

UNION-FIND

IIUWr. II rok informatyki.

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.

1.1 Uwagi i założenia

- Zbiór U jest mały ($|U| \ll$ pojemność pamięci wewnętrznej). Zwykle przyjmuje się, że $U = \{1, \dots, n\}$.
- Rozważa się dwa sposoby wykonywania ciągów σ :
 - *on-line* - wynik każdej instrukcji musi zostać obliczony przed wczytaniem kolejnej instrukcji;
 - *off-line* - ciąg σ może być wczytany całkowicie zanim zostanie obliczony wynik którejkolwiek instrukcji.

Nas interesować będzie sposób *on-line*.

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

2 Przykład zastosowania

2.1 Konstrukcja minimalnego drzewa rozpinającego grafu

```
T ← ∅
VS ← ∅
for each v ∈ V do wstaw zbiór {v} do VS
while |VS| > 1 do
    wybierz ⟨u, w⟩ z E o najmniejszym koszcie
    usuń ⟨u, w⟩ z E
    A ← FIND(u); B ← FIND(w)
    if A ≠ B then UNION(A, B, X)
                    wstaw ⟨u, w⟩ do T
```

3 Rozwiązania

3.1 Proste rozwiązanie

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

$$\forall_i \quad R[i] \text{ jest nazwą zbioru zawierającego } i.$$

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 $\text{UNION}(A, B, C)$ zbiór mniejszy przyłączany jest do większego.

3.2.2 Realizacja

Używamy tablic: $R, \text{ExtName}, \text{IntName}, \text{List}, \text{Next}$ i Size takich, że:

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

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

procedure UNION(I, J, K)
    A ← IntName[I]
    B ← IntName[J]
    Niech Size[A] ≤ Size[B]; w p.p. zamień A i B rolami
    el ← List[A]
    while el ≠ 0 do
        R[el] ← B
        last ← el
        el ← Next[el]
    Next[last] ← List[B]
    List[B] ← List[A]
    Size[B] ← Size[A] + Size[B]
    IntName[K] ← B
    ExtName[B] ← 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 $Element[1..n]$:

$Element[i] =$ wskaźnik na wierzchołek zawierający i .

- Tablica $Root$:

$Root[I] =$ wskaźnik na korzeń drzewa odpowiadającego zbiorowi I

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

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  $i \leftarrow 1$  to  $n$  do  $v \leftarrow Allocate - Node()$ 
     $Size[v] \leftarrow 1$ 
     $Name[v] \leftarrow i$ 
     $Father[v] \leftarrow NIL$ 
     $Element[i] \leftarrow v$ 
     $Root[i] \leftarrow v$ 
```

```

procedure Union(i, j, k)
  Niech  $Size[Root[i]] \leq Size[Root[j]]$ ; w p.p. zamień i oraz j rolami
  large  $\leftarrow Root[j]$ 
  small  $\leftarrow Root[i]$ 
  Father[small]  $\leftarrow large$ 
  Size[large]  $\leftarrow Size[large] + Size[small]$ 
  Name[large]  $\leftarrow k$ 
  Root[k]  $\leftarrow large$ 

```

```

procedure Find(i)
  list  $\leftarrow NIL$ 
  v  $\leftarrow Element[i]$ 
  while Father[v]  $\neq NIL$  do wstaw v na list
    v  $\leftarrow Father[v]$ 
  for each w  $\in list$  do Father[w]  $\leftarrow v$ 
  return Name[v]

```

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 $\log^* r$.

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. Obarczmy 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 struktura 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 .*

Dowód tego twierdzenia, mimo, że nie jest trudny, wykracza poza zakres naszego przedmiotu. Można go znaleźć w [1].

UWAGA: na ćwiczeniach pokażemy, że przy pomocy struktur drzewiastych można w czasie $O(n \log^* n)$ realizować ciągi σ , które oprócz instrukcji $Union$ i $Find$ zawierają także instrukcje $Insert$ i $Delete$.

Literatura

- [1] A.V. Aho, J.E. Hopcroft i J.D. Ullman, *Projektowanie i Analiza Algorytmów Komputerowych*, PWN, 1983 (oraz Helion 2003).