Data Structure

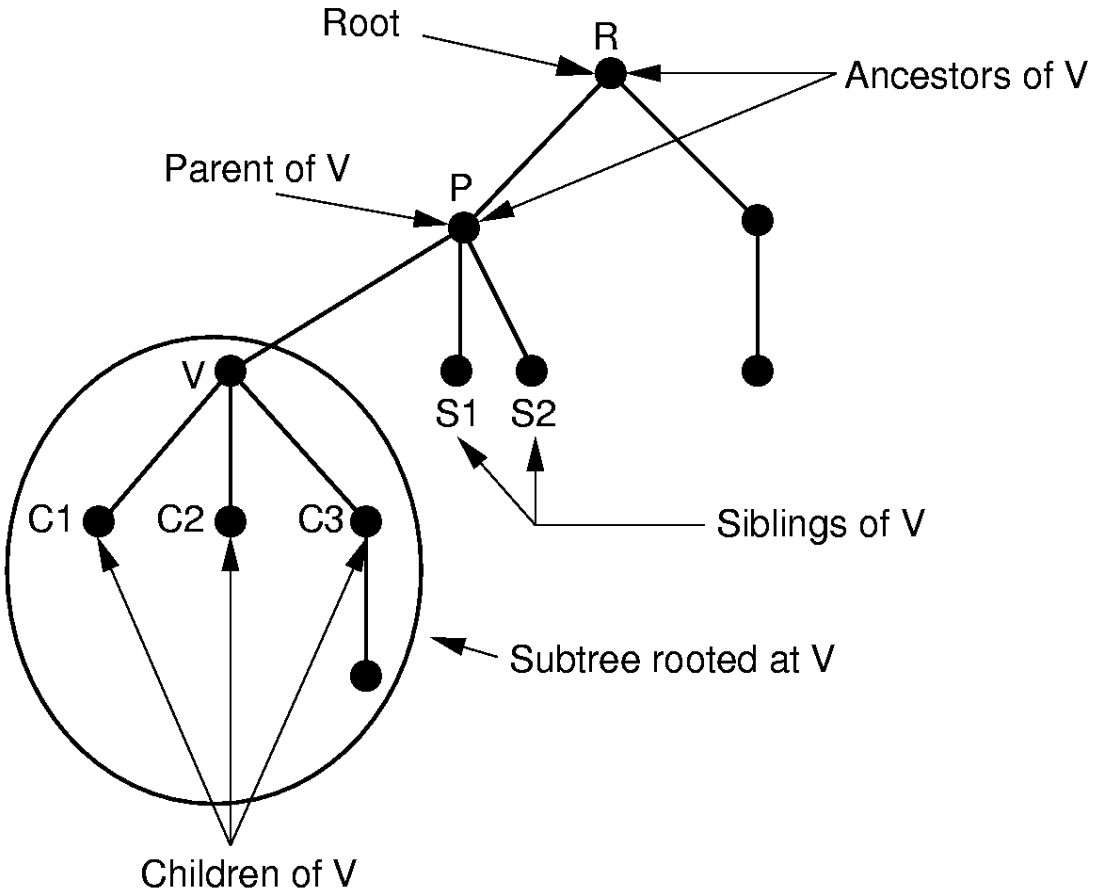
Lecture#14: Non-Binary Trees (Chapter 6)

U Kang
Seoul National University

In This Lecture

- The concept of the general tree, its ADT, and its operation
- Motivation and the main idea of the Union/Find operation
- Path compression for Union/Find

General Trees

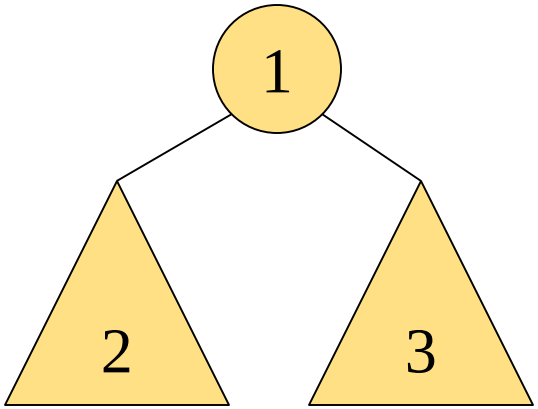


General Tree Node

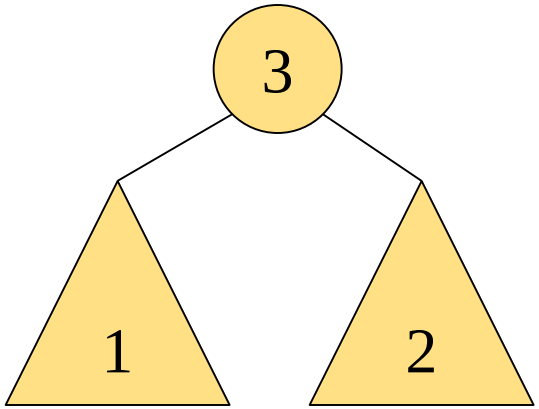
```
interface GTNode<E> {
   public E value();
   public boolean isLeaf();
   public GTNode<E> parent();
   public GTNode<E> leftmostChild();
   public GTNode<E> rightSibling();
   public void setValue(E value);
   public void setParent(GTNode<E> par);
   public void insertFirst(GTNode<E> n);
   public void insertNext(GTNode<E> n);
   public void removeFirst();
   public void removeNext();
}
```

General Tree Traversal

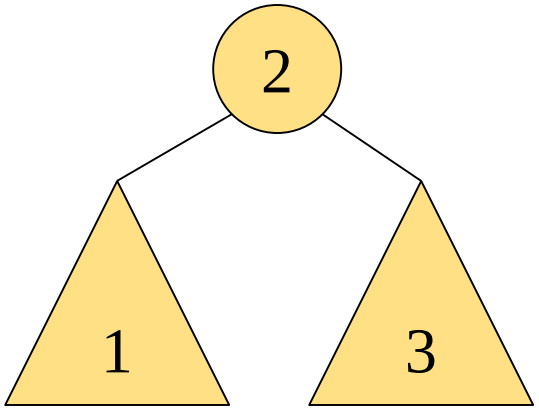
- Binary Tree



Preorder Traversal

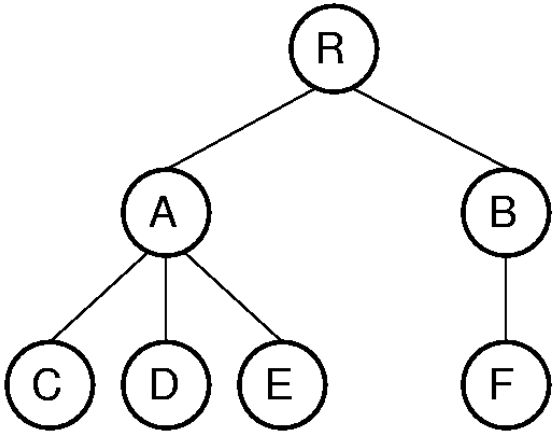


Postorder Traversal



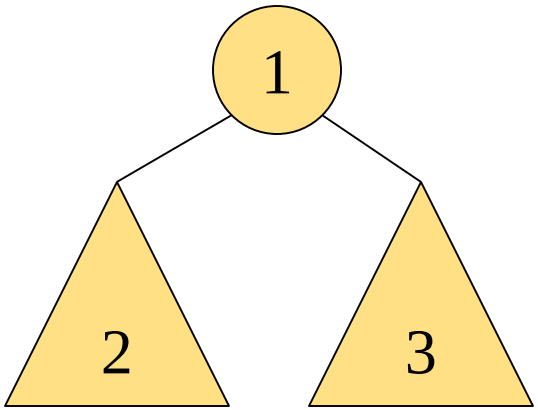
Inorder Traversal

- Non-binary Tree?

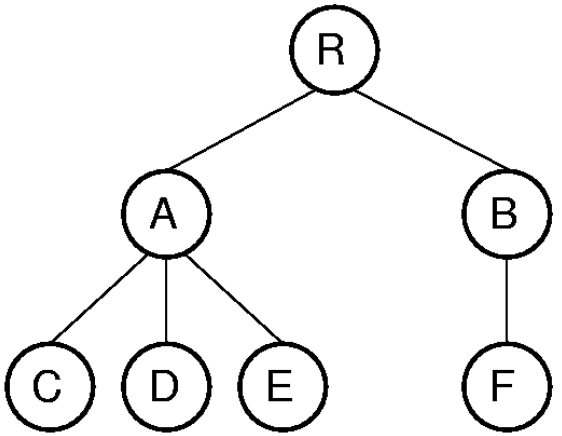


Preorder Traversal

```
/** Preorder traversal for general trees */
static <E> void preorder(GTNode<E> rt) {
  PrintNode(rt);
  if (!rt.isLeaf()) {
    GTNode<E> temp = rt.leftmostChild();
    while (temp != null) {
      preorder(temp);
      temp = temp.rightSibling();
    }
  }
}
```

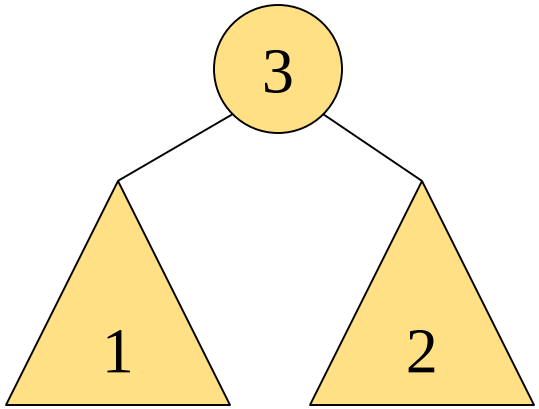


Preorder in binary tree

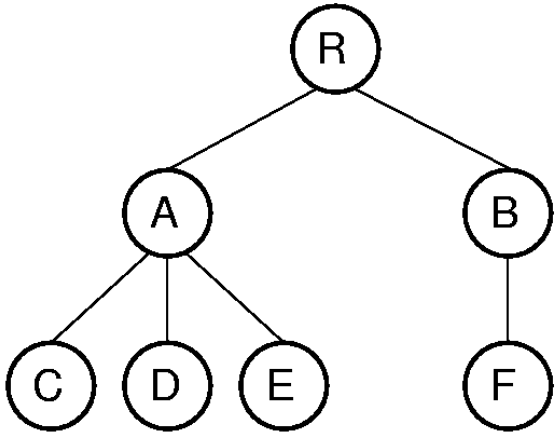


Well defined: R A C D E B F

Postorder Traversal

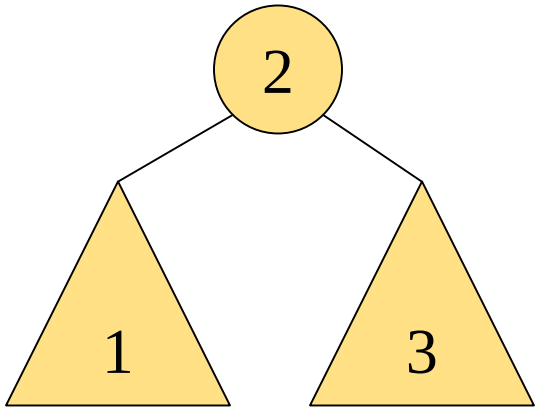


Postorder in binary tree

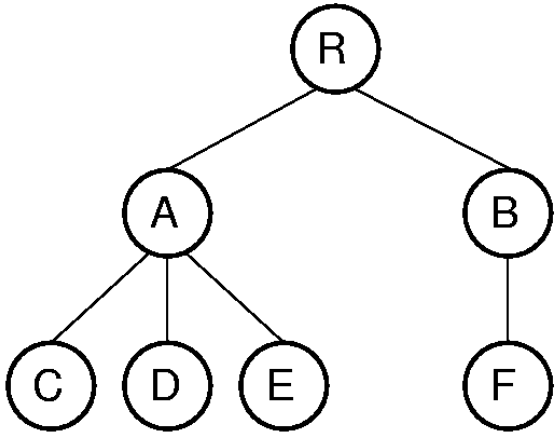


Well defined: C D E A F B R

Inorder Traversal

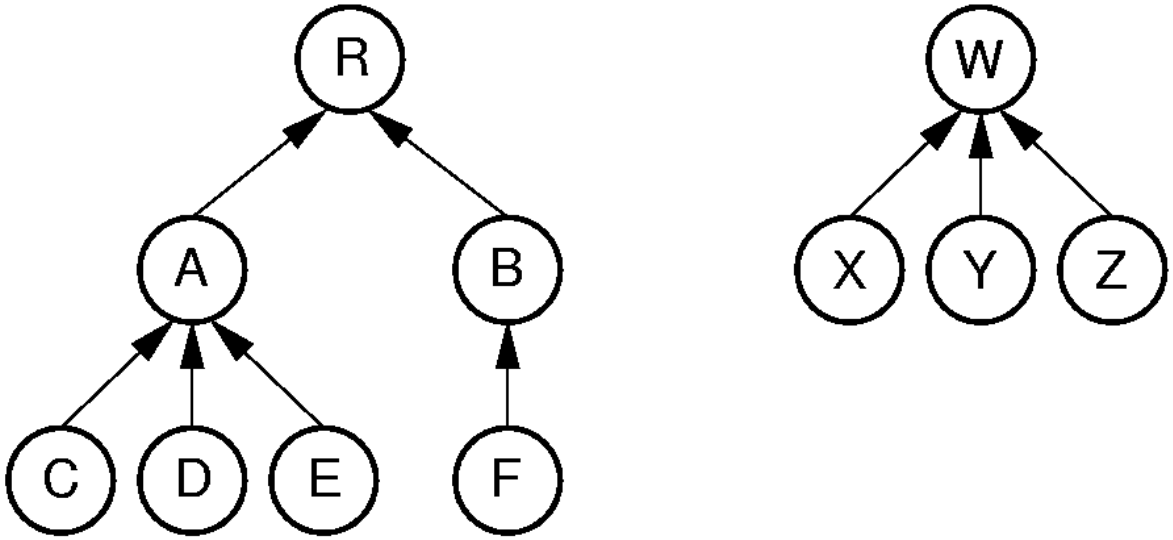


Inorder in binary tree



Well defined?

Parent Pointer Implementation



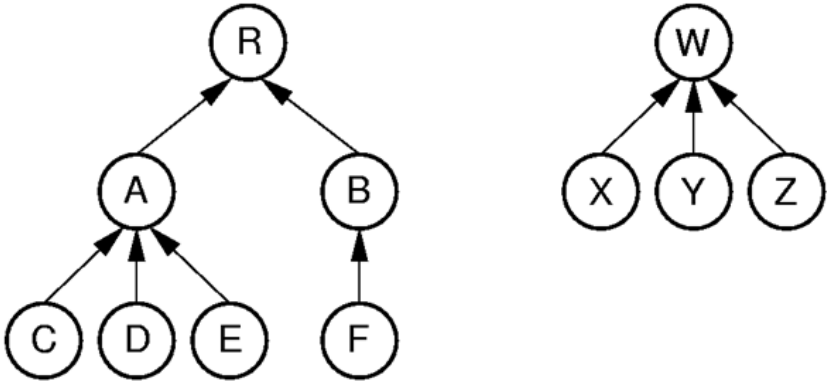
Node Index	0	1	2	3	4	5	6	7	8	9	10
Label	R	A	B	C	D	E	F	W	X	Y	Z
Parent's Index	/	0	0	1	1	1	2	/	7	7	7

Equivalence Class Problem

- The parent pointer representation is good for answering:
  - Are two elements in the same tree?

```
/** Determine if nodes in different trees */
public boolean differ(int a, int b) {
  Integer root1 = FIND(a);
  Integer root2 = FIND(b);
  return root1 != root2;
}
```

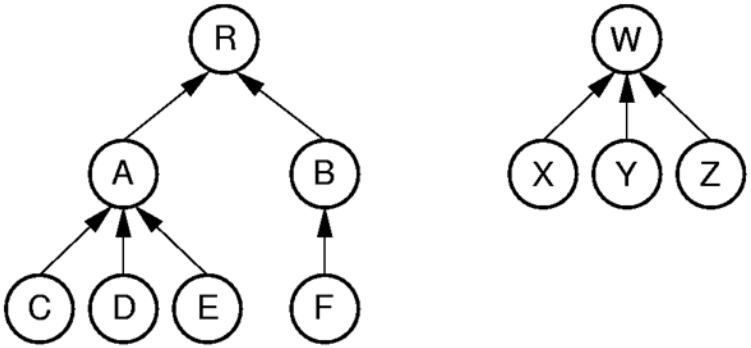
FIND function returns the root



Find

```
/** return the root of the curr's tree */
public Integer FIND(Integer curr) {
  if (array[curr] == null) return curr;
  while (array[curr] != null)
    curr = array[curr];
  return curr;
}

/** Determine if nodes in different trees */
public boolean differ(int a, int b) {
  Integer root1 = FIND(a);
  Integer root2 = FIND(b);
  return root1 != root2;
}
```



Union/Find

- Two important operations for equivalence class problem
  - UNION: merge two sets together
  - FIND: determine if two objects are in the same sets
- Applications
  - Graph connectivity
  - Clustering

Implementations for Union/Find

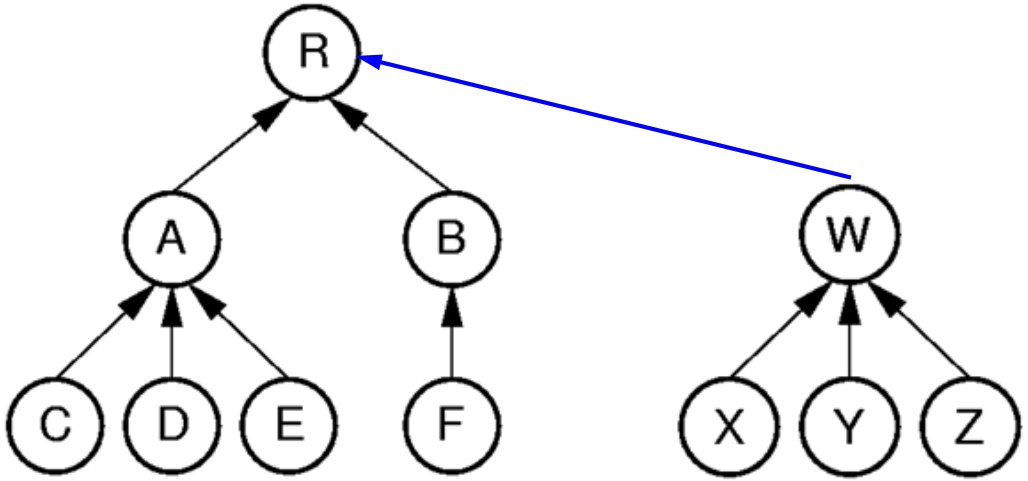
- UNION: merge two sets together
- FIND: determine if two objects are in the same sets; i.e., we want to get id(i) for an object i
- Naïve algorithm

  n: # of items in A ∪ B

  - UNION of A and B: update id(i) for all i in A ∪ B. Takes O(*n*)
  - FIND of i: O(1) using id(i)
- Using parent pointer implementation for tree
  - UNION of A and B: O(FIND(i)) = O(log *n*)
  - FIND of i: O(log *n*). If we use path compression, it can be O(1) for most i

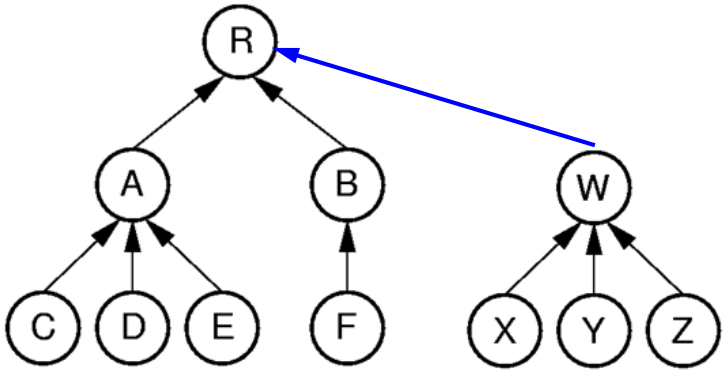
Union

```
/** Merge two subtrees */
public void UNION(int a, int b) {
  Integer root1 = FIND(a); // Find a's root
  Integer root2 = FIND(b); // Find b's root
  if (root1 != root2) array[root2] = root1;
}
```

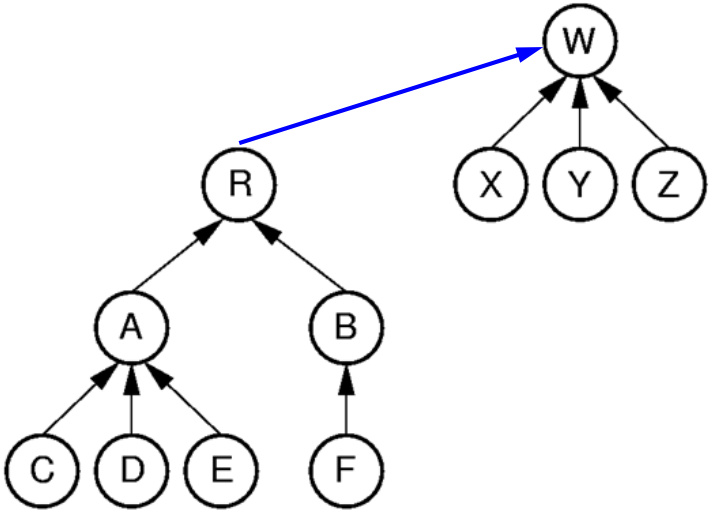


Union

- In Union, we want to keep the depth of the tree small (Why?)
- Between the following two union methods, which is better? Why?



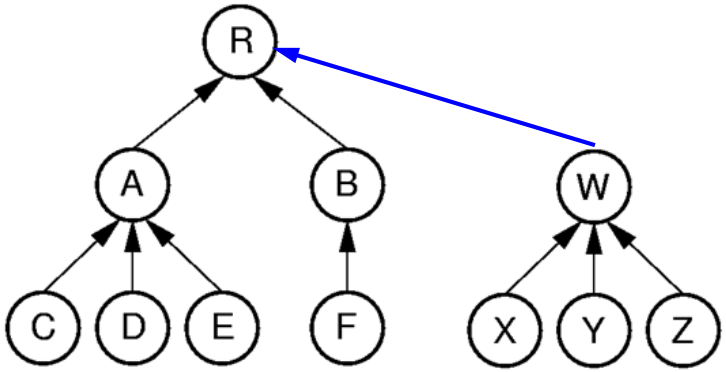
Method 1



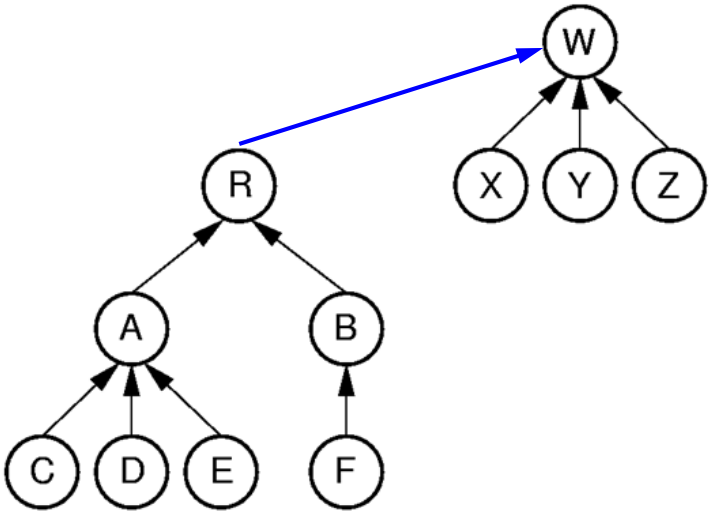
Method 2

Union

- Weighted union rule: join the tree with fewer nodes to the tree with more nodes.
  - Results in a tree with a small height



Method 1



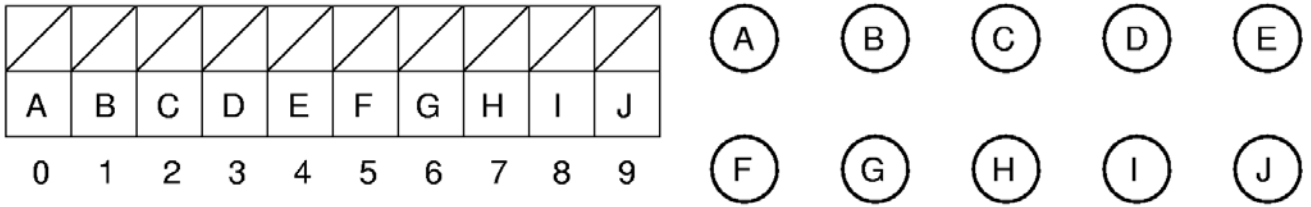
Method 2

Union

- (Theorem) Assume *n* nodes in *n* independent equivalent classes. Performing Union operations in any order results in a tree with depth at most log *n*.
- (Proof)
  - Consider a node v with the maximum depth in the final tree. Initially, v's depth was 0. v's depth increased only when the subtree A containing v is merged with another tree B, and |A| < |B|. Then,

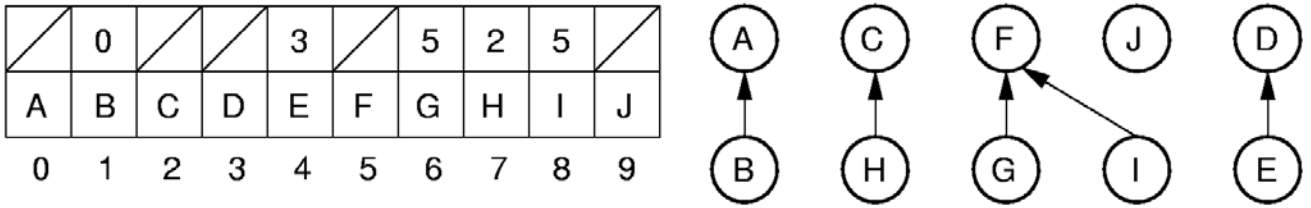
    v's final depth = (# of times v's depth increased) ≤ log *n*.

Equiv Class Processing (1)



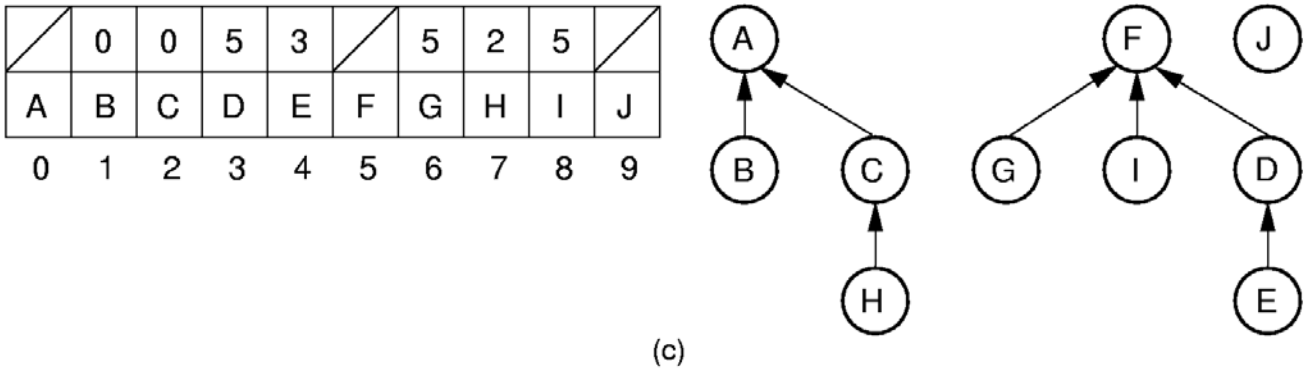
(a)

(Equivalence Relation)
(A,B) (C,H) (G,F) (D,E) (I,F)



(b)

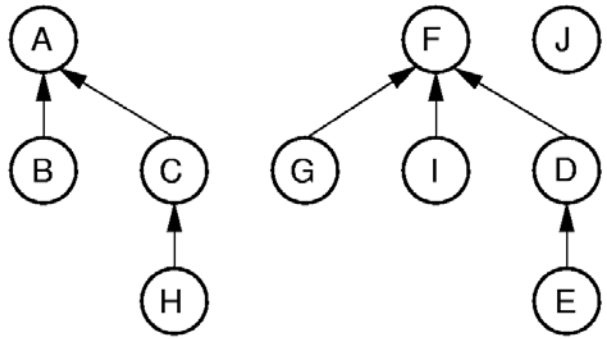
(Equivalence Relation)
(H,A) (E,G)



(c)

Equiv Class Processing (2)

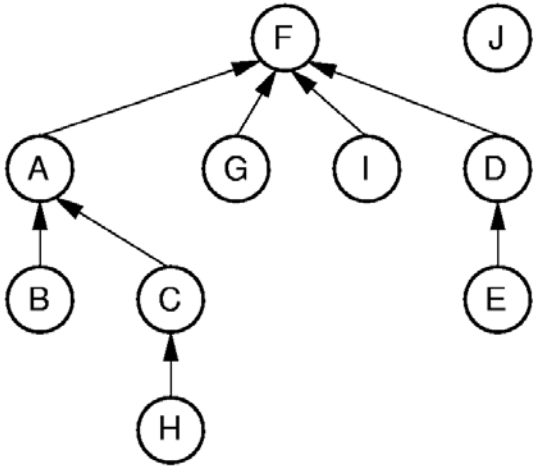
	0	0	5	3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



(c)

(Equivalence Relation)
(H,E)

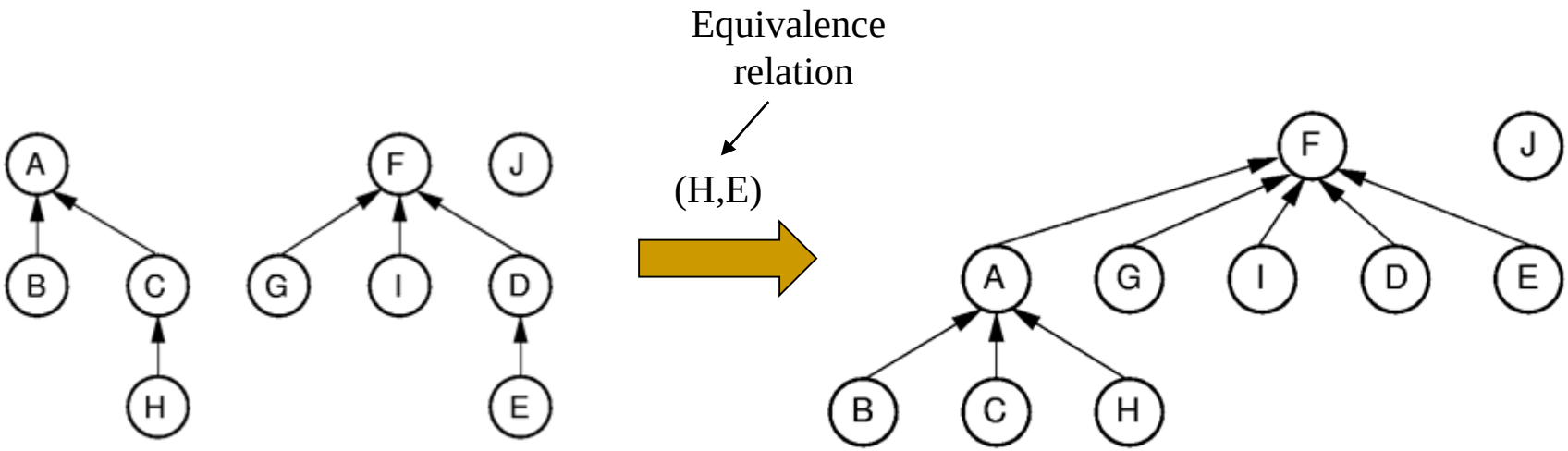
5	0	0	5	3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



(d)

Path Compression

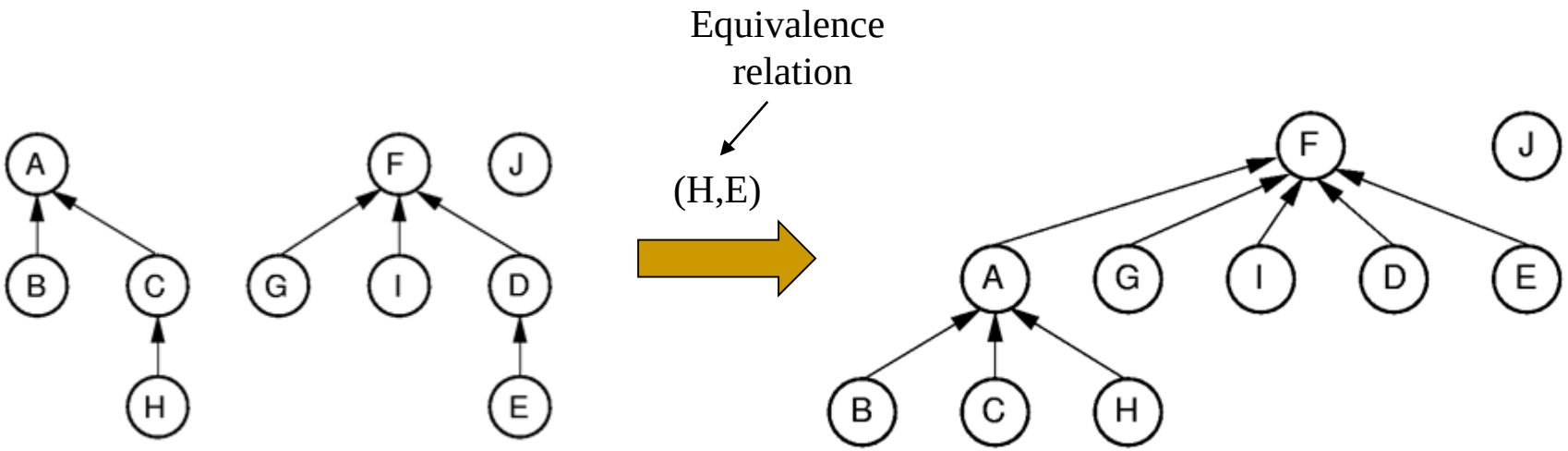
- How can we further decrease the max depth of the tree?
  - Path Compression: while doing FIND, make each node on the path to point to root



Path Compression

```
/** FIND for path compression */
public Integer FIND(Integer curr) {
  if (array[curr] == null) return curr;
  array[curr] = FIND(array[curr]);
  return array[curr];
}
```

Compare this with FIND in slide 11



What you need to know

- The ADT and operations of general tree
- Goal and implementation of the Union/Find operation
- Goal and implementation of Path compression for Union/Find

Questions?