# Disjoint Sets

CSE-216

Data Structures and Algorithms

Sessional-II

# Disjoint-Set Data Structure

- Disjoint-Set Data Structure: A data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets.

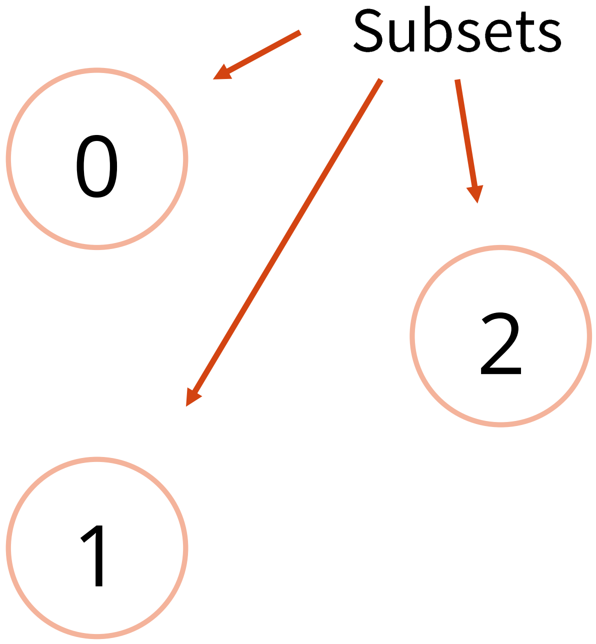
**Supported operations:**

- ➢ **Makeset:** The MakeSet operation makes a new set by creating a new element
- ➢ **Union:** Join two subsets into a single subset.
- ➢ **Find:** Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.

# Disjoint-Set Data Structure

MakeSet operation

```
MakeSet(x)
{
  x.p = x
  x.rank = 0
}
```

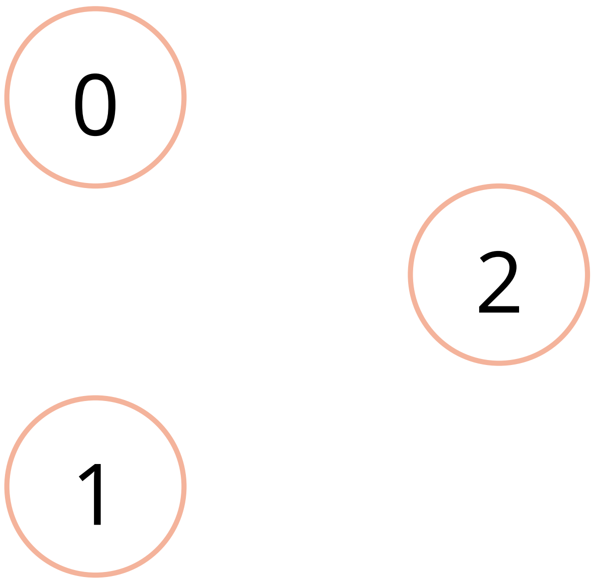



| Node | 0 | 1 | 2 |
|---|---|---|---|
| Parent | 0 | 1 | 2 |

# Disjoint-Set Data Structure

FindSet operation

```
FindSet(x)
{
  if x != x.p
      return FindSet(x.p)
  return x
}
```

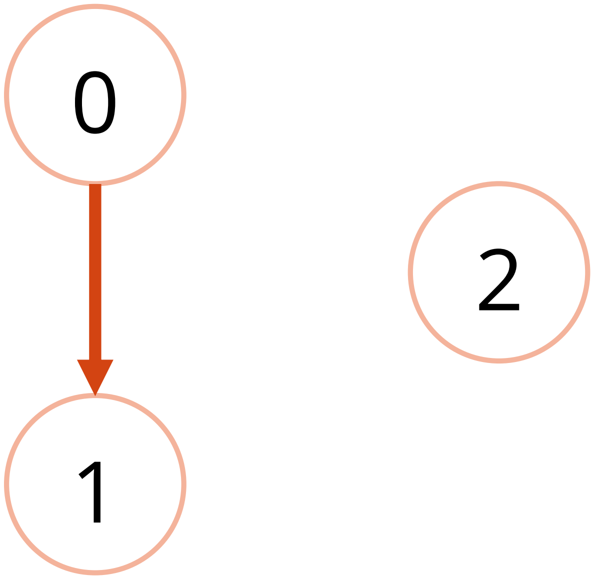



| Node | 0 | 1 | 2 |
|---|---|---|---|
| Parent | 0 | 1 | 2 |

# Disjoint-Set Data Structure

Union operation: {0} U {1}

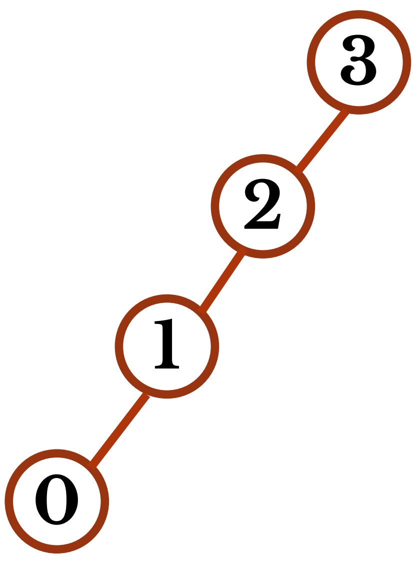
```
Union(u,v)
{
  uRoot = FindSet(u)
  vRoot = FindSet(v)
  vRoot.p = uRoot
}
```




| Node | 0 | 1 | 2 |
|---|---|---|---|
| Parent | 1 | 1 | 2 |

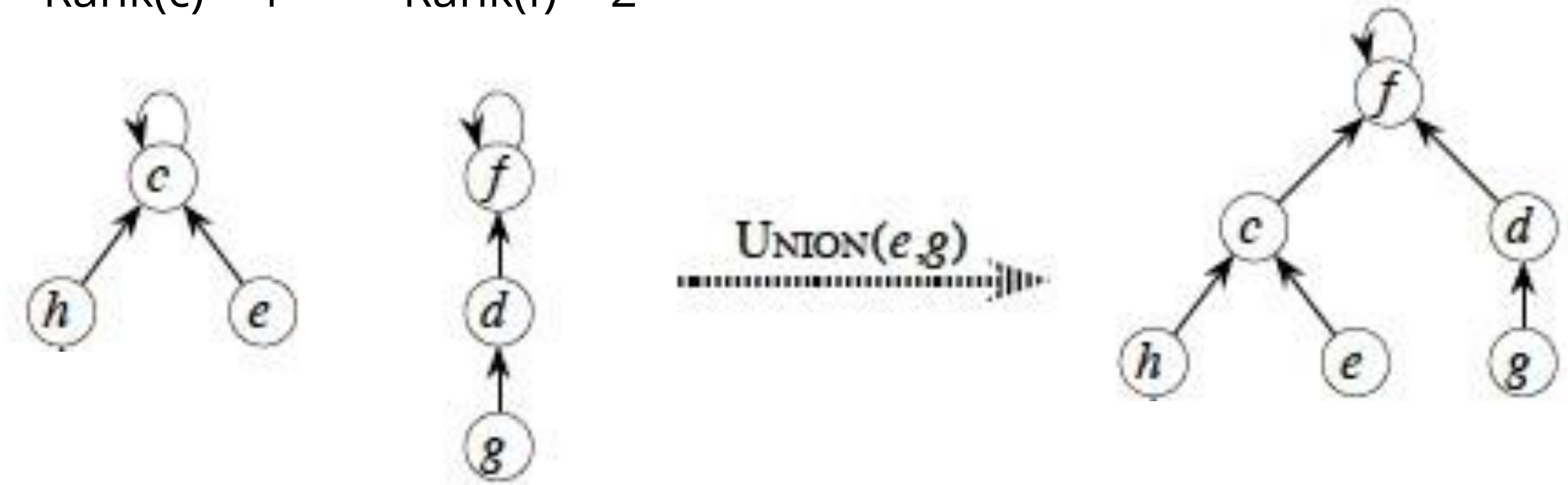
# Union without Optimization

- For, the above mentioned union, worst case time complexity is linear.
- The trees created to represent subsets can become like a linked list.
- Example: 4 elements {0},{1},{2},{3}; Worst case 3 edges {(0,1),(1,2),(2,3)}



# Rank Heuristics

Rank(c) = 1 Rank(f) = 2 Rank(f) = 2



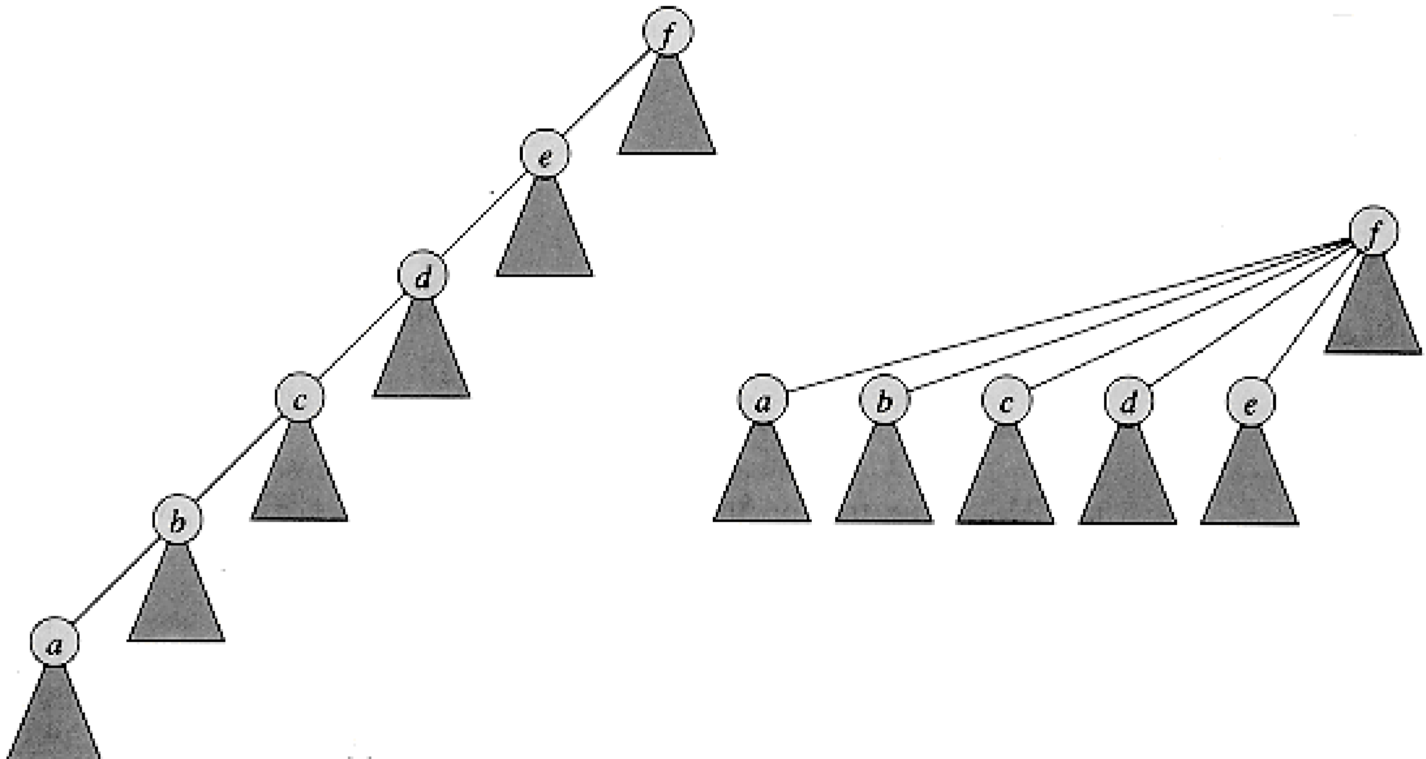

Rank = upper bound of height

# Rank Heuristics

```
Union(u,v)
{
  uRoot = FindSet(u)
  vRoot = FindSet(v)

  if (uRoot.rank > vRoot.rank)
      vRoot.p = uRoot
  else
      uRoot.p = vRoot
      if (uRoot.rank == vRoot.rank)
            vRoot.rank++
}
```

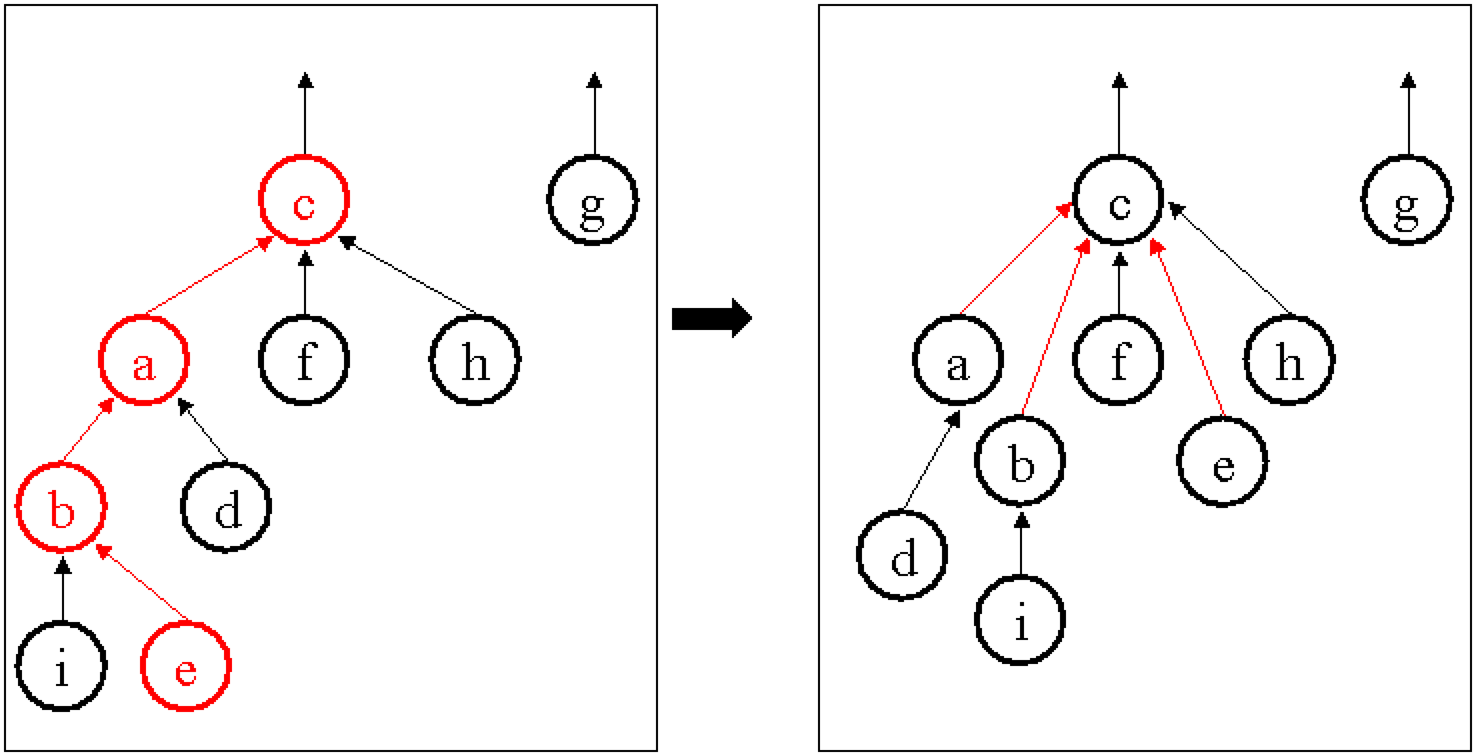
# Path Compression




(a) (b)

# Path Compression

find(e)



# Path Compression

```
FindSet(x)
{
  if x != x.p
      //return FindSet(x.p)
      x.p = FindSet(x.p)
  return x.p
}
```

# Path Compression (Optimized)

- When find() is called for an element x, root of the tree is returned.
- Path Compression: make the found root as parent of x so that we don’t have to traverse all intermediate nodes again.
- If x is root of a subtree, then path (to root) from all nodes under x also compresses.
- Path Compression & Union with Rank complement each other, complexity becomes even smaller than O(Logn)
- Amortized time complexity effectively becomes small constant

# Union-Find Algorithm

- Union-Find Algorithm can be used to check whether an undirected graph contains cycle or not.

# Union-Find Algorithm

Step of finding cycle:

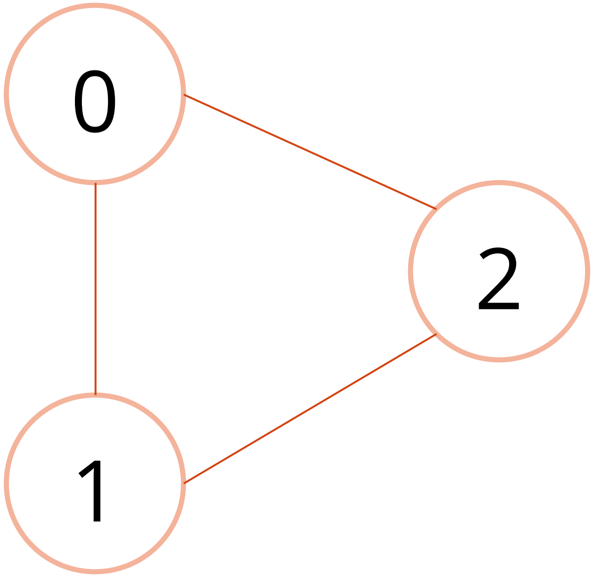
{0} {1} {2}

Process edge 0-1:

If 0 and 1 are in diff sets?

- Yes, Continue

{0, 1} {2}




Actual Graph

# Union-Find Algorithm

Step of finding cycle:

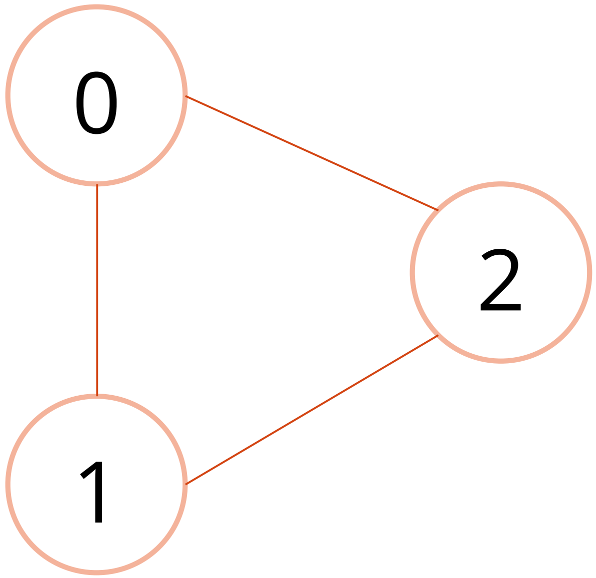
{0, 1}    {2}

Process edge 1-2:

If 1 and 2 are in diff sets?

- Yes, Continue

{0, 1, 2}




Actual Graph

# Union-Find Algorithm

Step of finding cycle:

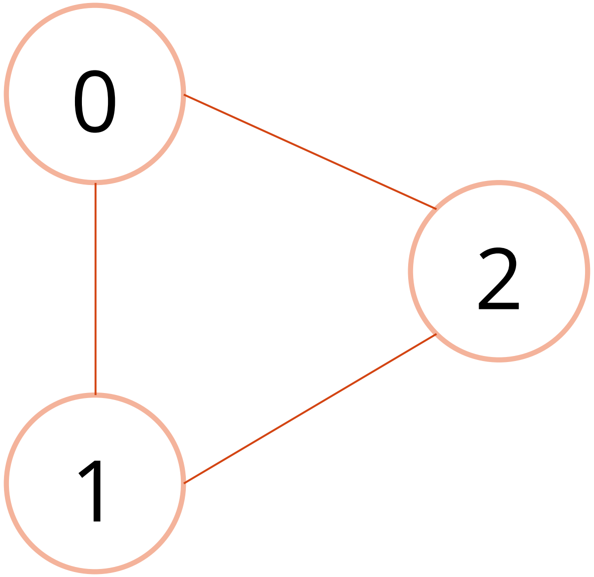
{0, 1, 2}

Process edge 0-2:

If 0 and 2 are in diff sets?

- No.

- **Cycle found!**




Actual Graph