Lecture 20: Disjoint Sets

CSE 373: Data Structures and Algorithms

Kruskal's Algorithm Implementation

```
KruskalMST(Graph G)
   initialize each vertex to be an independent component
   sort the edges by weight
   foreach(edge (u, v) in sorted order){
      if(u and v are in different components){
         add (u,v) to the MST
         update u and v to be in the same component
      }
   }
```

```
KruskalMST(Graph G)
   foreach (V : vertices) {
      makeMST(v); +?
   }
   sort edges in ascending order by weight +ElogE
   foreach(edge (u, v)){
      if(findMST(v) is not in findMST(u)){+?
         union(u, v)  +?
      }
   }
```

+V(makeMST)

+E(2findMST + union)

How many times will we call union?
V – 1
-> +Vunion + EfindMST

New ADT

Set ADT

state

Set of elements
- Elements must be unique!
- No required order

Count of Elements

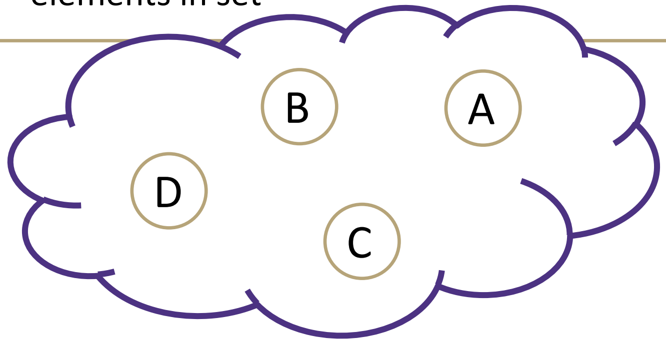
behavior

create(x) - creates a new set with a single member, x

add(x) - adds x into set if it is unique, otherwise add is ignored

remove(x) – removes x from set

size() – returns current number of elements in set



Disjoint-Set ADT

state

Set of Sets
- **Disjoint:** Elements must be unique across sets
- No required order
- Each set has representative

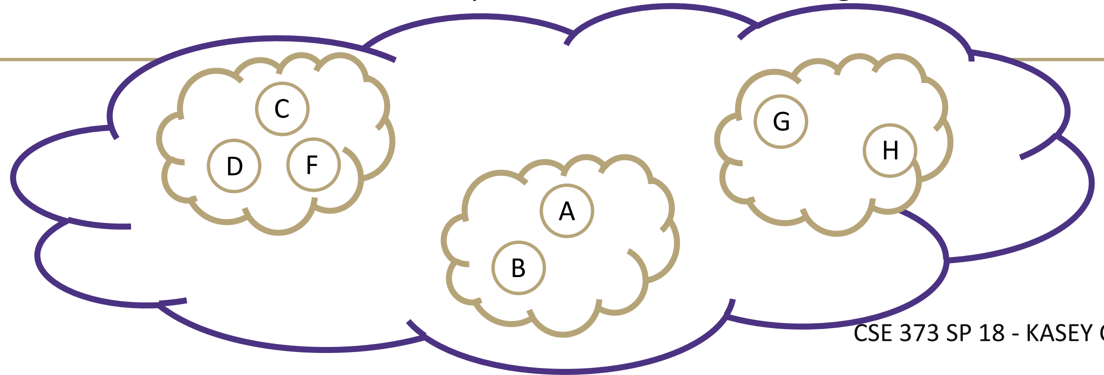
Count of Sets

behavior

makeSet(x) – creates a new set within the disjoint set where the only member is x. Picks representative for set

findSet(x) – looks up the set containing element x, returns representative of that set

union(x, y) – looks up set containing x and set containing y, combines two sets into one. Picks new representative for resulting set



Example

new()

makeSet(a)

makeSet(b)

makeSet(c)

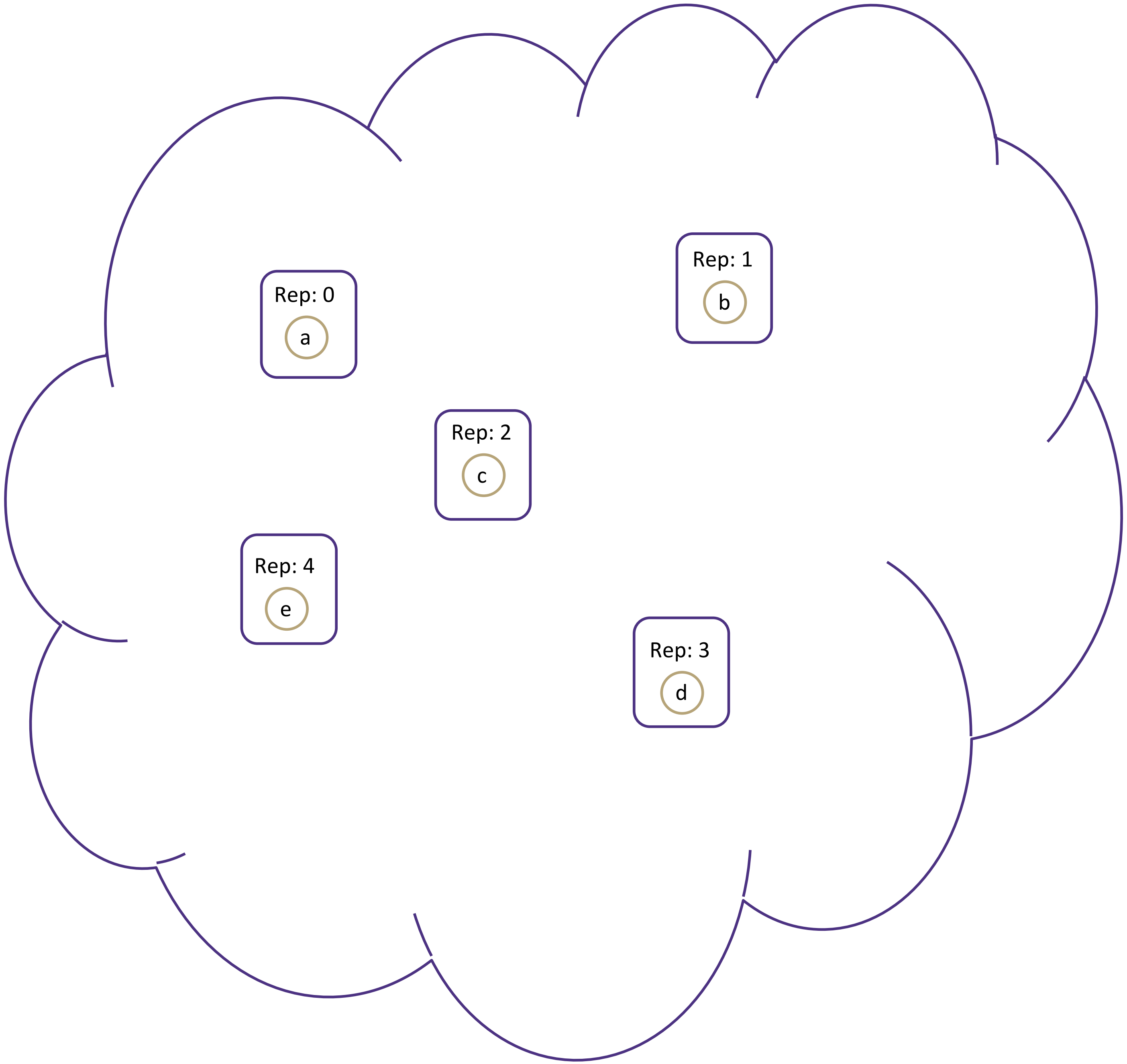
makeSet(d)

makeSet(e)

findSet(a)

findSet(d)

union(a, c)



Example

new()

makeSet(a)

makeSet(b)

makeSet(c)

makeSet(d)

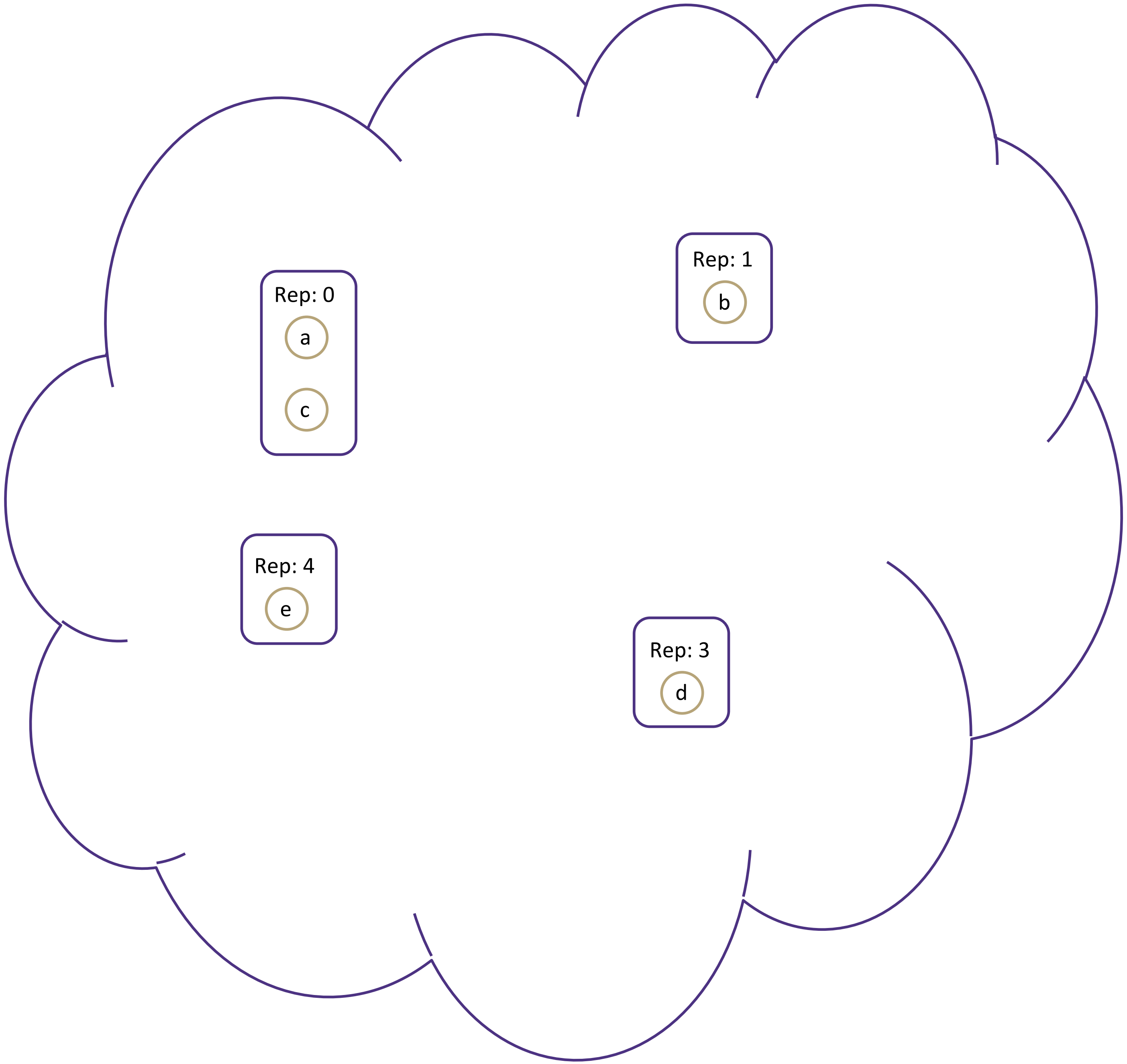
makeSet(e)

findSet(a)

findSet(d)

union(a, c)

union(b, d)



Example

new()

makeSet(a)

makeSet(b)

makeSet(c)

makeSet(d)

makeSet(e)

findSet(a)

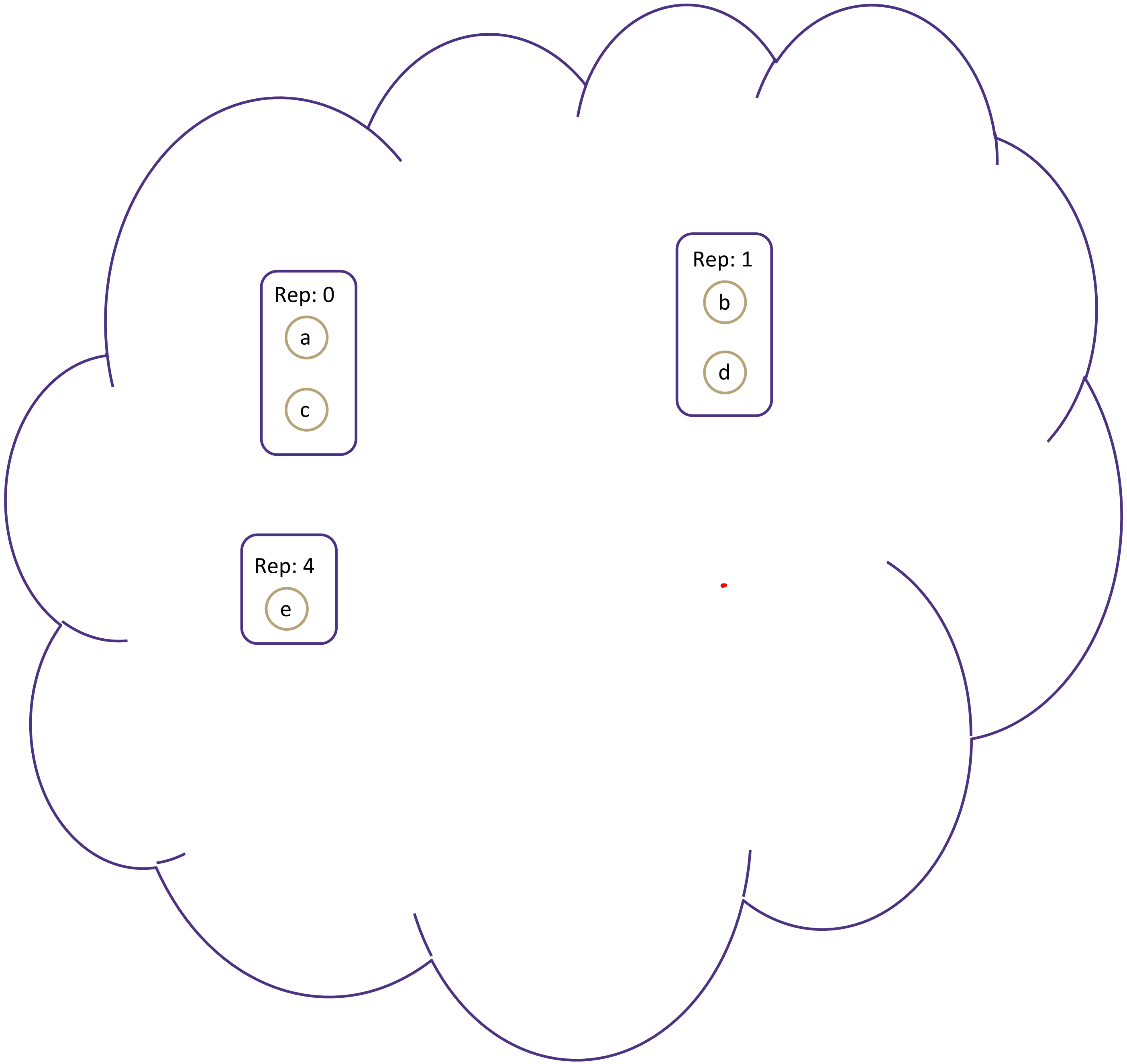
findSet(d)

union(a, c)

union(b, d)

findSet(a) == findSet(c)

findSet(a) == findSet(d)



Implementation

Disjoint-Set ADT

state

Set of Sets

- **Disjoint:** Elements must be unique across sets
- No required order
- Each set has representative

Count of Sets

behavior

makeSet(x) – creates a new set within the disjoint set where the only member is x. Picks representative for set

findSet(x) – looks up the set containing element x, returns representative of that set

union(x, y) – looks up set containing x and set containing y, combines two sets into one. Picks new representative for resulting set

TreeDisjointSet<E>

state

`Collection<TreeSet> forest`

`Dictionary<NodeValues, NodeLocations> nodeInventory`

behavior

`makeSet(x)`-create a new tree of size 1 and add to our forest

`findSet(x)`-locates node with x and moves up tree to find root

`union(x, y)`-append tree with y as a child of tree with x

TreeSet<E>

state

`SetNode overallRoot`

behavior

`TreeSet(x)`

`add(x)`

`remove(x, y)`

`getRep()`-returns data of overallRoot

SetNode<E>

state

`E data`

`Collection<SetNode> children`

behavior

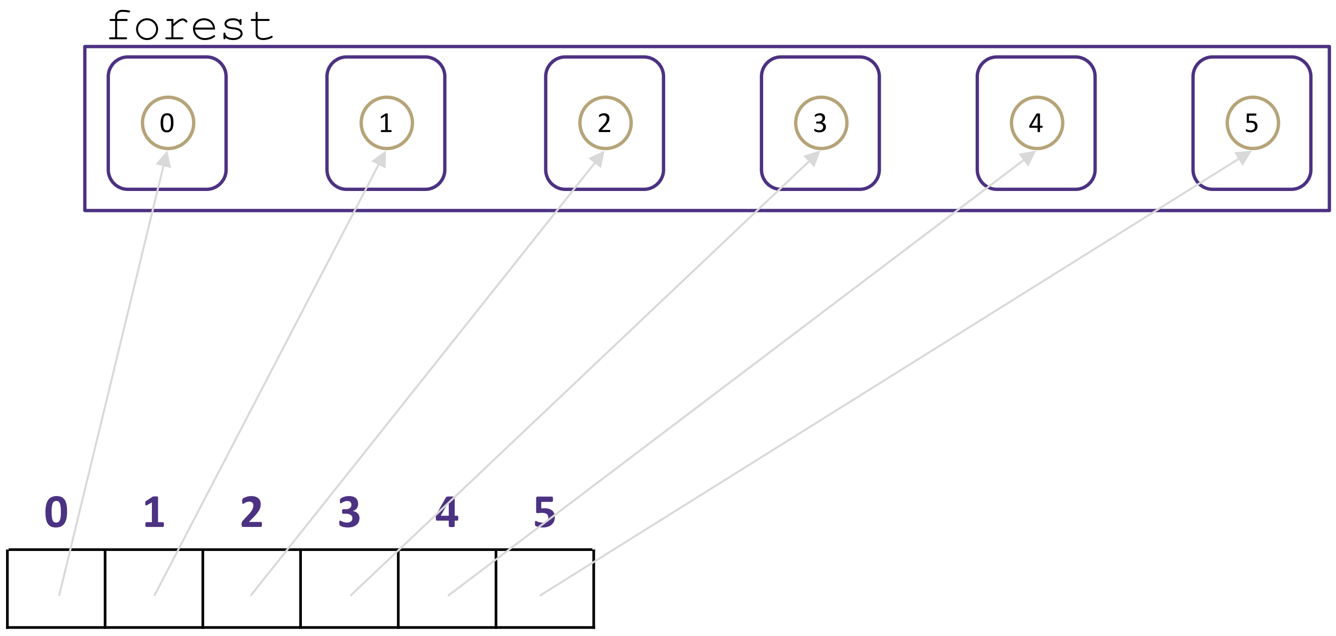
`SetNode(x)`

`addChild(x)`

`removeChild(x, y)`

Implement makeSet(x)

```
makeSet(0)
makeSet(1)
makeSet(2)
makeSet(3)
makeSet(4)
makeSet(5)
```



TreeDisjointSet<E>

state

```
Collection<TreeSet> forest
Dictionary<NodeValues, NodeLocations> nodeInventory
```

behavior

`makeSet(x)`-create a new tree of size 1 and add to our forest

`findSet(x)`-locates node with x and moves up tree to find root

`union(x, y)`-append tree with y as a child of tree with x

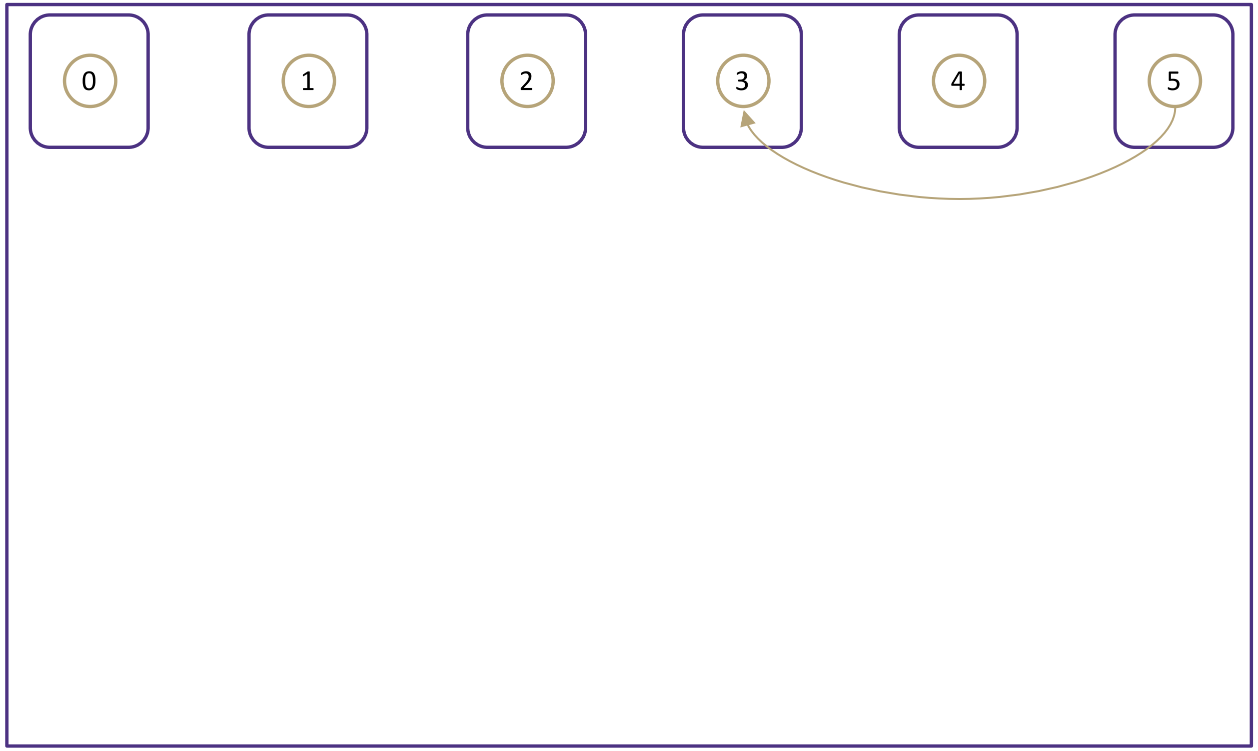
Worst case runtime?

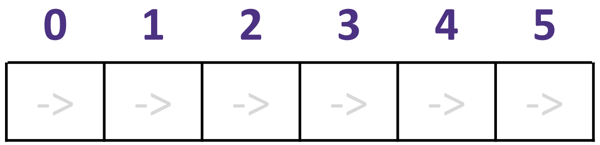
O(1)

Implement union(x, y)

```
union(3, 5)
```

forest





TreeDisjointSet<E>

state

```
Collection<TreeSet> forest
Dictionary<NodeValues, NodeLocations> nodeInventory
```

behavior

`makeSet(x)`-create a new tree of size 1 and add to our forest

`findSet(x)`-locates node with x and moves up tree to find root

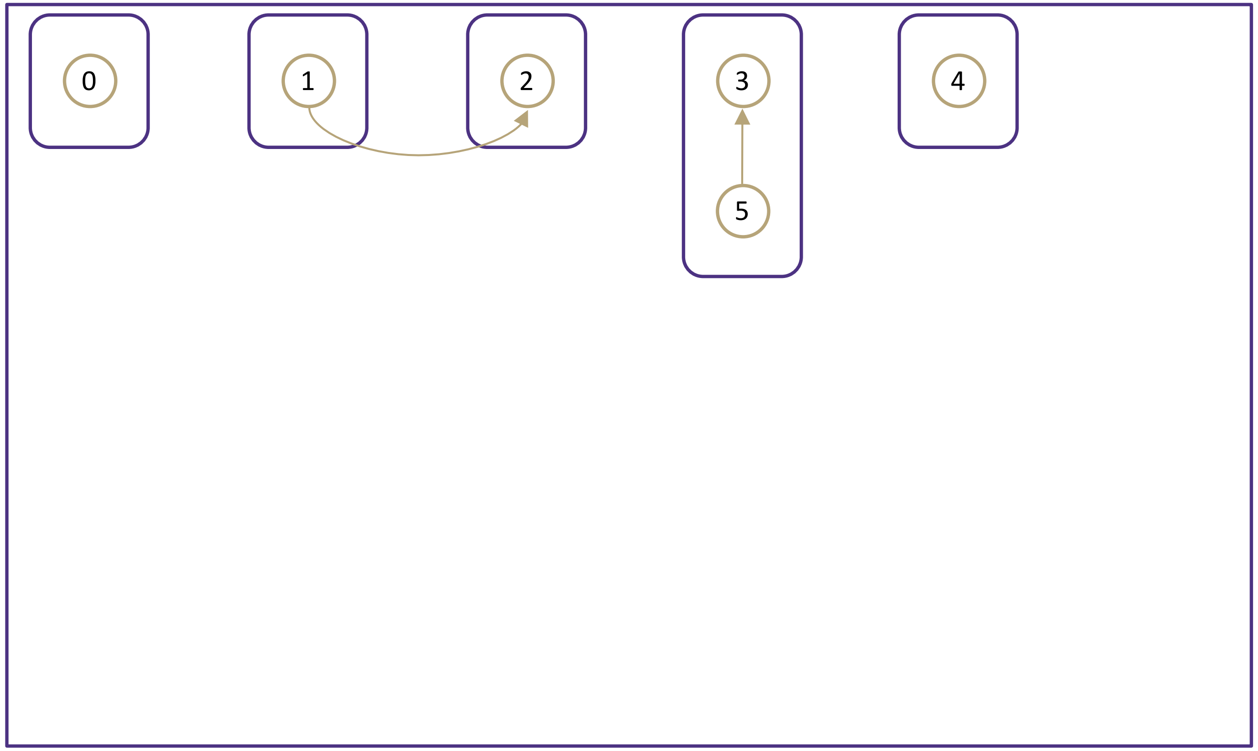
`union(x, y)`-append tree with y as a child of tree with x

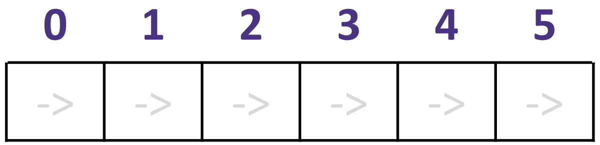
Implement union(x, y)

```
union(3, 5)

union(2, 1)
```

forest





TreeDisjointSet<E>

state

```
Collection<TreeSet> forest
Dictionary<NodeValues,
NodeLocations> nodeInventory
```

behavior

`makeSet(x)`-create a new tree of size 1 and add to our forest

`findSet(x)`-locates node with x and moves up tree to find root

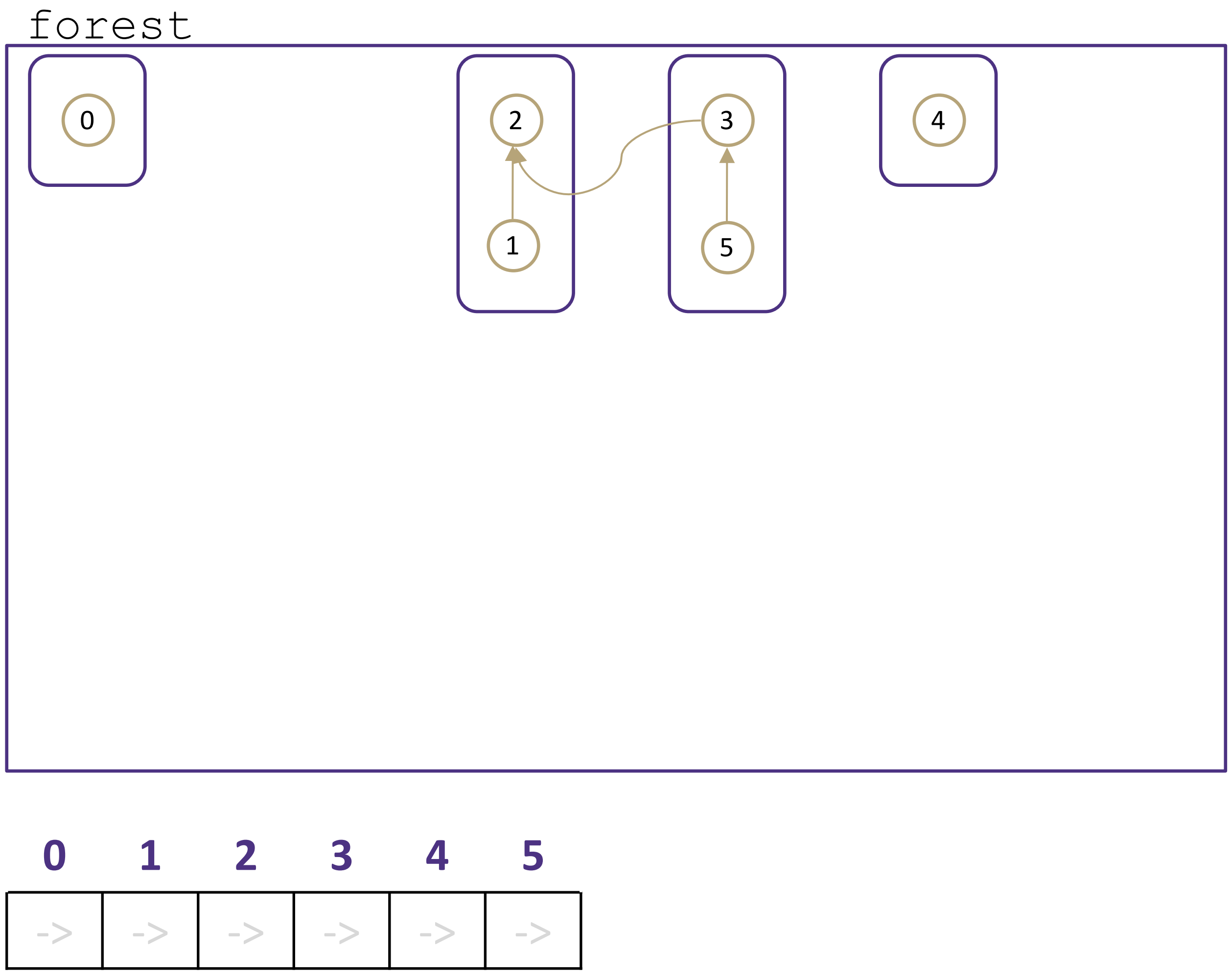
`union(x, y)`-append tree with y as a child of tree with x

Implement union(x, y)

```
union(3, 5)

union(2, 1)

union(2, 5)
```



TreeDisjointSet<E>

state

Collection<TreeSet> forest

Dictionary<NodeValues, NodeLocations> nodeInventory

behavior

makeSet(x)-create a new tree of size 1 and add to our forest

findSet(x)-locates node with x and moves up tree to find root

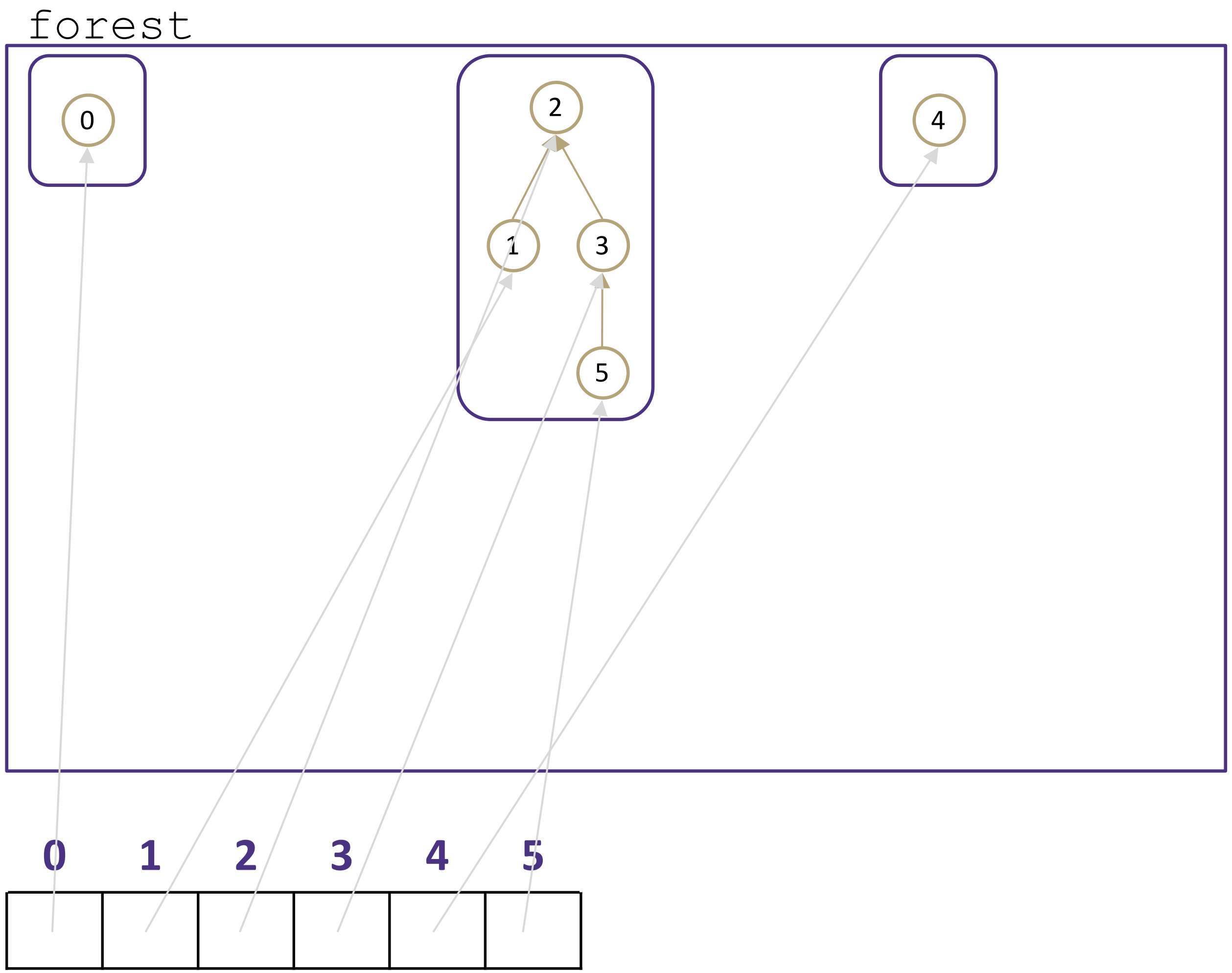
union(x, y)-append tree with y as a child of tree with x

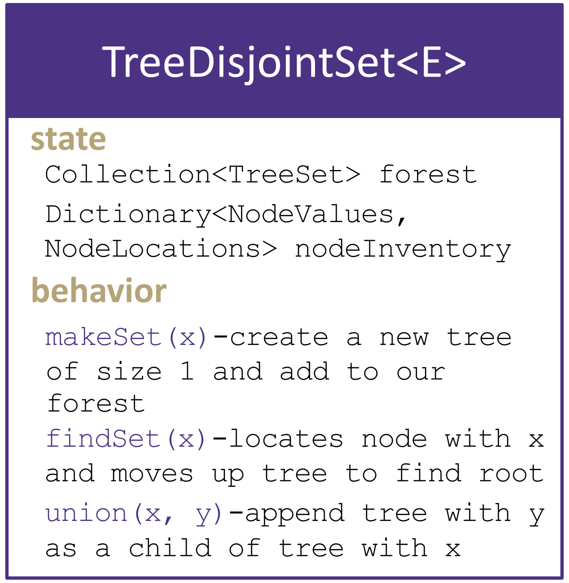
Implement union(x, y)

```
union(3, 5)

union(2, 1)

union(2, 5)
```



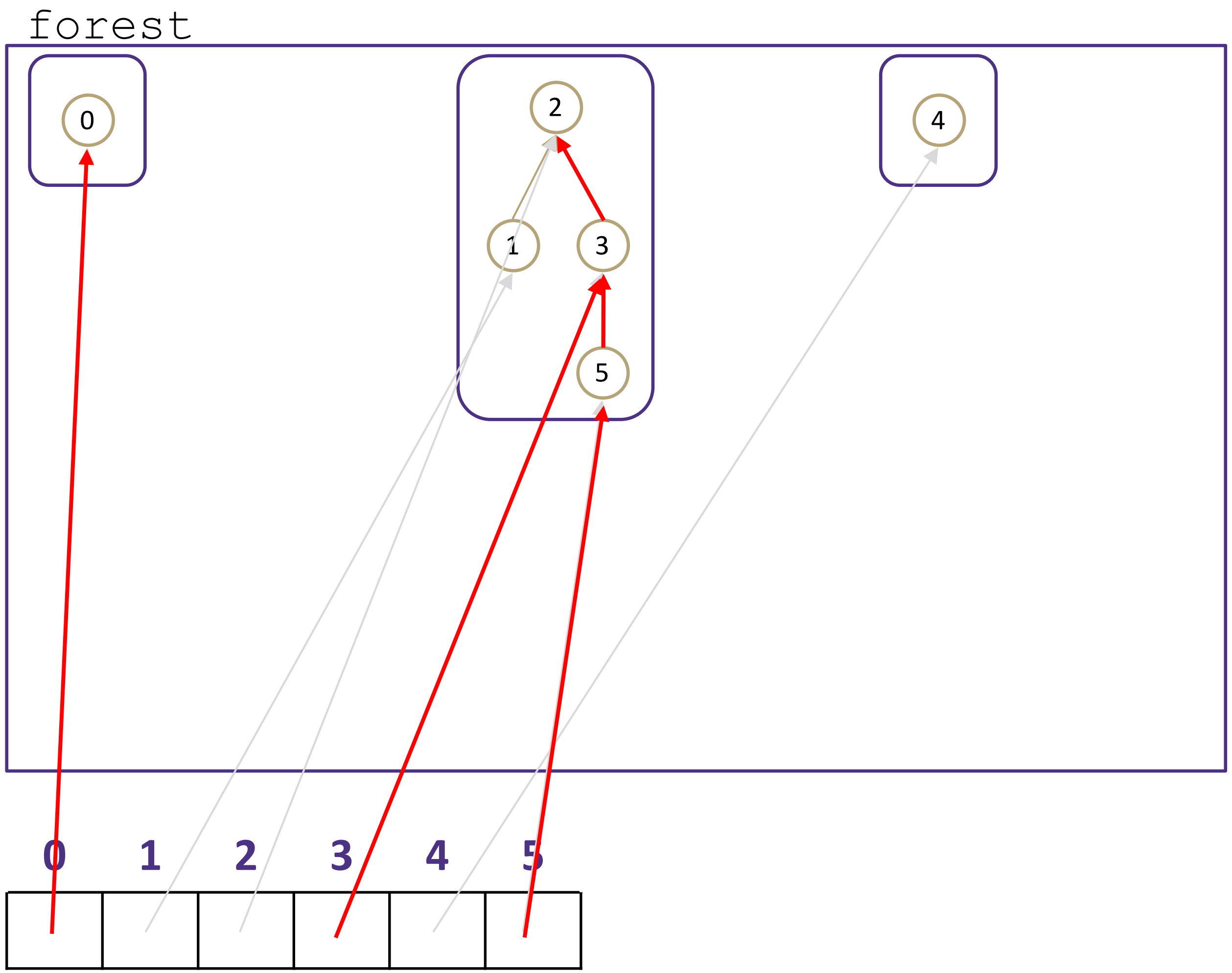


Implement findSet(x)

findSet(0)

findSet(3)

findSet(5)



Worst case runtime?

O(n)

Worst case runtime of union?

O(n)

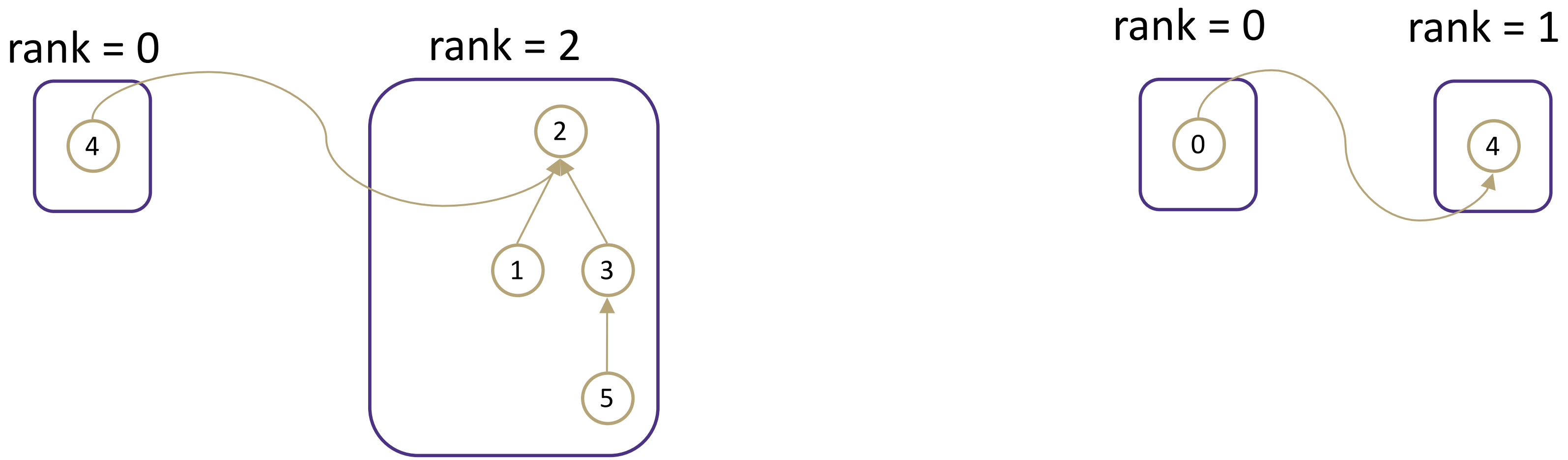
TreeDisjointSet<E>
state
`Collection<TreeSet> forest`
`Dictionary<NodeValues, NodeLocations> nodeInventory`
behavior
`makeSet(x)`-create a new tree of size 1 and add to our forest
`findSet(x)`-locates node with x and moves up tree to find root
`union(x, y)`-append tree with y as a child of tree with x

Improving union

Problem: Trees can be unbalanced

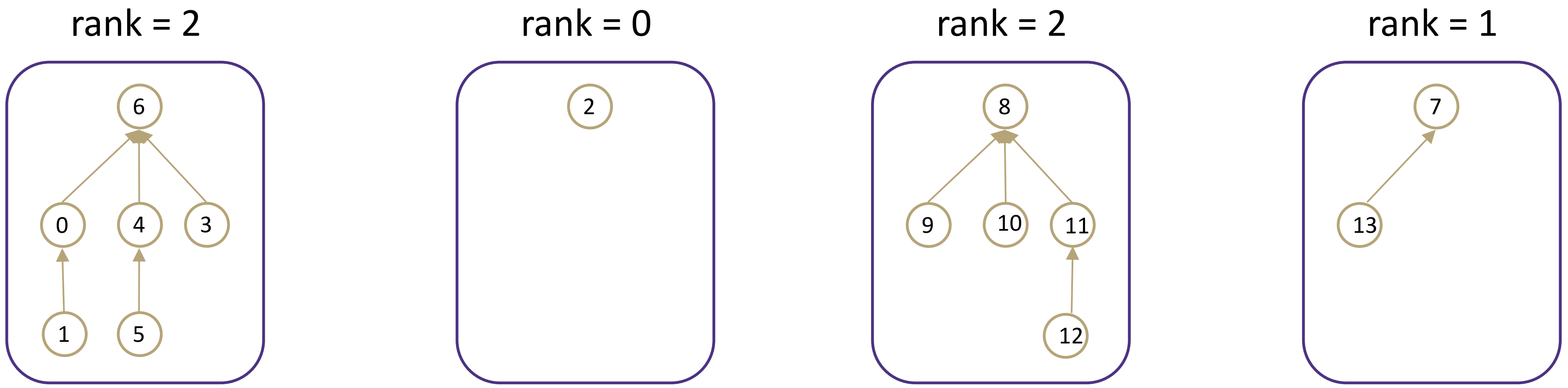
Solution: Union-by-rank!

- let rank(x) be a number representing the upper bound of the height of x so rank(x) >= height(x)
- Keep track of rank of all trees
- When unioning make the tree with larger rank the root
- If it's a tie, pick one randomly and increase rank by one



Practice

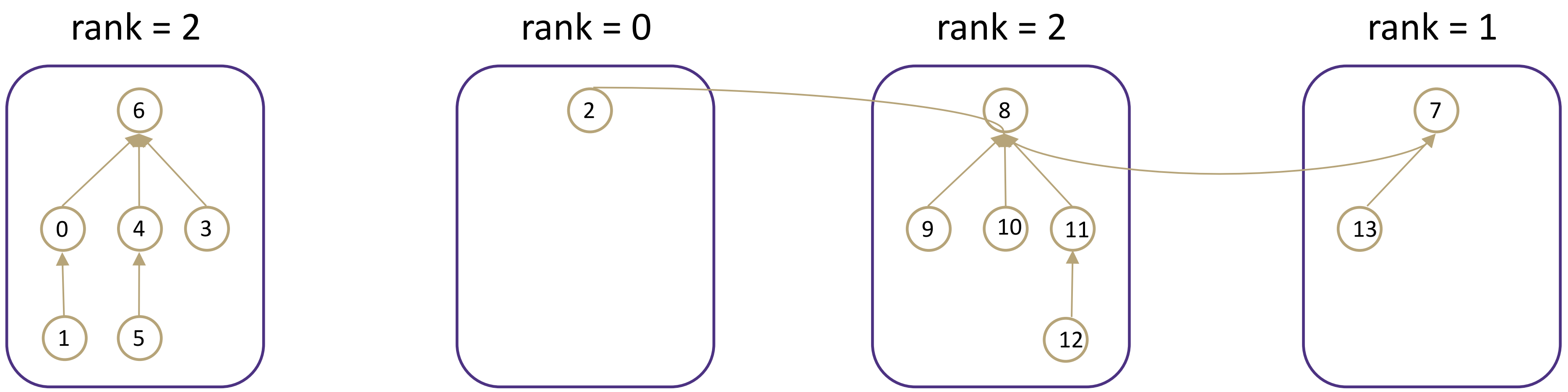
Given the following disjoint-set what would be the result of the following calls on union if we add the “union-by-rank” optimization. Draw the forest at each stage with corresponding ranks for each tree.




```
union(2, 13)
union(4, 12)
union(2, 8)
```

Practice

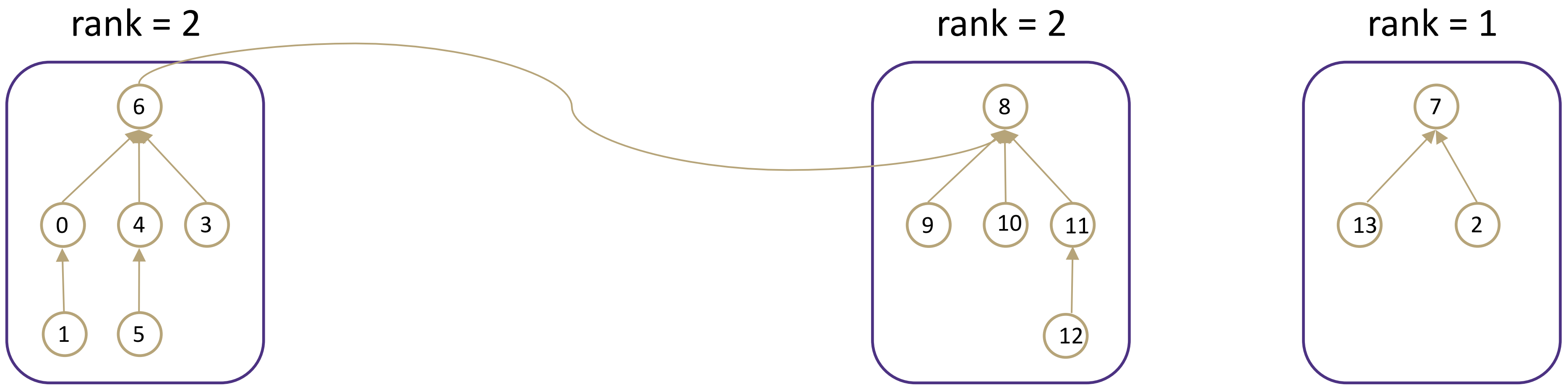
Given the following disjoint-set what would be the result of the following calls on union if we add the "union-by-rank" optimization. Draw the forest at each stage with corresponding ranks for each tree.



`union(2, 13)`

Practice

Given the following disjoint-set what would be the result of the following calls on union if we add the "union-by-rank" optimization. Draw the forest at each stage with corresponding ranks for each tree.

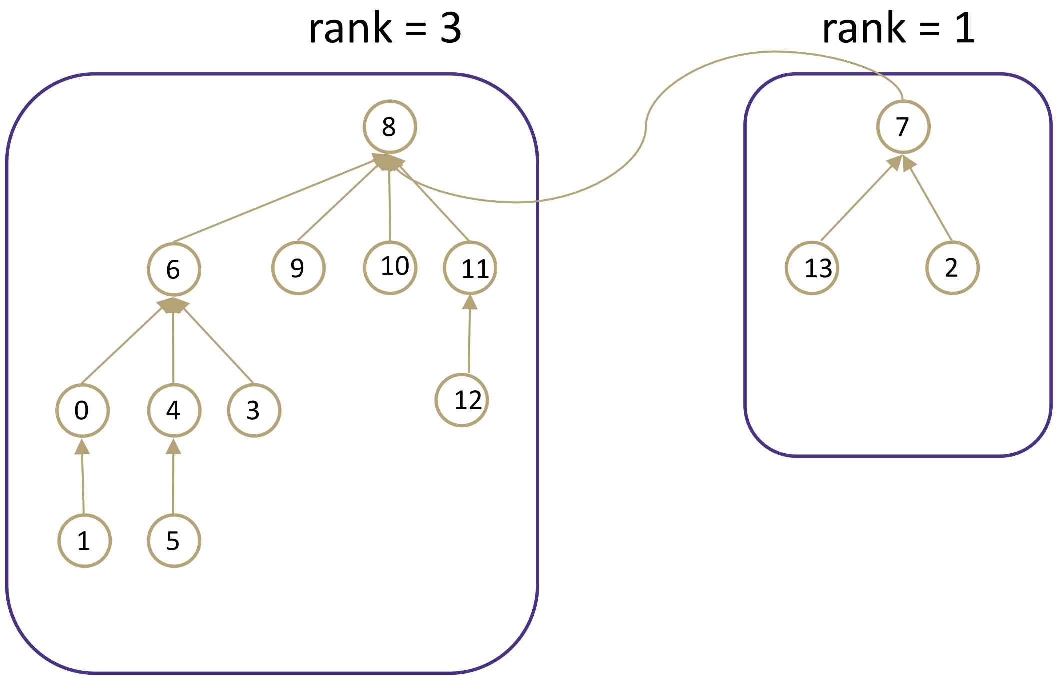



```
union(2, 13)

union(4, 12)
```

Practice

Given the following disjoint-set what would be the result of the following calls on union if we add the “union-by-rank” optimization. Draw the forest at each stage with corresponding ranks for each tree.

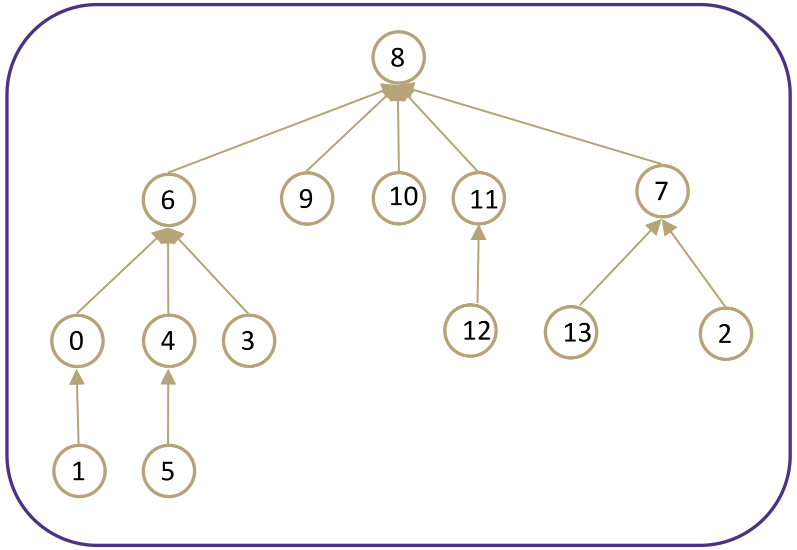



```
union(2, 13)
union(4, 12)
union(2, 8)
```

Practice

Given the following disjoint-set what would be the result of the following calls on union if we add the "union-by-rank" optimization. Draw the forest at each stage with corresponding ranks for each tree.

rank = 3




```
union(2, 13)
union(4, 12)
union(2, 8)
```

Does this improve the worst case runtimes?

findSet is more likely to be O(log(n)) than O(n)

Improving findSet()

Problem: Every time we call findSet() you must traverse all the levels of the tree to find representative

Solution: Path Compression

- Collapse tree into fewer levels by updating parent pointer of each node you visit
- Whenever you call findSet() update each node you touch's parent pointer to point directly to overallRoot

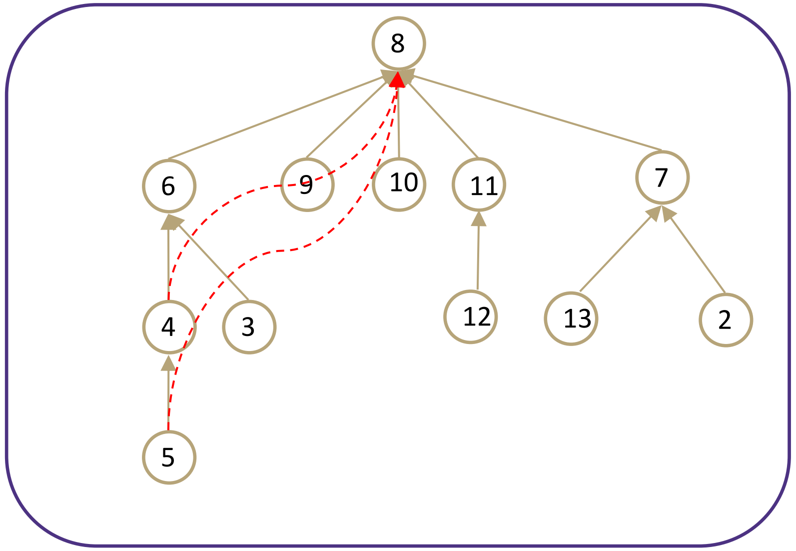
```
findSet(5)

findSet(4)
```

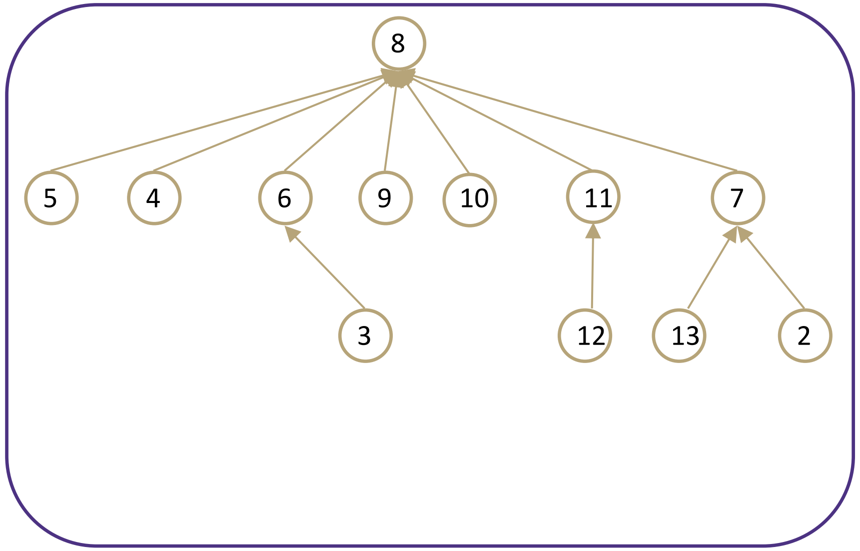
Does this improve the worst case runtimes?

findSet is more likely to be O(1) than O(log(n))

rank = 3



rank = 2

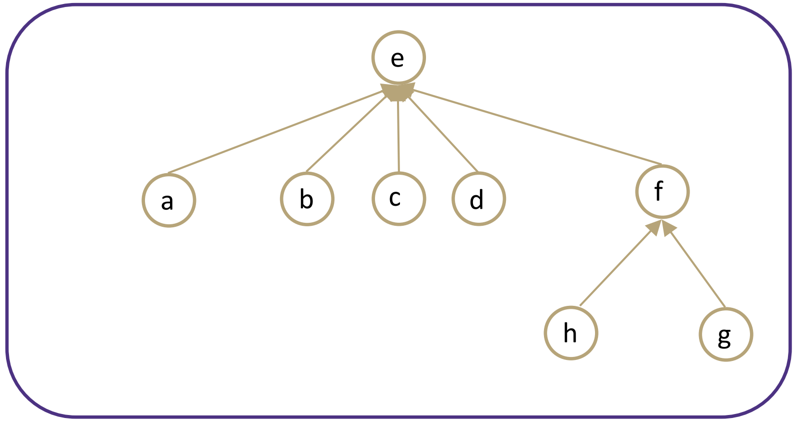


Example

Using the union-by-rank and path-compression optimized implementations of disjoint-sets draw the resulting forest caused by these calls:

1. makeSet(a)
2. makeSet(b)
3. makeSet(c)
4. makeSet(d)
5. makeSet(e)
6. makeSet(f)
7. makeSet(h)
8. union(c, e)
9. union(d, e)
10. union(a, c)
11. union(g, h)
12. union(b, f)
13. union(g, f)
14. union(b, c)

rank = 2



Optimized Disjoint Set Runtime

makeSet(x)

Without Optimizations O(1)

With Optimizations O(1)

findSet(x)

Without Optimizations O(n)

With Optimizations Best case: O(1) Worst case: O(logn)

union(x, y)

Without Optimizations O(n)

With Optimizations Best case: O(1) Worst case: O(logn)

Worksheet question 1

```
function Kruskal(Graph G)
    initialize each vertex to be a component
    sort all edges by weight
    for each edge (u, v) in sorted order do
        if u and v are in different components then
            add edge (u,v) to the MST
            update u and v to be in the same component
        end if
    end for
end function
```

Worksheet question 1

```
function Kruskal(Graph G)
    initialize a disjoint set; call makeSet() on each vertex
    sort all edges by weight
    for each edge (u, v) in sorted order do
        if findSet(u) ≠ findSet(v) then
            add edge (u,v) to the MST
            union(u, v)
        end if
    end for
end function
```

Implementation

Use Nodes?

In modern Java (assuming 64-bit JDK) each object takes about 32 bytes

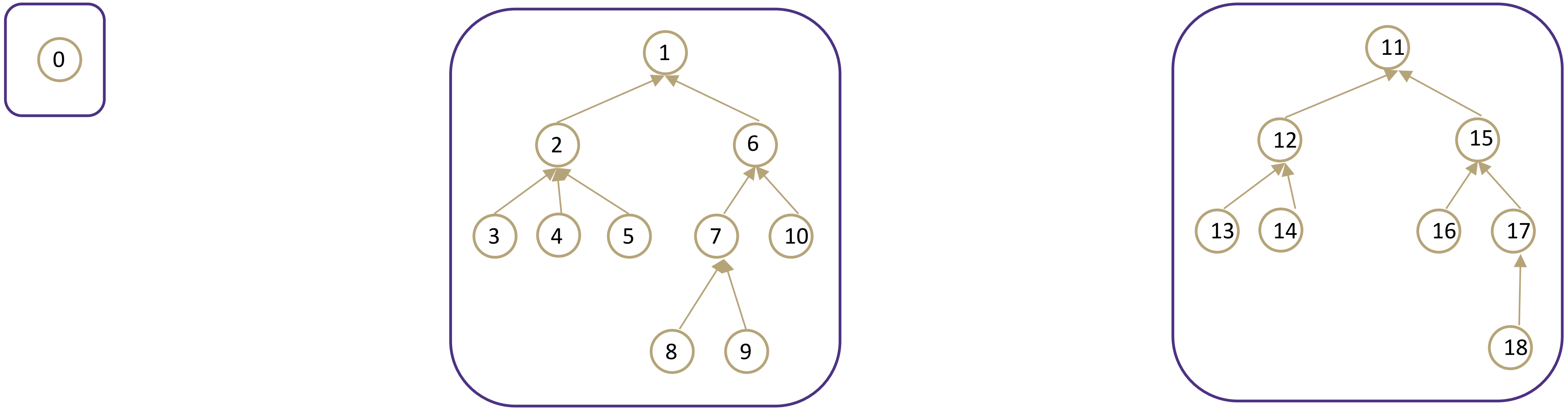
- int field takes 4 bytes
- Pointer takes 8 bytes
- Overhead ~ 16 bytes
- Adds up to 28, but we must partition in multiples of 8 => 32 bytes

Use arrays instead!

- Make index of the array be the vertex number
  - Either directly to store ints or representationally
  - We implement makeSet(x) so that **we** choose the representative
- Make element in the array the index of the parent

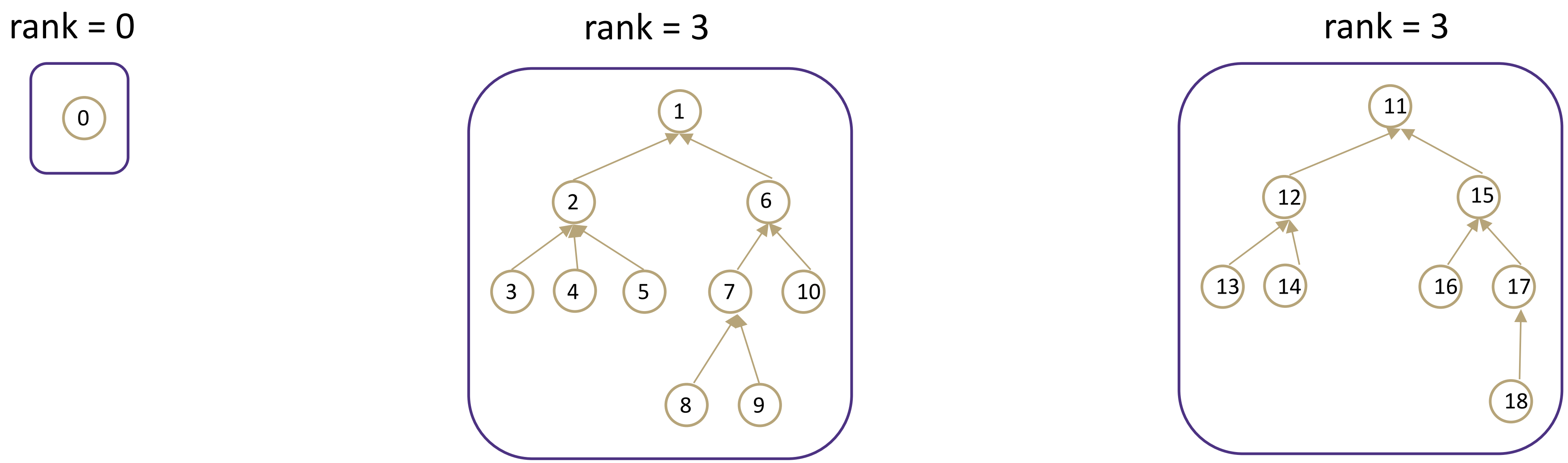
Array implementation

rank = 0 rank = 3 rank = 3



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Array implementation

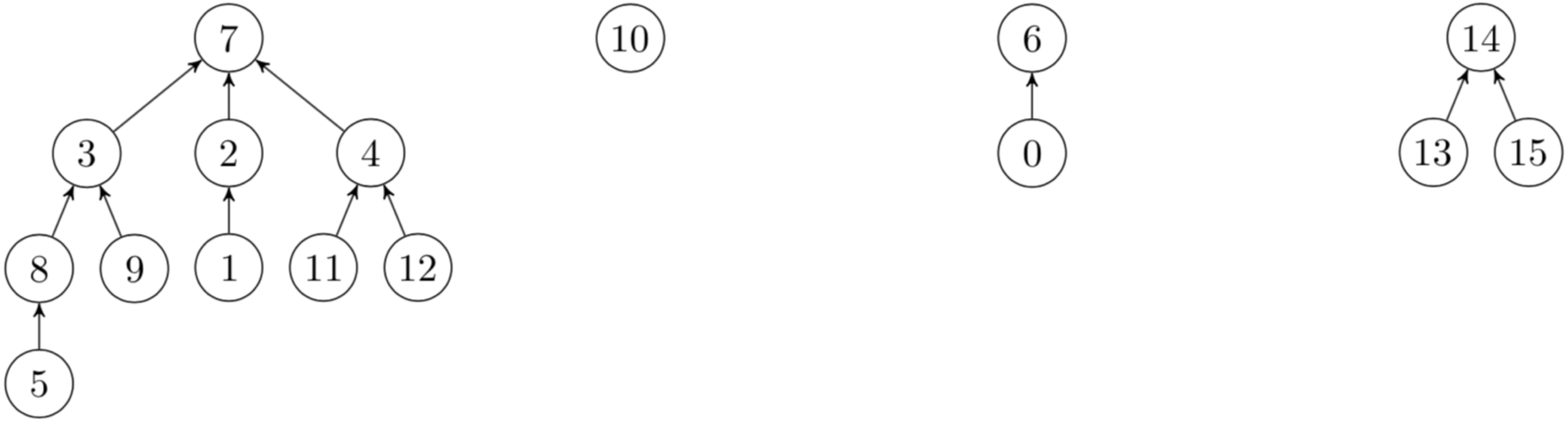


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-1	-4	1	2	2	2	1	6	7	7	6	-4	11	12	12	11	15	15	17

Store (rank * -1) - 1

Example

Consider the following disjoint set. Assume that (from left) the first tree has rank 3, the second has rank 0, the third has rank 1, and the last tree has rank 1.



Write the array representation of this disjoint set in the array below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Example

Consider the following disjoint set. Assume that (from left) the first tree has rank 3, the second has rank 0, the third has rank 1, and the last tree has rank 1.

Write the array representation of this disjoint set in the array below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6	2	7	7	7	8	-2	-4	3	3	-1	4	4	14	-2	14

Array method implementation

makeSet(x)

add new value to array with a rank of -1

findSet(x)

Jump into array at index/value you’re looking for, jump to parent based on element at that index, continue until you hit negative number

union(x, y)

findSet(x) and findSet(y) to decide who has larger rank, update element to represent new parent as appropriate

Graph Review

Graph Definitions/Vocabulary
- Vertices, Edges
- Directed/undirected
- Weighted
- Etc…

Graph Traversals
- Breadth First Search
- Depth First Search

Finding Shortest Path
- Dijkstra's

Topological Sort, Strongly connected components

Minimum Spanning Trees
- Primm's
- Kruskal's

Disjoint Sets
- Implementing Kruskal's