Disjoint-Set Operations with Examples

Disjoint Set Operations

We have a collection of disjoint sets of elements. Each set is identified by a representative element. We want to perform union operations, and tell which set something is in. This is useful in a minimum spanning tree algorithm and many other applications. Formally, we have the following operations.

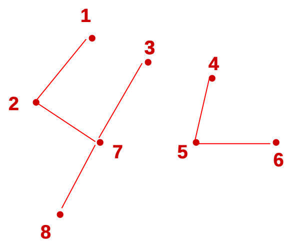
- MAKE-SET(x) : Create new set {x} with representative x.
- UNION(x,y) : x and y are elements of two sets. Remove these sets and add their union. Choose a representative for it.
- FIND-SET(x) : return the representative of the set containing x.

Example

MAKE-SET(1)	{1}	
MAKE-SET(2)	{2}	
MAKE-SET(3)	{3}	
MAKE-SET(4)	{4}	
FIND(3)	(returns 3)	
FIND(2)	(returns 2)	
UNION(1,2)	(representative 1, say)	{1,2}
FIND(2)	(returns 1)	
FIND(1)	(returns 1)	
UNION(3,4)	(representative 4, say)	{3,4}
FIND(4)	(returns 4)	
FIND(3)	(returns 4)	
UNION(1,3)	(representative 4, say)	{1,2,3,4}
FIND(2)	(returns 4)	
FIND(1)	(returns 4)	
FIND(4)	(returns 4)	
FIND(3)	(returns 4)	

An Application of Disjoint-Set Data Structure

```
Connected-Components(G)
{    for each vertex $v \in V[G]$
         do Make-Set(v)
     for each edge(u,v)$\in E[G]$
         do if Find-Set(u) $\neq$ Find-Set(v)
             then Union(u,v)   }
```



{1} , {2}

{1,2}

Linked List Implementation with Concatenation

Each cell has an element, a pointer to the next member, and a pointer to the first element in the list, which is the set representative.

For union, we append one list to another, changing the pointers to the set representative on the list at the end.

MAKE-SET	O(1) time
FIND-SET	O(1) time
UNION	O(m) time

Thus we get quadratic worst case performance.

Make-Set(x_1),......, Make-Set(x_n)

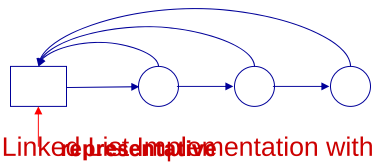
$\overset{1}{\text{Union}(x_1,x_2)}$, $\overset{2}{\text{Union}(x_2,x_3)}$, $\overset{3}{\text{Union}(x_3,x_4)}$ ......, $\overset{q-1}{\text{Union}(x_{q-1},x_q)}$

$$n = \lceil m/2 \rceil + 1, q = m - n$$

$$\sum_{i=1}^{q-1} i = O(q^2) = O(m^2)$$

Disjoint-Set Union

Use a linked list to represent a set



Linked List Implementation with Concatenation：

- Make-Set：O(1)
- Find-Set：O(1)
- Union(A,B)：Copy elements of A into B, O(A).

Worst-Case Analysis

$|S_i| = 1$ initially.

Union(S_1,S_2)	1
Union(S_2,S_3)	2
⋮	
Union(S_{n-1},S_n)	n-1
	$\Theta(n^2)$

A single Union can still take $\Theta(n)$, but n unions take only $O(n \log n)$ time.

- Improvement：Copy smaller set into larger.

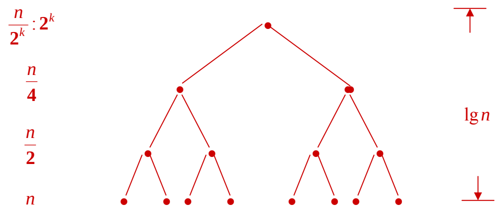
Refined Linked List Implementation

(Concatenate longer list onto shorter one)

MAKE-SET	O(1) time
FIND-SET	O(1) time
UNION	O(log n) time (rough bound)

Proof of the bound for UNION :

A set has to double in size each time it is concatenated onto the end.
Thus we get O(n log n) worst case performance for n operations.



Amortized Analysis

- In a set of size n, any element can be copied at most lgn times.

  Each time copied, it was in smaller set.
- n elements each copied O(lgn) times ➔ n Unions take O(nlgn) time.
- Each UNION takes amortized O(lgn).

- Forest representation :

Here we represent each set as a tree, and the representative is the root . For example, the following forest represents the set {1,2,3}, {4,5}, {6} :

```
        2         5     6
      /   \       |
     1     3      4
```

Implementation

MAKE-SET(x)	Create a tree
FIND-SET(x)	Go to the root
UNION(x,y)	Add a pointer

Thus we would get the following form UNION(1,4)

```
        2         6
      / | \
     1  3  5
           |
           4
```

This representation does not improve the running time in the worst case over the linked list representation.

Path Compression and Ranks

These are refinements of the forest representation which make it significantly faster.

FIND-SET:	Do path compression
UNION:	Use ranks

"Path compression" means that when we do FIND-SET(X), we make all nodes encountered point directly to the representative element for x. Initially, all elements have rank 0. The ranks of representative elements are updated so that if two sets with representatives of the same rank are unioned, then the new representative is incremented by one.

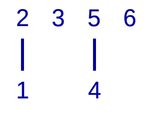
- Example :

We show a sequence of operations and what the forest would look like.

MAKE-SET(1) ... MAKE-SET(6)

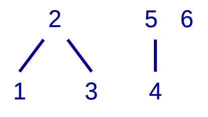
1 2 3 4 5 6 RANKS : 0

UNION(1,2) UNION(4,5)



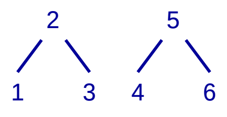
RANK(2)=1
RANK(5)=1

UNION(1,3)



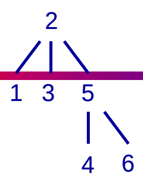
RANK(2)=1
RANK(5)=1

UNION(5,6)



RANK(2)=1
RANK(5)=1

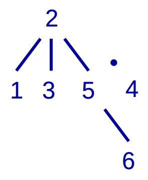
UNION(4,3)



RANK(2)=2

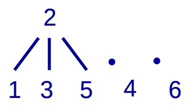
RANK(5)=1

FIND(4)



FIND(3) no change

FIND(6) (path compression)



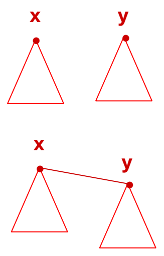
Opertation Algorithms

- Make_Set(x)：
  - Algorithm：

```
Make_Set(x)
    {P(x) 🡐 x
     Rank[x] 🡐 0   }
```

- Union(x,y)：
  - Algorithm：

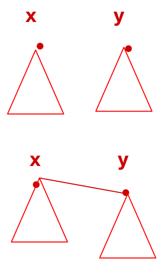
```
Union(x,y)
{     Link(Find-Set(x),Find-Set(y))      }
```



Opertation Algorithms - contd.

- Link(x,y) :
  - Algorithm :

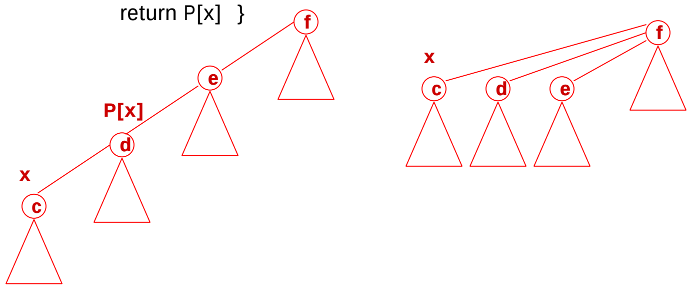
```
Link(x,y)
{     if rank[x] > rank[y]
          then P[y] ← x
          else P[x] ← y
              if rank[x]=rank[y] then rank[y]++ }
```



Opertation Algorithms - contd.

- Find_Set(x) :
  - Algorithm :

```
Find_Set(x)
{  if  x ≠ P[x]
        then P[x] ← Find-Set(P[x])
     return P[x]  }
```



Analysis of MST with Disjoint-Set Union

- Sort： $\theta(E\lg E)=\theta(E\lg V)$
- O(V)：Make-Set's
- O(E)：Find-Set's
- O(V)：Union's
- Above disjoint-set operations together take $O(E\cdot\alpha(E,V))$, using best known algorithm for disjoint-set union.
- Total：$O(E\lg V)$
- m operations on n sets :$O(m\cdot\alpha(m,n))$, where $\alpha(m,n)$ is a functional inverse" of Ackermann's function.

The End