Union-Find

1 Definicja problemu

Dany jest skończony zbiór U oraz ciąg σ instrukcji UNION i FIND:

- UNION(A, B, C); gdzie A, B - rozłączne podzbiory U;
  wynikiem instrukcji jest utworzenie zbioru C takiego, że $C \leftarrow A \cup B$, oraz usunięcie zbiorów A i B;
- FIND(i); gdzie $i \in U$;
  wynikiem instrukcji jest nazwa podzbioru, do którego aktualnie należy i.

Problem polega na zaprojektowaniu struktury danych umożliwiającej szybkie wykonywanie ciągów σ. Początkowo każdy element U tworzy jednoelementowy podzbiór.

- Często nazwy podzbiorów są nieistotne, a instrukcja FIND służy jedynie do stwierdzenia czy dane elementy należą do tego samego podzbioru.

Przykładowe rozwiązanie konstrukcja minimalnego drzewa rozpinającego grafu

$T \leftarrow \emptyset$
$VS \leftarrow \emptyset$
for each $v \in V$ **do** wstaw zbiór $\{v\}$ do VS
while $|VS| > 1$ **do**
  wybierz $\langle u, w \rangle$ z E o najmniejszym koszcie
  usuń $\langle u, w \rangle$ z E
  $A \leftarrow FIND(u)$; $B \leftarrow FIND(w)$
  if $A \neq B$ **then** $UNION(A, B, X)$
    wstaw $\langle u, w \rangle$ do T

ładny dowód czasu zamortyzowanego u Pawła na githubie!!!

3.1 Proste rozwiązanie

Do reprezentowania rodziny zbiorów używamy tablicy $R[1..n]$ takiej, że

$\forall_i$ $R[i]$ jest nazwą zbioru zawierającego i.

Koszt: FIND - $\Theta(1)$; UNION -$\Theta(n^2)$.

3.2 Modyfikacja prostego rozwiązania

3.2.1 Idea

Oparta na dwóch trickach:

- Wprowadzamy nazwy wewnętrzne zbiorów (niewidoczne dla użytkownika).
- Podczas wykonywania UNION(A, B, C) zbiór mniejszy przyłączany jest do większego.

3.2.2 Realizacja

Używamy tablic: $R, ExtName, IntName, List, Next$ i $Size$ takich, że:

$R[i]$	=	nazwa wewnętrzna zbioru zawierającego i,
$ExtName[j]$	=	nazwa zewnętrzna zbioru o nazwie wewnętrznej j,
$IntName[k]$	=	nazwa wewnętrzna zbioru o nazwie zewnętrznej j,
$List[j]$	=	wskaźnik na pierwszy element w liście elementów zbioru o nazwie wewnętrznej j,
$Next[i]$	=	następny po i element w liście elementów zbioru $R[i]$,
$Size[j]$	=	liczba elementów w zbiorze o nazwie wewnętrznej j.

procedure $Find(i)$
  return $(ExtName(R[i]))$

procedure $UNION(I, J, K)$
  $A \leftarrow IntName[I]$
  $B \leftarrow IntName[J]$
  Niech $Size[A] \leq Size[B]$; w p.p. zamień A i B rolami
  $el \leftarrow List[A]$
  while $el \neq 0$ **do** $R[el] \leftarrow B$
    $last \leftarrow el$
    $el \leftarrow Next[el]$
  $Next[last] \leftarrow List[B]$
  $List[B] \leftarrow List[A]$
  $Size[B] \leftarrow Size[A] + Size[B]$
  $IntName[K] \leftarrow B$
  $ExtName[B] \leftarrow K$

Twierdzenie 1 *Używając powyższego algorytmu można wykonać dowolny ciąg σ o długości $O(n)$ w czasie $O(n \log n)$.*

4 Struktury drzewiaste dla problemu Union-Find

4.1 Elementy składowe struktury danych

- Las drzew.
  Każdy podzbiór reprezentowany jest przez drzewo z wyróżnionym korzeniem. Wierzchołki wewnętrzne zawierają wskaźnik na ojca (nie ma wskaźników na dzieci!).
- Tablica $Element[1..n]$:

$$Element[i] = \text{ wskaźnik na wierzchołek zawierający } i.$$

- Tablica $Root$:

$$Root[I] = \text{ wskaźnik na korzeń drzewa odpowiadającego zbiorowi } I$$

(nazwy zbiorów są dla nas nieistotne; będą one liczbami z $[1, .., n]$).

4.2 Realizacja instrukcji

$Union(A, B, C)$ polega na połączeniu drzew odpowiadających zbiorom A i B w jedno drzewo i umieszczeniu w jego korzeniu nazwy C.

$Find(i)$ polega na przejściu ścieżki od wierzchołka wskazywanego przez $Element(i)$ do korzenia drzewa i odczytaniu pamiętanej tam nazwy drzewa.

Przy wykonywaniu tych instrukcji stosujemy następującą strategię:

1. instrukcję $Union$ wykonujemy w sposób zbalansowany - korzeń mniejszego (w sensie liczby wierzchołków) drzewa podwieszamy do korzenia drzewa większego (a dokładniej drzewa nie większego do korzenia drzewa nie mniejszego),
2. podczas instrukcji $Find(i)$ wykonujemy *kompresję ścieżki* prowadzącej od i do korzenia - wszystkie wierzchołki leżące na tej ścieżce podwieszamy bezpośrednio pod korzeń.

4.3 Implementacja

Każdy wierzchołek v zawiera pola:

- $Father[v]$ - wskaźnik na ojca (równy NIL, gdy v jest korzeniem),
- $Size[v]$ - liczba wierzchołków w drzewie o korzeniu v,
- $Name[v]$ - nazwa drzewa o korzeniu v

Zawartość pól $Size[v]$ i $Name[v]$ ma znaczenie tylko wówczas, gdy v jest korzeniem.

```
procedure InitForest
    for i ← 1 to n do v ← Allocate − Node()
                       Size[v] ← 1
                       Name[v] ← i
                       Father[v] ← NIL
                       Element[i] ← v
                       Root[i] ← v
```

```
procedure Union(i, j, k)
    Niech Size[Root[i]] ≤ Size[Root[j]]; w p.p. zamień i oraz j rolami
    large ← Root[j]
    small ← Root[i]
    Father[small] ← large
    Size[large] ← Size[large] + Size[small]
    Name[large] ← k
    Root[k] ← large
```

```
procedure Find(i)
    list ← NIL
    v ← Element[i]
    while Father[v] ≠ NIL do wstaw v na list
                          v ← Father[v]
    for each w ∈ list do Father[w] ← v
    return Name[v]
```

4.4.1 Górne ograniczenie

Twierdzenie 2 *Niech c będzie dowolną stałą. Wówczas istnieje inna stała c' (zależna od c) taka, że powyższe procedury wykonują dowolny ciąg σ złożony z cn instrukcji Union i Find w czasie $c'n \log^* n$.*

IDEA DOWODU: Instrukcje *Union* wykonują się w czasie stałym. Wystarczy więc oszacować koszt instrukcji *Find*.

Koszt każdej instrukcji $Find(v)$ jest proporcjonalny do liczby wierzchołków na ścieżce od v do korzenia. Obarczymy tym kosztem niektóre z odwiedzanych wierzchołków jak i samą instrukcję $Find(v)$. Stosujemy przy tym następującą strategię:

- za odwiedzenie wierzchołka w jednostkowym kosztem obarczamy instrukcję $Find(v)$, jeśli:
  - w jest korzeniem drzewa lub
  - w jest synem korzenia drzewa lub
  - w i jego ojciec mają rzędy w innych grupach.
- w pozostałych przypadkach jednostkowym kosztem obarczamy odwiedzany wierzchołek.

Tezę otrzymujemy na podstawie dwóch spostrzeżeń:

- Ponieważ grup rzędów jest nie więcej niż $\log^* n$, każda instrukcja *Find* zostanie obciążona kosztem nie większym niż $\log^* n + 1$.
- Pokazujemy dla każdej grupy rzędów, że sumaryczne obciążenie wszystkich wierzchołków, których rzędy należą do niej, jest $O(n)$.

4.4.2 Dolne ograniczenie

Otrzymane ograniczenie jest bliskie liniowemu, ale nie liniowe. Powstaje więc naturalne pytanie, czy tego ograniczenia nie można poprawić. Okazuje się, że można. Funkcja $\log^* n$ może zostać zastąpiona przez odwrotną funkcję Ackermanna, która rośnie jeszcze wolniej niż $\log^* n$. Kolejne twierdzenie pokazuje jednak, że zaprezentowana strutura drzewiasta nie osiąga złożoności liniowej. Nie wiadomo, czy istnieją struktury danych pozwalające na osiągnięcie czasu liniowego.

Twierdzenie 3 *Algorytm realizujący ciągi instrukcji Union i Find przy użyciu powyższych procedur ma złożoność większą niż cn dla dowolnej stałej c.*

4.4 Analiza algorytmu

Lemat 1 *Jeśli instrukcje Union wykonujemy w sposób zbalansowany, to każde powstające drzewo o wysokości h ma co najmniej 2^h wierzchołków.*

Definicja 1 *Niech $\tilde{\sigma}$ będzie ciągiem instrukcji Union powstałym po usunięciu wszystkich instrukcji Find z ciągu σ. Rzędem wierzchołka v względem σ nazywamy jego wysokość w lesie powstałym po wykonaniu ciągu $\tilde{\sigma}$.*

Lemat 2 *Jest co najwyżej $\frac{n}{2^r}$ wierzchołków rzędu r.*

Wniosek 1 *Każdy wierzchołek ma rząd co najwyżej $\log n$.*

Lemat 3 *Jeśli w trakcie wykonywania ciągu σ wierzchołek w staje się potomkiem wierzchołka v, to rząd w jest mniejszy niż rząd v.*

Definicja 2

$$\log^*(n) \stackrel{df}{=} \min\{k \mid F(k) \geq n\},$$

gdzie $F(0) = 1$ i $F(i) = 2^{F(i-1)}$ dla $i > 0$.

Rzędy wierzchołków dzielimy na *grupy*. Rząd r umieszczamy w grupie $\log^* r$.

Jaki byłby koszt wykonania ciągu δ złożonego z $O(n)$ operacji $UNION$ i $FIND$, gdyby w operacji $UNION$ zbiory były łączone w dowolny (niekoniecznie zrównoważony sposób), a operacja $FIND$ nadal byłaby wykonywana z kompresją ścieżek?

Przy zastosowaniu tylko kompresji ścieżki, koszt uniona pozostaje $O(1)$, a amortyzowany koszt finda to $O(\log n)$. W najgorszym przypadku, koszt, to pewnie $O(n \log n)$.

Podaj definicję pojęć: rząd wierzchołka, grupa rzędu.

▼ Rozwiązanie

Definicja 18 *Niech $\tilde{\sigma}$ będzie ciągiem instrukcji Union powstałym po usunięciu wszystkich instrukcji Find z ciągu σ. Rzędem wierzchołka v względem σ nazywamy jego wysokość w lesie powstałym po wykonaniu ciągu $\tilde{\sigma}$.*

grupa rzędu - Dla rzędu r, jego grupa $g = log^*(r)$

W jakim czasie można wykonać ciąg n operacji **union** i **find**, w którym wszystkie operacje **union** poprzedzają operacje **find**? Odpowiedź uzasadnij.

▼ Rozwiązanie

Ciąg n operacji **Union** wykonujemy każdą operację w czasie stałym. Następnie operacje **Find** wykonujemy w czasie stałym zamortyzowanym dzięki kompresji ścieżek po których idziemy z wierzchołka do korzenia.

Łączna suma czasu wykonywania `n` operacji jest równa `cn` gdzie `c` to jakaś mała stała.

W analizie problemu Union Find wykorzystywaliśmy pojęcia rzędu wierzchołka oraz grupy rzędu.

Przypomnij definicje tych pojęć.

Ile maksymalnie bitów potrzebujemy przeznaczyć na pamiętanie rzędu w każdym wierzchołku?

Podaj jaki jest pesymistyczny czas wykonywania operacji find, gdy operacja union wykonywana jest w sposób:

(a) zbalansowany,

(b) niezbalansowany.

analizie problemu Union-find wykorzystaliśmy pojęcie rzędu wierzchołka oraz grupy rzęc

maksymalnie rzędów może należeć do tej samej grupy, gdy liczba elementów jest równa

aj sensowne ograniczenie górne na największą wysokość drzewa, które może powstać w wyniku wykonania w sposób zbalansowar racji *union* na rozłącznych zbiorach (początkowo zbiory te są jednoelementowe).

wodnij swoje stwierdzenie.

---

ozwiązanie

gn)

1. (1pkt) Niech σ będzie ciągiem instrukcji *Union* i *Find*, w którym wszystkie instrukcje *Union* występują przed instrukcjami *Find*. Udowodnij, że algorytm oparty na strukturach drzewiastych wykonuje σ w czasie proporcjonalnym do długości σ.

2. (2pkt) Rozważamy ciągi operacji $Insert(i)$, $DeleteMin$ oraz $Min(i)$ wykonywanych na S - podzbiorze zbioru $\{1, \ldots, n\}$. Obliczenia rozpoczynamy z $S = \emptyset$. Instrukcja $Insert(i)$ wstawia liczbę i do S. Instrukcja $DeleteMin$ wyznacza najmniejszy element w S i usuwa go z S. Natomiast wykonanie $Min(i)$ polega na usunięciu z S wszystkich liczb mniejszych od i.

   Niech σ będzie ciągiem instrukcji $Insert(i)$, $DeleteMin$ oraz $Min(i)$ takim, że dla każdego i, $1 \leq i \leq n$, instrukcja $Insert(i)$ występuje co najwyżej jeden raz. Mając dany ciąg σ naszym zadaniem jest znaleźć ciąg liczb usuwanych kolejno przez instrukcje $DeleteMin$. Podaj algorytm rozwiązujący to zadanie.
   UWAGA: Zakładamy, że cały ciąg σ jest znany na początku, czyli interesuje nas wykonanie go *off-line*.
   WSKAZóWKA: Rozdział 4.8 z książki Aho,... .

3. (2pkt) Rozważamy ciągi instrukcji: $Link(r, v)$ oraz $Depth(v)$ wykonywanych na lesie rozłącznych drzew o wierzchołkach z etykietami ze zbioru $\{0, ..., n-1\}$ (różne wierzchołki mają różne etykiety). Operacja $Link(r, v)$ czyni r, korzeń jednego z drzew, synem v, wierzchołka innego drzewa. $Depth(v)$ oblicza głębokość wierzchołka v.

   Naszym celem jest napisanie algorytmu, który dla danego ciągu σ wypisze w sposób on-line wyniki instrukcji $Depth$ (tzn. wynik każdej instrukcji $Depth$ ma być obliczony przed wczytaniem kolejnej instrukcji z ciągu σ). Pokaż jak zastosować drzewiastą strukturę danych dla problemu $Union - Find$ do rozwiązania tego problemu.
   WSKAZóWKA: Rozdział 4.8 z książki Aho,... .

4. (1pkt) Rozważ taką wersję wykonywania kompresji ścieżek, w której wierzchołki wizytowane podczas wykonywania operacji *Find* podwieszane są pod własnego dziadka. Czy analiza złożoności przeprowadzona na wykładzie da się zastosować w tym przypadku?

5. (1,5pkt) Dany jest graf $G = (V, E)$, wyróżniony wierzchołek v oraz ciąg jego krawędzi $e_i = \{v_i, u_i\}$, $(i = 1, 2, \ldots, m)$. Ułóż algorytm, który dla każdego wierzchołka u wyznaczy minimalną wartość j, taką że po usunięciu z grafu G krawędzi $e_1, \ldots, e_j$, nie istnieje ścieżka łącząca wierzchołki u i v.

```
Teza.
  Drzewo budowane przez operację UNION o wysokości n, ma co najmniej 2^n wierzchołków.

D-d. przez indukcję po n.

1. Podstawa
  n = 0, zachodzi
  n = 1, zachodzi

2. Załóżmy, że zachodzi dla n
3. Pokażę, że zachodzi dla n+1
  Aby stworzyć drzewo o wysokości n+1 musimy połączyć dwa drzewo o wysokości co
  najmniej n(bo inaczej nie zwiększyłaby się wysokość) i o wysokości nie większej
  niż n(bo gdyby jedno drzewo miało wysokość większą niż n, to drugie mniejsze
  drzewo zostałoby podpięte pod to drzewo - ponieważ mam zbalansowany UNION).
  Więc łącząc dwa drzewa o wysokości n i korzystając z założenia dostajemy 2^n + 2^n = 2^n+1
  Stąd zachodzi teza indukcyjna.
<p>
```