Algorytmy i Struktury Danych, 12. ćwiczenia (wersja 1.0.1)

2024-01-15

Implementacja struktury Find-Union

Algorytm 1: Init(n)

```
foreach i ∈ {1..n} do
    p[i] = −1
    size[i] = 1
```

Algorytm 2: Find(i)

```
if p[i] = −1 then
    return i
else
    p[i] := Find(p[i])
    return p[i]
```

Algorytm 3: Union(i, j)

```
i = Find(i)
j = Find(j)
if i ≠ j then
    if size[j] > size[i] then
        (i, j) = (j, i)
    p[j] = i
    size[i] = size[i] + size[j]
```

Zadanie 11.2

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 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).

Zadanie 11.1

(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 $v \notin T$)

W naszym rozwiązaniu do reprezentacji lasu ukorzenionych drzew będziemy utrzymywać strukturę Find-Union. W strukturze Find-Union wskaźniki $p[v]$ nie muszą odpowiadać strukturze lasu, jednak za pomocą dodatkowego atrybutu $d[v]$ (pseudo-głębokość) będziemy mogli zapewnić obliczanie Find-Depth(v).

W trakcie działania algorytmu utrzymujemy następujący niezmiennik: jeśli wierzchołek v ma w lesie ukorzenionych drzew głębokość h (czyli Find-Depth(v)=h), a w strukturze Find-Union mamy następującą ścieżkę:

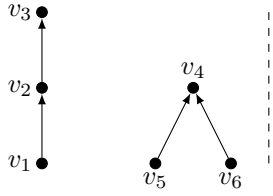
$$p_0 = v, p_1 = p[v], p_2 = p[p[v]], \ldots, p_k = p[p_{k-1}], p_{k+1} = p[p_k] = \text{nil}$$

to

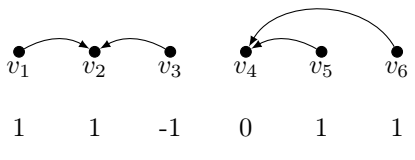
$$h = \sum_{i=0}^{k} d[p_i].$$

Przykładowy stan lasu i struktury Find-Union:

Las drzew ukorzenionych



Struktura Find-Union



Algorytm 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)

Algorytm 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 $u = link[v]$
    $d_1 =$ Find-Depth(u)
    if $link[u] \neq nil$ **then**
        $d[v] += d[u]$
        $link[v] = link[u]$

Algorytm 6: Graft(r, v)

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

Zadanie 11.3

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

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

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

a) Zaproponuj strukturę danych, która umożliwi efektywne wykonywanie ciągu operacji Pokoloruj i Kolor.

b) 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$[1, \ldots, n]$ (początkowo wypełnioną kolorem zielonym).

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

Algorytm 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 11.4

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!).

Algorytm 8: Off-Line-Minimum

```
for i ∈ 1, ..., n do
    wyznacz j takie, że i ∈ I_j
    if j ≠ m + 1 then
        extracted[j]=i
        niech l będzie najmniejszą wartością wiekszą niż j, dla której
          zbiór I_l istnieje
        I_l = I_j ∪ I_l (zbiór I_j zostaje zniszczony)
```

Zadanie 11.5

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 lesie 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.

Algorytm 9: LCA(u)

```
Make-Set(u)
ancestor[Find-Set(u)]=u
for v ∈ adj(u) do
    LCA(v)
    UNION(u, v)
    ancestor[Find-Set(u)]=u
color[(u)]=CZARNY
for v : {u, v} ∈ 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,
- poprawnosć algorytmu,

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.

Algorytm 10: 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
```

Zadanie 12.1

Dana jest tablica $P[0, \ldots, n]$ nieujemnych liczb całkowitych.

a) Zaproponuj algorytm, który efektywnie sprawdzi, czy P jest tablicą prefiksów-sufiksów z algorytmu KMP dla pewnego słowa nad alfabetem $\{a, b\}$?

Uwaga: kolejne symbole słowa są indeksowane od 1.

$P[0]$ odpowiada słowu pustemu.

b) Czy istnieje słowo nad alfabetem $\{a, b\}$, dla którego $P = [0, 0, 1, 0, 1, 2, 3, 4, 1, 2]$ jest tablicą prefiksów-sufiksów?

Rozwiązanie: a) Bez straty ogólności możemy założyć, że rekonstruowane słowo rozpoczyna się od litery a. Dla dowolnego $i > 1$, jeśli $P[i] = 0$ to dodajemy literę b, wpp. $P[i] < i$ i możemy użyć litery o indeksie $P[i]$ z rekonstruowanego słowa. Następnie obliczamy tablicę P dla tak otrzymanego słowa i porównujemy z wejściową tablicą.

b) używając algorytmu otrzymujemy słowo $S = aabaabaaa$, które niestety nie przechodzi weryfikacji ($P = [0, 0, 1, 0, 1, 2, 3, 4, \mathbf{5}, 2]$)

Zadanie 12.2

W tym zadaniu rozważamy słowa na alfabetem binarnym.

a) Ile wynosi suma elementów tablicy P dla słowa $(01)^{2020}$?

b) Ile wynosi wysokość drzewa sufiksowego (liczona liczbą krawędzi) dla słowa $(01)^{2020}$?

c) Zaprojektuj efektywny algorytm, który dla dodatniej liczby całkowitej k oraz słów x, y takich, że $|x| = k$, $|y| = k^2$ sprawdzi, ile jest w y podsłów o długości k, z których każde różni się od x na dokładnie jednej pozycji.

Rozwiązanie: a) $\sum_{i=1}^{4038} i$, tablica P ma postać $[0, 0, 0, 1, 2, 3, 4, 5, \ldots, 4038]$.

b) 2020

c) [rozwiązanie proste dla $|y| = k^2$] Budujemy zbiór słów $X = \{x_i : 1 \leq i \leq k\}$ gdzie x_i oznacza słowo x z zanegowanym i-tym znakiem. Uruchamiany algorytm wyszukiwania wielu wzorców X w tekście y (alg. Aho-Corasick) działający w czasie $O(\sum_{i=1}^{k} |x_i| + |y|) = O(k^2)$.

c) [rozwiązanie bardziej ogólne dla dowolnej długości y] Budujemy drzewa sufiksowe dla $x\$y\#$ i $\textsc{Rev}(x)\$\textsc{Rev}(y)\#$. Przygotowujemy drzewa do zapytań LCA (Longest common ancestor) — preprocessing $O(n)$, zapytania $O(1)$. Dla każdej pozycji i w y możemy w czasie $O(1)$ wyznaczyć $l' = LCP(x, y[i, \ldots])$ (longest common prefix) oraz $l'' = LCS(x, y[1, \ldots, i + k - 1])$ (longest common suffix). Jeśli $l' + l'' = k - 1$ to $y[i, \ldots, i + k - 1]$ różni się od x na dokładnie jednej pozycji.

Zadanie 12.3

Pokaż w jaki sposób z pomocą tablic sufiksowych można wydajnie rozwiązać następujące problemy:

1. znajdź wszystkie wystąpienia słowa q w słowie x
2. znajdź najdłuższe słowo, które pojawia się w x co najmniej 2 razy
3. znajdź najdłuższe wspólne podsłowo słów q i x

Rozwiązanie:
Wszystkie wystąpienia słowa q w słowie x (off-line)
Budujemy tablicę sufiksową (SA) i LCP dla tekstu $T = x\$q\#$. Niech i to pozycja w tablicy sufiksowej, która odpowiada sufiksowi $q\#$ (czyli $SA[i] = |x| + 2$). Odpowiedzią jest suma zbiorów $OCC_1 \cup OCC_2$:

$$OCC_1 = \{SA[j] : j < i \text{ oraz } \min(LCP[j + 1, \ldots, i]) = |q|\}$$

$$OCC_2 = \{SA[j] : j > i \text{ oraz } \min(LCP[i + 1, \ldots, j]) = |q|\}$$

Znajdź najdłuższe słowo, które pojawia się w x co najmniej 2 razy
Budujemy tablicę sufiksową (SA) i LCP dla tekstu $T = x\$$. Długość najdłuższego słowa, które występuje co najmniej dwa razy to $\max(LCP)$.

Znajdź najdłuższe wspólne podsłowo słów q i x
Budujemy tablicę sufiksową (SA) i LCP dla tekstu $T = q\$x\#$. Niech

$$id(i) = \begin{cases} 1 & 1 \leq i \leq |q| \\ 2 & |q| + 2 \leq i \leq |q| + 2 + |x| \\ NULL & \text{wpp} \end{cases}$$

Długość najdłuższgo wspólnego słowa q i x jest wyznaczona przez

$$\max\{LCP[i] : 2 \leq i \leq |T|, id(SA[i - 1]) \not= id(SA[i])\}$$

Zadanie 12.4

Powiemy, że dwa ciągi liczb wymiernych $a_1, a_2, \ldots, a_n$ i $b_1, b_2, \ldots, b_n$ są podobne, gdy istnieje takie c, że $b_i = c \cdot a_i$, dla każdego $i = 1, 2, \ldots, n$. Zaprojektuj wydajny algorytm, który dla danych dwóch ciągów dodatnich liczb wymiernych $x_1, x_2, \ldots, x_n$ oraz $y_1, y_2, \ldots, y_m$, $n \leq m$, wyznacza w ciągu y wszystkie pozycje i takie, że podciąg $y_i, y_{i+1}, \ldots, y_{i+n-1}$ oraz ciąg x są podobne. Możesz przyjąć, że operacje arytmetyczne na liczbach wymiernych wykonywane są w czasie stałym.

Rozwiązanie: Jeśli $n = 1$ to jedno-elementowy ciąg X występuje na każdej pozycji w Y.
Jeśli $n > 1$ i elementy ciągów są różne od 0, to obliczamy nowe ciągi:

$$X' = \left[\frac{x_2}{x_1}, \frac{x_3}{x_2}, \ldots, \frac{x_n}{x_{n-1}}\right]$$

$$Y' = \left[\frac{y_2}{y_1}, \frac{y_3}{y_2}, \ldots, \frac{y_m}{y_{m-1}}\right]$$

Za pomocą KMP znajdujemy wystąpienia X' w Y'.

Zadanie 12.5

W tym zadaniu rozważamy słowa zbudowane z cyfr $0, 1, \ldots, 9$. Każde takie słowo można traktować jako zapis w układzie dziesiętnym pewnej nieujemnej liczby całkowitej. Zaprojektuj efektywny algorytm, który dla danego niepustego słowa x:

a) obliczy liczbę wszystkich takich par indeksów (i, j), $1 \leq i \leq j \leq |x|$, że słowo $x[i..j]$ jest zapisem liczby podzielnej przez 3;

b) wyznaczy taką parę indeksów (i, j), $1 \leq i \leq j \leq |x|$, że słowo $x[i..j]$ jest zapisem największej liczby podzielnej przez 3 wśród wszystkich podsłów słowa x;

c) obliczy liczbę wszystkich parami różnych podsłów (różniących się długością lub znakami na odpowiadających sobie pozycjach) słowa x będących zapisami liczb podzielnych przez 3.

Rozwiązanie: Dla ustalenia uwagi, niech S oznacza zadane słowo $c_1, \ldots, c_n$ ($c_i \in \{0, \ldots, 9\}$).

a) Obliczamy tablicę $Ile(i, j)$ dla $0 \leq i \leq n$, $j \in \{0, 1, 2\}$ oznaczającą liczbę prefiksów $S[1, \ldots, i]$ o sumie $\equiv j$ (mod 3). Następnie obliczamy tablicę $Wynik(i)$ dla $0 \leq i \leq n$ oznaczającą liczbę podsłów $S[1, \ldots, i]$ o sumie podzielnej przez 3.

$$Wynik(i) = Wynik(i-1) + Ile(i-1, Suma(i))$$

b) Dla każdej pozycji i zaznaczamy najdłuższe słowo podzielne przez 3 kończące się na danej pozycji. Następnie wybieramy tylko słowa o maksymalnej długości. Za pomocą drzewa sufiksowego wyznaczamy największe leksykograficznie słowo.

c)

Zadanie 12.6

Niech x będzie słowem binarnym o długości co najmniej 2 i zawierającym co najmniej jedno `0` (zero) oraz co najmniej jedną `1` (jedynkę). Zaprojektuj efektywny algorytm, który w słowie binarnym x znajduje dwa podsłowa o maksymalnej długości, które różnią na każdej pozycji.

Rozwiązanie: Obliczamy drzewo sufiksowe dla $S' = x\$\textsc{Neg}(x)\#$. Dla takiego słowa szukamy najdłuższego słowa, które występuje w S' co najmniej dwa razy, raz w x raz w $\textsc{Neg}(x)$. Złożoność czasowa i pamięciowa: $O(n)$.

Zadanie 12.7

Zaprojektuj efektywny algorytm, który dla danych słów x, y nad alfabetem $\{d, i, k, s\}$ obliczy ile różnych słów będących cyklicznymi przesunięciami słowa x jest podsłowami słowa y.

Rozwiązanie: Budujemy drzewo sufiksowe dla słowa $S = y\$xx\#$. Znajdujemy zbiór węzłów drzewa V na głębokości $|x|$ (zbiór powinien zawierać również węzły, które znajdują się wewnątrz skompresowanych krawędzi drzewa). Dla każdego wierzchołka $v \in V$ sprawdzamy, czy w jego poddrzewie istnieje sufiks rozpoczynający się we fragmencie xx.

Zadanie 12.8

Zaproponuj efektywny algorytm, który w słowie S o długości $n \geq 3$ nad alfabetem $\{a, b\}$ znajdzie dwa najdłuższe, takie same podsłowa nie zachodzące na siebie

Przykład
W słowie *aaaaaa* takie dwa najdłuższe podsłowa to *aaa* zaczynające się na pozycji 1 i *aaa* zaczynające się na pozycji 4.
W słowie *abbabbbab* takie słowa, to np. *bbab* zaczynające się na pozycji 2 i *bbab* zaczynające się na pozycji 6.

Rozwiązanie: Budujemy drzewo sufiksowe S, oraz w każdym węźle u zapisujemy:

- liczbę liści w poddrzewie,
- głębokość (suma wag krawędzi na ścieżce $r \to u$),
- minimalny i maksymalny indeks liścia w poddrzewie.