

CS2040S – Data Structures and Algorithms

Lecture 10 – UFDS

chongket@comp.nus.edu.sg



Motivation for UFDS

- We have seen that we can use Hashtable to implement simple Set ADT with the following operations
 - Find
 - Insert
 - remove
- But what if we need to represent multiple disjoint sets and also to union them?

A simple yet effective data structure to model disjoint sets...

<https://visualgo.net/en/ufds>

CP4 Book 1, Section 2.4.2

UNION-FIND DISJOINT SETS DATA STRUCTURE

Union-Find Disjoint Sets (UFDS)

UFDS is a collection of disjoint sets

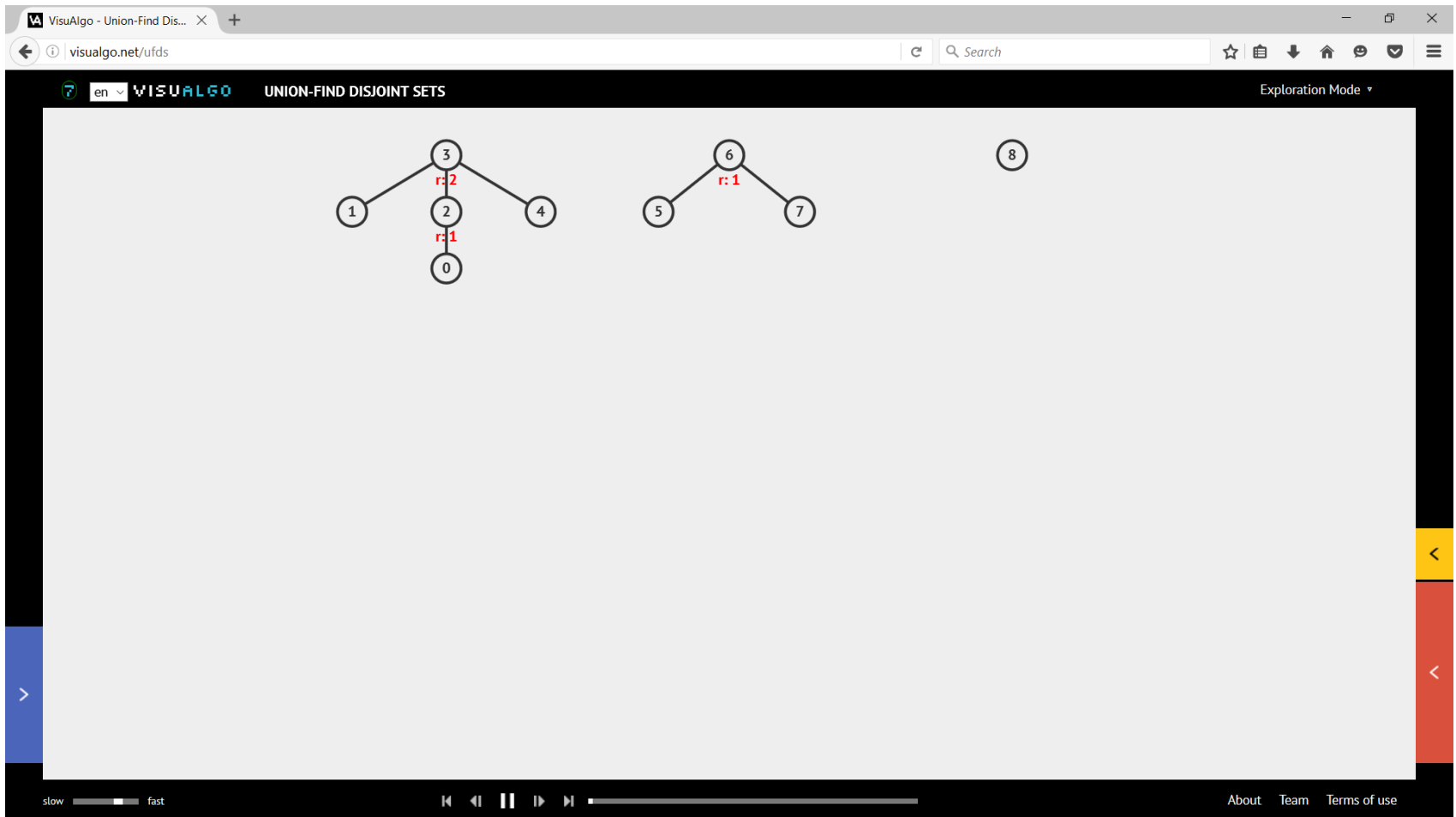
Given several disjoint sets in the UFDS the **operations** we have are

- Union two disjoint sets when needed
- Find which set an item belongs to
- Check if two items belong to the same set

Key ideas:

- Each set is modeled **as a tree**
 - Thus a collection of disjoint sets form **a forest of trees**
- Each set is represented by a **representative item**
 - Which is the root of the corresponding tree of that set

Example with 3 Disjoint Sets

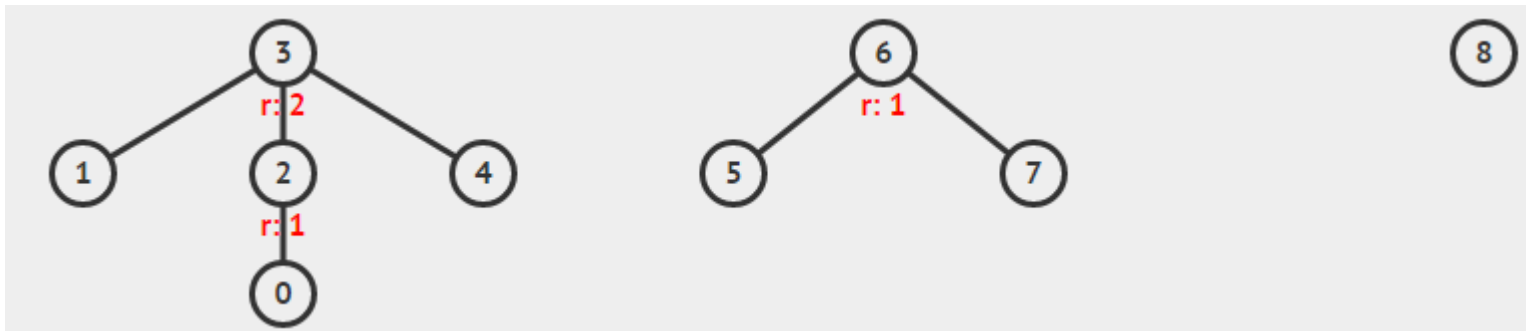


Data Structure to store UFDS

We can record this forest of trees with an array \mathbf{p}

- $\mathbf{p}[i]$ records the parent of item i
- if $\mathbf{p}[i] = i$, then i is a root
 - And also the representative item of the set that contains i

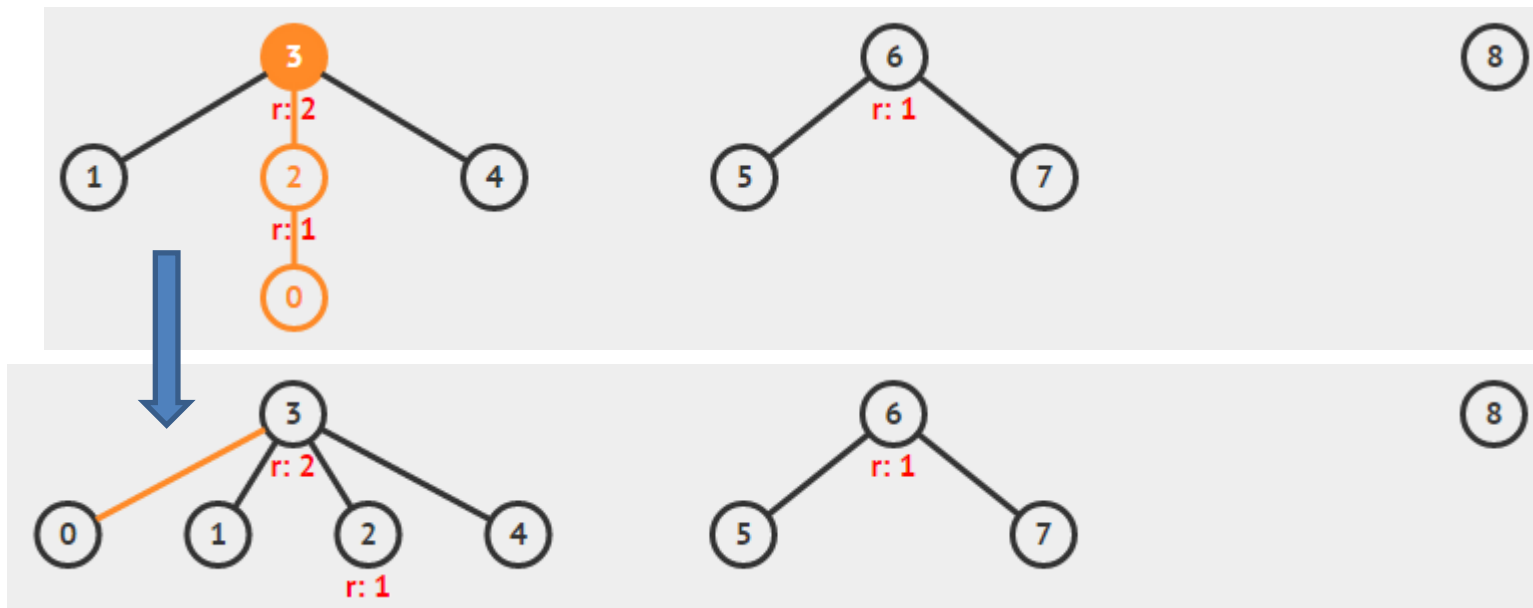
For the example below, we have $\mathbf{p} = \{2,3,3,3,3,6,6,6,8\}$
index: 0,1,2,3,4,5,6,7,8



UFDS – findSet(i) Operation

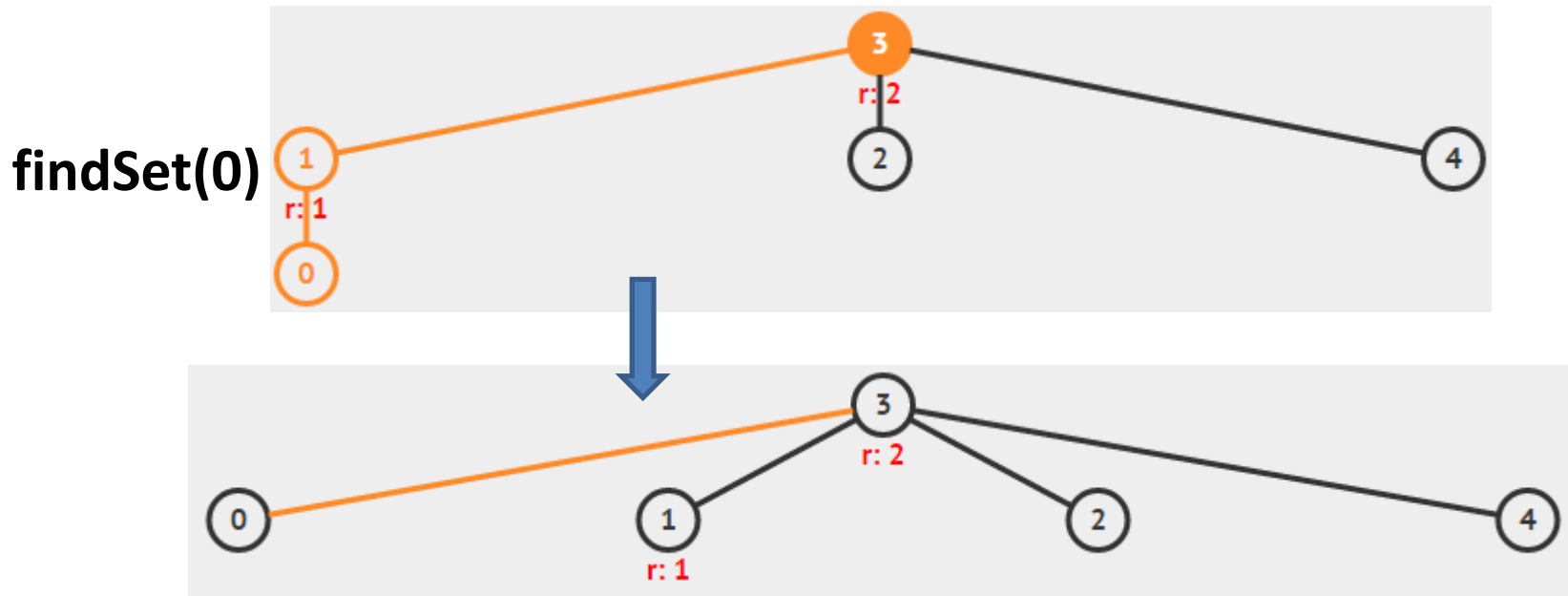
For each item i , we can find the representative item of the set that contains item i by recursively visiting $p[i]$ until $p[i] = i$; Then, we *compress the path* to make future find operations (very) fast, i.e. $O(1)$

- Example of findSet(0), *ignore attribute 'r' for now*



findSet code

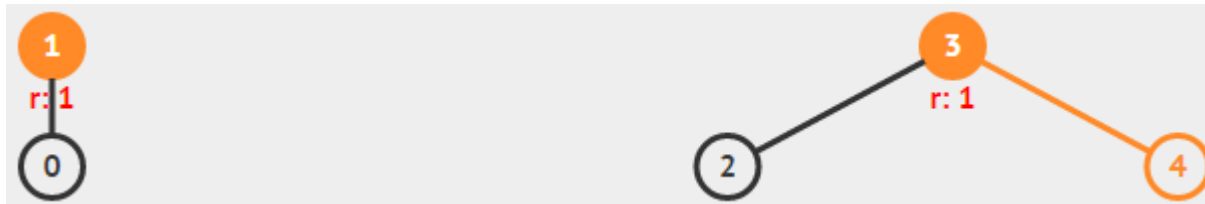
```
public int findSet(int i) {  
    if (p[i] == i)  
        return i;  
    else {  
        p[i] = findSet(p[i]);  
        return p[i];  
    }  
}
```



UFDS – isSameSet(i,j) Operation

For item i and j we can check whether they are in the same set in $O(1)$ by finding the representative item for i and j and checking if they are the same or not

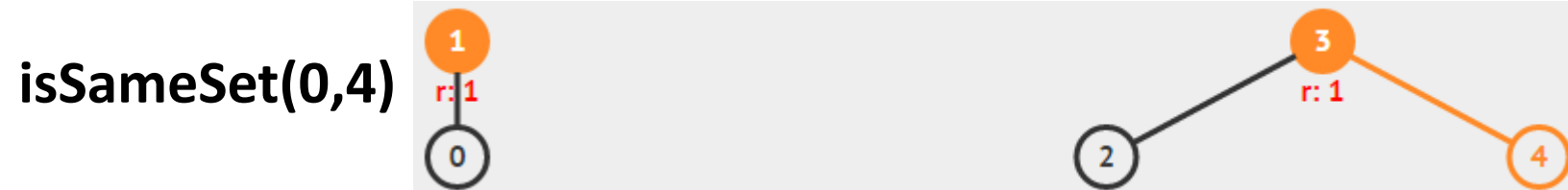
- Example: `isSameSet(0,4)` will return false



isSameSet code

```
public Boolean isSameSet(int i, int j) {  
    return findSet(i) == findSet(j);  
}
```

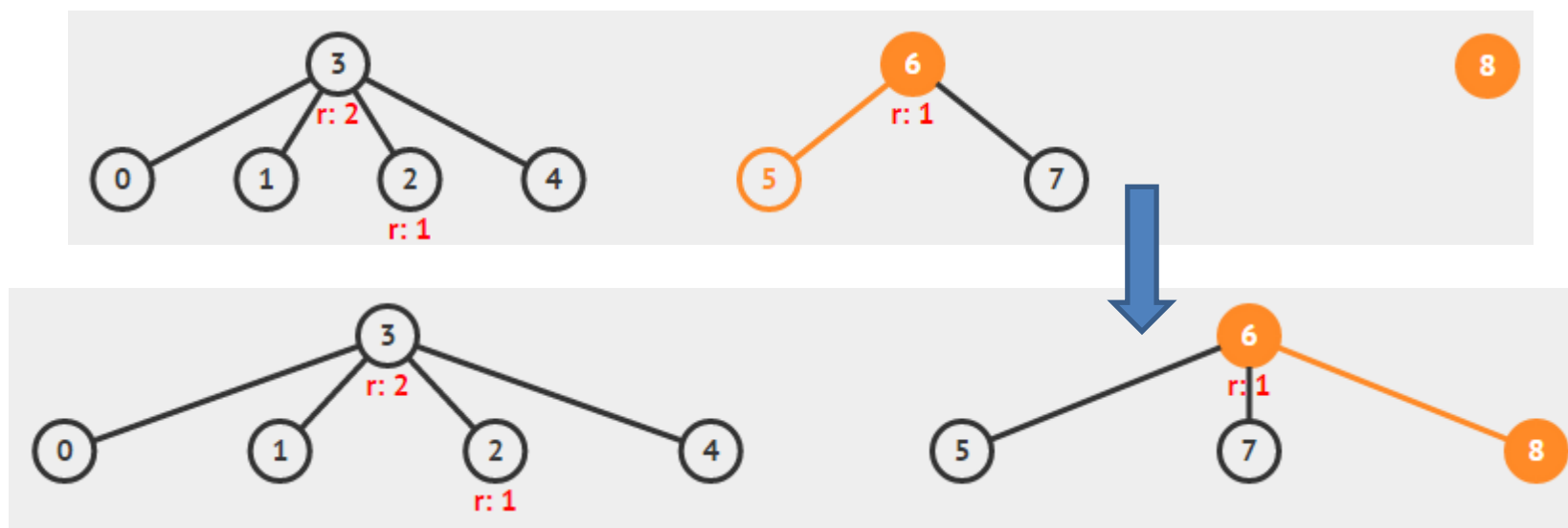
As the representative items of the sets that contains item **0** and **4** are different, we say that **0** and **4** are **not** in the same set!



UFDS – unionSet(i,j) Operation (1)

If two items i and j currently belong to different disjoint sets, we can **union** them by setting the representative item of the one with taller* tree to be the new representative item of the combined set

- Example of unionSet(5, 8), see attribute 'r' (elaborated soon)



UFDS – unionSet(i,j) Operation (2)

This is called the “*Union-by-Rank*” heuristic

- This helps to make the resulting combined tree shorter
 - Convince yourself that doing the opposite action will make the resulting tree taller (we do not want this)

If both trees are equally tall, this heuristic is not used

We use another integer array **rank**, where **rank[i]** stores the upper bound of the height of (sub)tree rooted at **i**

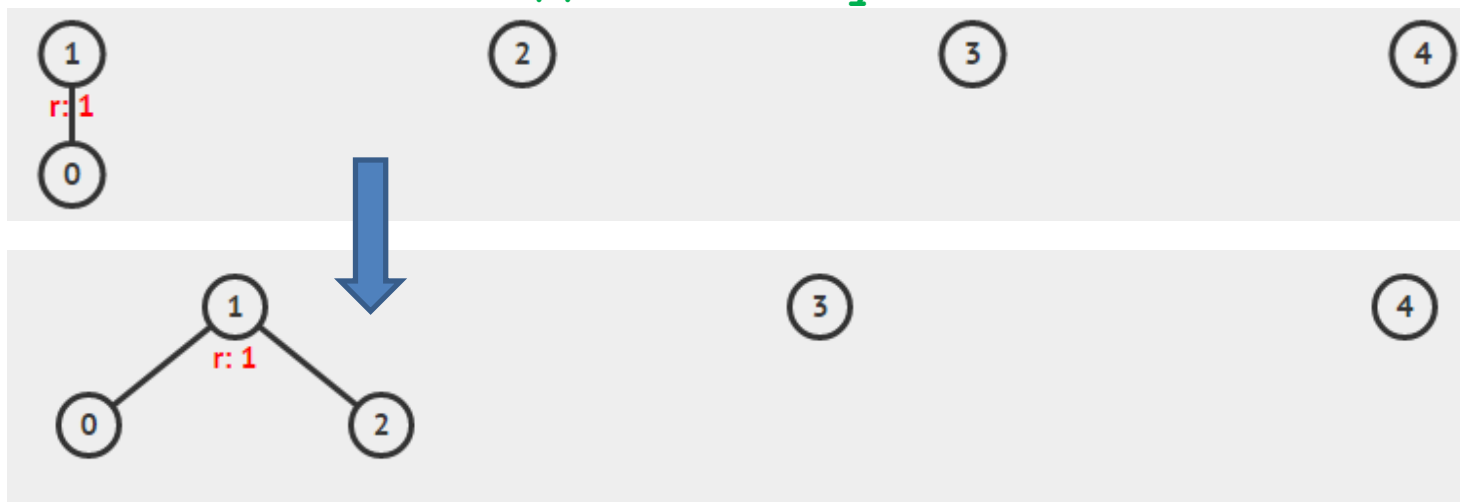
- This is just an upper bound as path compressions can make (sub)trees shorter than its upper bound and we do not want to waste effort maintaining the correctness of **rank[i]**

unionSet code

```

public void unionSet(int i, int j) {
    if (!isSameSet(i, j)) {
        int x = findSet(i), y = findSet(j);
        // rank is used to keep the tree short
        if (rank[x] > rank[y])
            p[y] = x;
        else {
            p[x] = y;
            if (rank[x] == rank[y]) // rank increases
                rank[y] = rank[y] + 1; // only if both trees
                                      // initially have the same rank
        }
    }
}

```



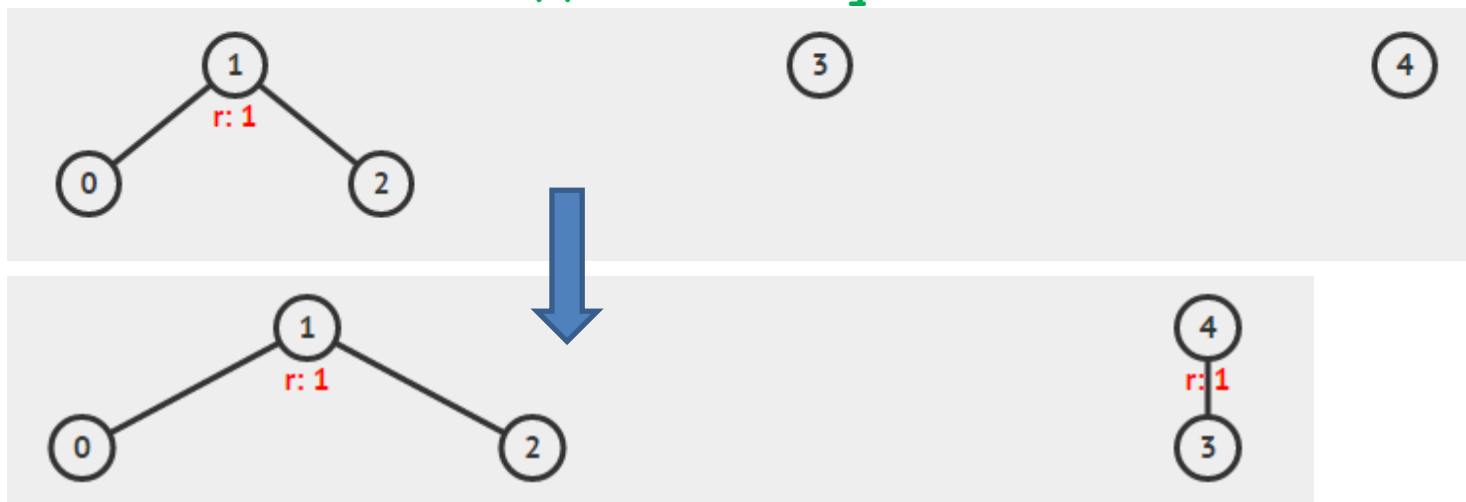
unionSet(0,2)

unionSet code

```

public void unionSet(int i, int j) {
    if (!isSameSet(i, j)) {
        int x = findSet(i), y = findSet(j);
        // rank is used to keep the tree short
        if (rank[x] > rank[y])
            p[y] = x;
        else {
            p[x] = y;
            if (rank[x] == rank[y]) // rank increases
                rank[y] = rank[y] + 1; // only if both trees
                                      // initially have the same rank
        }
    }
}

```



unionSet(3,4)

Constructor, UnionFind(N)

```
class UnionFind {  
    public int[] p;  
    public int[] rank;  
  
    public UnionFind(int N) {  
        p = new int[N];  
        rank = new int[N];  
        for (int i = 0; i < N; i++) {  
            p[i] = i;  
            rank[i] = 0;  
        }  
    }  
  
    // ... other methods in the previous slides  
}
```

UnionFind(5)

0

1

2

3

4

UFDS – Summary

That's the basics... we will not go into further details


- UFDS operations runs in just $O(\alpha(N))$ if UFDS is implemented with both “union-by-rank” and “path-compression” heuristics
 - $\alpha(N)$ is called the **inverse Ackermann** function
 - This function grows very slowly
 - You can assume it is “constant”, i.e. $O(1)$ for practical values of N ($\leq 1M$)
- Review UFDS at <https://visualgo.net/en/ufds> and again train lots on Visualgo

VisuAlgo UFDS Exercise (1)

First, click “**Initialize(N)**”, enter **6**, then click “**Go**”

Do a sequence of union and/or find operations to get the left subtree of (**Samples: 2 Trees of Rank 1**)

7 VISUALGO UNION-FIND DISJOINT SETS Exploration Mode ▾



Samples
Initialize(N)
FindSet(i)
isSameSet(i,j)
UnionSet(i,j)

slow ——— fast

⏮ ⏪ ⏸ ⏩ ⏭

About Team Terms of use

VisuAlgo UFDS Exercise (2)

First, click “**Initialize(N)**”, enter **8**, then click “**Go**”

Do a sequence of union and/or find operations to get the left subtree of (**Samples: 2 Trees of Rank 3**)

7 VISUALGO UNION-FIND DISJOINT SETS Exploration Mode ▾

Samples
Initialize(N)
FindSet(i)
isSameSet(i,j)
UnionSet(i,j)

slow fast

About Team Terms of use