Data Structures for Disjoint Sets

Application:

Connected Components

Minimum Spanning Tree

Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET

Disjoint Sets

- Some applications require maintaining a collection of disjoint sets.

- A Disjoint Set ***S*** is a collection of sets $S_1, \ldots\ldots S_n$

  where $\forall_{i \neq j} S_i \cap S_j = \phi$

- Each set has a representative which is a member of the set (usually the minimum if the elements are comparable)

Disjoint Set Operations

- Make-Set(x) – Creates a new set S_x where x is it's only element (and therefore it is the representative of the set).
  $O(1)$ time.

- Union(x, y) – Replaces S_x, S_y by $S_x \cup S_y$.
  One of the elements of $S_x \cup S_y$ becomes the representative of the new set.
  $O(\log n)$ time.

- Find(x) – Returns the representative of the set containing x
  $O(\log n)$ time.

Analyzing Operations

- We usually analyze a sequence of *m* operations, of which *n* of them are Make_Set operations, and *m* is the total of Make_Set, Find, and Union operations.
- Each union operations decreases the number of sets in the data structure, so there can not be more than *n*-1 Union operations.

Applications

- Equivalence Relations (e.g Connected Components)
- Minimum Spanning Trees

Connected Components

- Given a graph G we first preprocess G to maintain a set of connected components

  CONNECTED_COMPONENTS(G)

- Later a series of queries can be executed to check if two vertexes are part of the same connected component

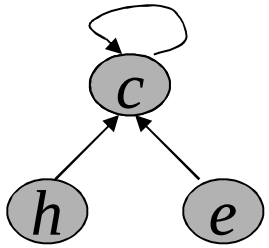
  SAME_COMPONENT(u, v)

Connected Components

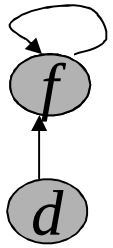
```
CONNECTED_COMPONENTS(G)
    for each vertex v in V[G] do
        MAKE_SET (v)

    for each edge (u, v) in E[G] do
        if FIND_SET(u) != FIND_SET(v) then
            UNION(u, v)


SAME_COMPONENT(u, v)
    if FIND_SET(u) == FIND_SET(v) then
        return TRUE
    else return FALSE
```

Disjoint-Set Implementation: Forests

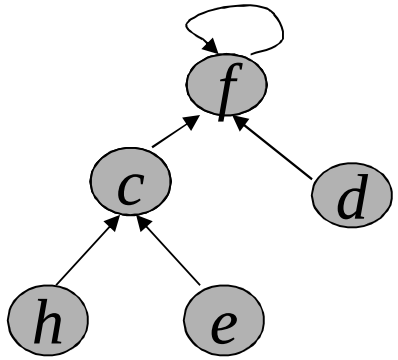
- Rooted trees, each tree is a set, root is the representative. Each node points to its parent. Root points to itself.



Set $\{c, h, e\}$



Set $\{f, d\}$



UNION

Straightforward Solution

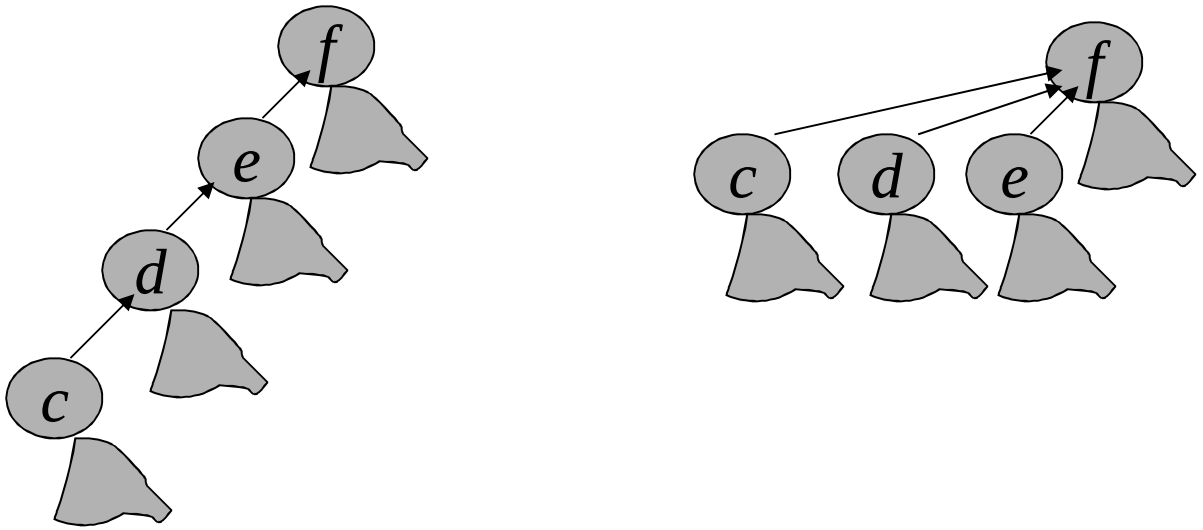
- Three operations
  - MAKE-SET(x): create a tree containing x. $O(1)$
  - FIND-SET(x): follow the chain of parent pointers until to the root. $O(h)$, h is height of x's tree
  - UNION(x, y): let the root of one tree point to the root of the other. $O(1)$
- It is possible that n-1 UNIONs results in a tree of height n-1. (just a linear chain of n nodes).
- So n FIND-SET operations will cost $O(n^2)$.

Union by Rank & Path Compression Heuristics

- Union by Rank: Each node is associated with a rank, which is the upper bound on the height of the node (i.e., the height of subtree rooted at the node), then when UNION, let the root with smaller rank point to the root with larger rank.

- Path Compression: used in FIND-SET(x) operation, make each node in the path from x to the root directly point to the root. Thus reduce the tree height.

Path Compression



Path compression can cause a very deep tree to become very shallow

Algorithm for Disjoint-Set Forest

```
MAKE-SET(x)
1.  p[x]←x
2.  rank[x]←0
```

```
UNION(x, y)
1. LINK(FIND-SET(x), FIND-SET(y))
```

```
LINK(x, y)
1.  if rank[x] > rank[y]
2.  then p[y] ←x
3.  else  p[x] ←y
4.       if rank[x] = rank[y]
5.       then rank[y]++
```

```
FIND-SET(x)
1.  if x≠ p[x]
2.     then p[x] ←FIND-SET(p[x])
3.  return p[x]
```

- Worst case running time for m MAKE-SET, UNION, FIND-SET operations is: $O(m \cdot \alpha(n))$, where $\alpha(n) \leq 4$. So nearly linear in m.
- The find operation does not change: $O(\log n)$

Exercise

- Describe a data structure that supports the following operations:
  - find(x) – returns the representative of x
  - union(x, y) – unifies the groups of x and y
  - min(x) – returns the minimal element in the group of x

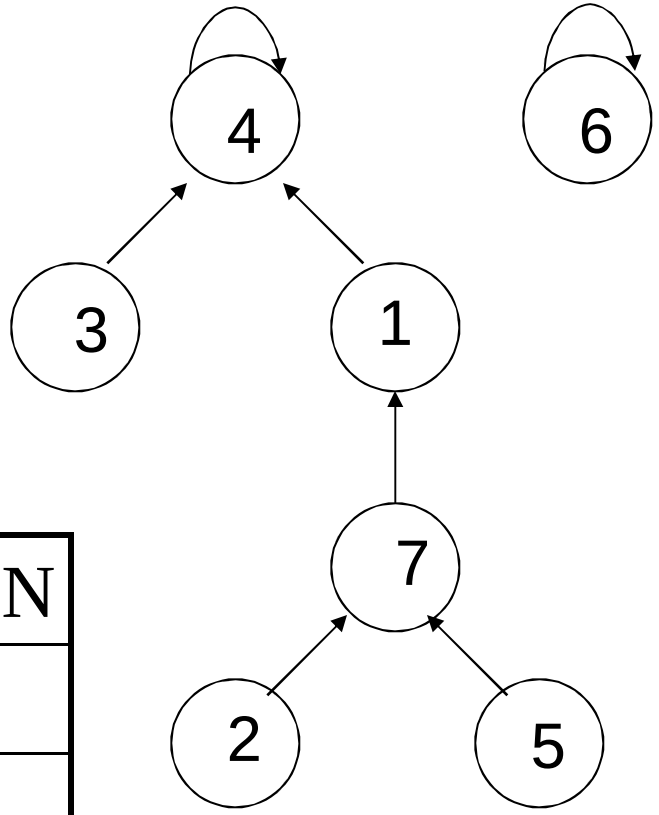
Solution

- We modify the disjoint set data structure so that we keep a reference to the minimal element in the group representative.
- The find operation does not change (log(n))
- The union operation is similar to the original union operation, and the minimal element is the smallest between the minimal of the two groups

Example

- Executing find(5)

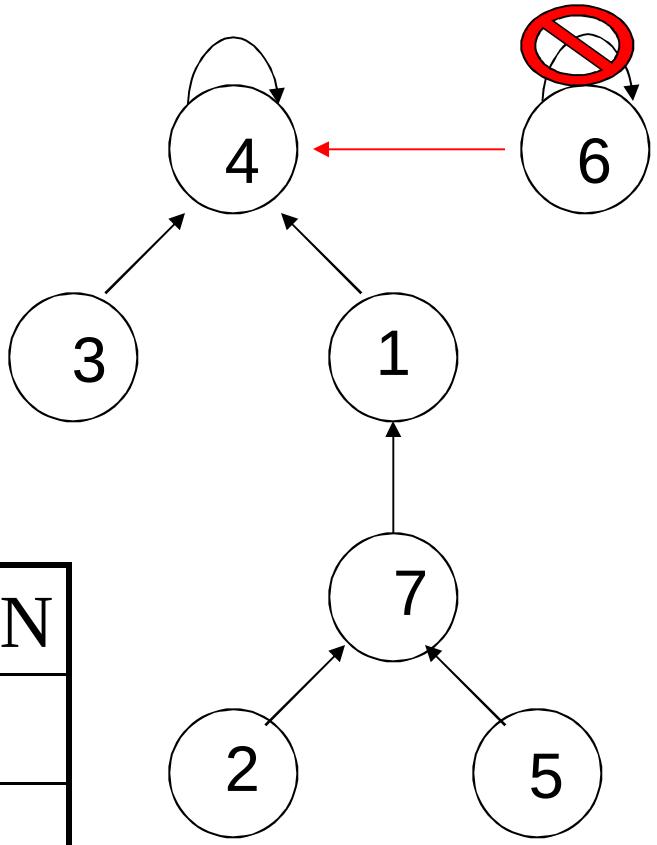
  7 →1→ 4→ 4



	1	2	3	4	5	6	..	N
Parent	4	7	4	4	7	6		
min				1		6		

Example

- Executing union(4,6)



	1	2	3	4	5	6	..	N
Parent	4	7	4	4	7	4		
min				1		1		