielony, czerwony}\}$  — pokoloruj kule o numerach od  $a$  do  $b$  na kolor  $kol$ ,

**Kolor(a):**  $1 \leq a \leq n$  — podaj kolor kuli o numerze  $a$ .

- Zaproponuj strukturę danych, która umożliwi efektywne wykonywanie ciągu operacji Pokoloruj i Kolor.
- Załóżmy, że na początku wykonujemy  $m \geq n$  z góry znanych operacji Pokoloruj, a następnie pytamy o kolor każdej kuli. Zaproponuj efektywny algorytm obliczający kolory kul po wykonaniu wszystkich operacji Pokoloruj.

**Rozwiązanie:** Punkt a) zadanie “Malowanie Autostrady” z laboratorium – drzewo przedziałowe, wszystkie operacje w czasie  $O(\log n)$ .

Punkt b) Tworzymy strukturę Find-Union, która dodatkowo dla każdego zbioru przechowuje:  $min$ ,  $max$ ,  $vis$  (początkowo false). Dodatkowo utrzymujemy tablicę KOL[1, ..., n] (początkowo wypełnioną kolorem zielonym).

Następnie wykonujemy operacje Pokoloruj od ostatniej do pierwszej (przy czym nasza implementacja koloruje tylko jeszcze niepokolorowane elementy):

---

**Algorithm 7:** Pokoloruj( $a, b, kol$ )

---

```
s = Find(a)
while max(s) ≤ b do
    ▷ niezmiennik: size(s) = 1 lub vis(s) = True
    if vis[max(s)] = False then
        KOL[max(s)] = kol
        vis[max(s)] = True
        s = Find(max(s) + 1)
s = Find(a)
while max(s) < b do
    s := Union(s, max(s) + 1)
```

---

## Zadanie 12.4

Dany ciąg operacji INSERT( $x$ ) ( $x \in 1, \dots, n$ , 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:

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

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

---

**Algorithm 8:** Off-Line-Minimum

---

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

---

## Zadanie 12.5

Dokonaj analizy rozwiązania problemu Find-Union ze zrównoważaniem drzew i kompresją ścieżek, przy założeniu że operacje Find wykonywane są dopiero po wykonaniu wszystkich operacji Union.

**Rozwiązanie:** Zakładam, że dla wszystkich operacji Union argumenty wskazują na reprezentantów zbiorów (więc czas wykonania pojedynczej operacji to  $O(1)$ ). Gdyby było inaczej można symulować operacje Find za pomocą  $Union(i, i)$ .

Teraz pozostaje nam pokazać, że dowolny ciąg  $m$  operacji Find na  $n$  zbiorach nie zajmie więcej niż  $O(n + m)$ .

Pokolorujmy krawędzie lasu Find-Union na dwa kolory, zielony jeśli krawędź prowadzi do reprezentanta zbioru i niebieski wpp. Zauważmy, że lasie Find-Union jest co najwyżej  $n - 2$  krawędzi niebieskich. Jeśli operacja Find przechodzi po  $k$  krawędziach, to oznacza, że przechodzi o  $k - 1$  krawędziach niebieskich i 1 zielonej. Ze względu na kompresję ścieżki, każda krawędź niebieska jest zamieniana na zieloną. Ponieważ mamy ograniczoną liczbę krawędzi niebieskich i po każdej jesteśmy w stanie przejść tylko raz, stąd całkowity czas wykonania operacji Find to  $O(n + m)$ .

## Off-line LCA

(w nowym wydaniu Cormena, problem na numer 21–3)

Dane jest drzewo  $T$ , oraz ciąg  $P$ , zapytań postaci  $LCA(x, y)$  — pytanie o najniższego wspólnego przodka węzłów  $x$  i  $y$ .

---

**Algorithm 9:** LCA( $u$ )

---

```

Make-Set( $u$ )
ancestor[Find-Set( $u$ )]= $u$ 
for  $v \in adj(u)$  do
    LCA( $v$ )
    UNION( $u, v$ )
    ancestor[Find-Set( $u$ )]= $u$ 
color[( $u$ )]=CZARNY
for  $v : \{u, v\} \in P$  do
    if color[ $v$ ]=CZARNY then
        najniższym wspólnym przodkiem  $u$  i  $v$  jest ancestor[Find-Set( $v$ )]

```

---

Trzeba uzasadnić:

- dla każdej pary  $(u, v) \in P$  udzielona zostanie dokładnie jedna odpowiedź,
- poprawność algorytmu,

## System różnych reprezentantów

Dana jest rodzina  $I$ ,  $n$  niepustych podzbiorów zbioru  $\{1, 2, \dots, n\}$ , 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 10:** SYSTEMRÓŻNYCHREPREZENTANTÓW( $I$ )

---

```

for  $i \in 1, \dots, n + 1$  do
    MAKE-SET( $i$ )
    Last[ $i$ ] =  $i$ 
posortuj przedziały  $I$  wg. drugiej i pierwszej współrzędnej
for  $[l, r] \in I$  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

```

---