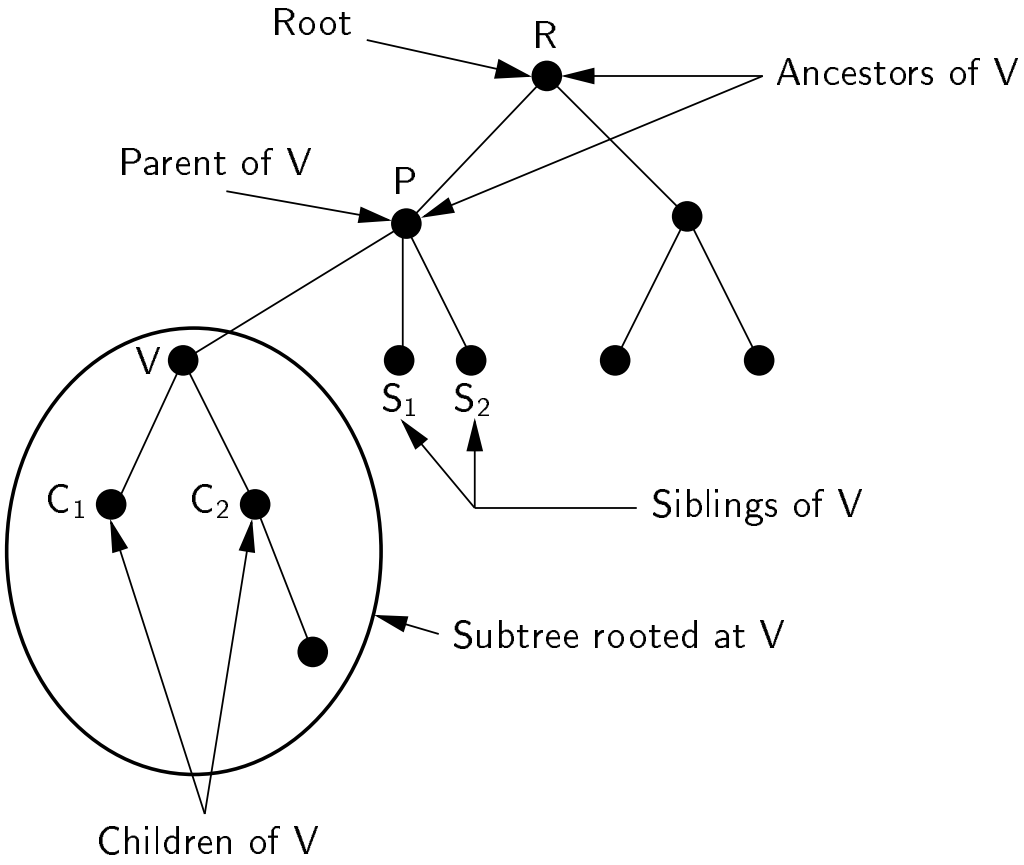
General Trees

A **tree** T is a finite set of one or more nodes such that there is one designated node r called the root of T, and the remaining nodes in $(T - \{r\})$ are partitioned into $n \geq 0$ disjoint subsets T_1, T_2, ..., T_k, each of which is a tree, and whose roots r_1, r_2, ..., r_k, respectively, are children of r.

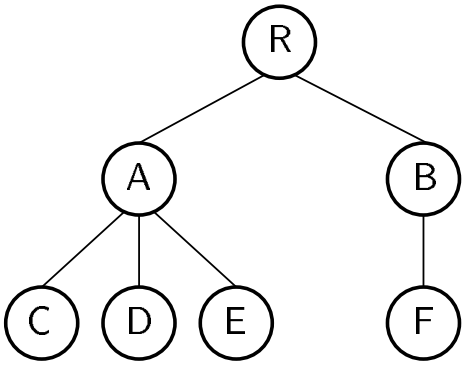


General Tree ADT

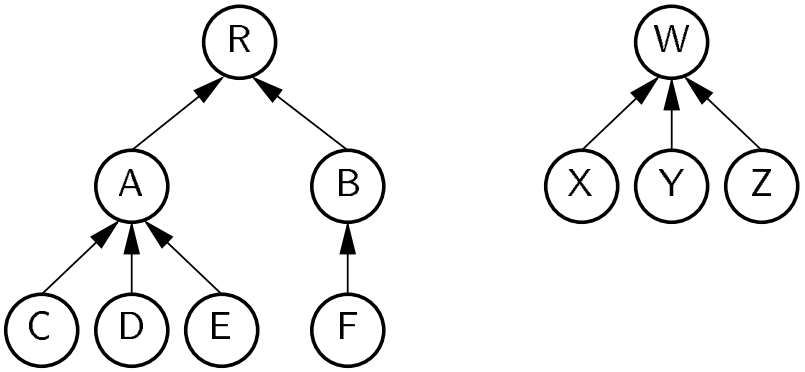
```
public interface GTNode {
  public Object value();
  public boolean isLeaf();
  public GTNode parent();
  public GTNode leftmost_child();
  public GTNode right_sibling();
  public void setValue(Object value);
  public void setParent(GTNode par);
  public void insert_first(GTNode n);
  public void insert_next(GTNode n);
  public void remove_first();
  public void remove_next();
}

public interface GenTree {
  public void clear();
  public GTNode root();
  public void newroot(Object value, GTNode first,
                                    GTNode sib);
}
```

General Tree Traversal

```
static void print(GTNode rt) { // Preorder traversal
  if (rt.isLeaf()) System.out.print("Leaf: ");
  else System.out.print("Internal: ");
  System.out.println(rt.value());
  GTNode temp = rt.leftmost_child();
  while (temp != null) {
    print(temp);
    temp = temp.right_sibling();
  }
}
```



Parent Pointer Implementation



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

Parent Pointer representation is good for answering:

Are two elements in the same tree?

```
public boolean differ(int a, int b) {
  GTNode root1 = FIND(array[a]); // Find root of a
  GTNode root2 = FIND(array[b]); // Find root of b
  return root1 != root2;         // Compare roots
}
```

Equivalence Classes

When joining equivalence classes, want to keep depth small.

Weighted Union Rule: join the tree with fewer nodes to the tree with more nodes.

Limits depth to $\log n$ for n nodes.

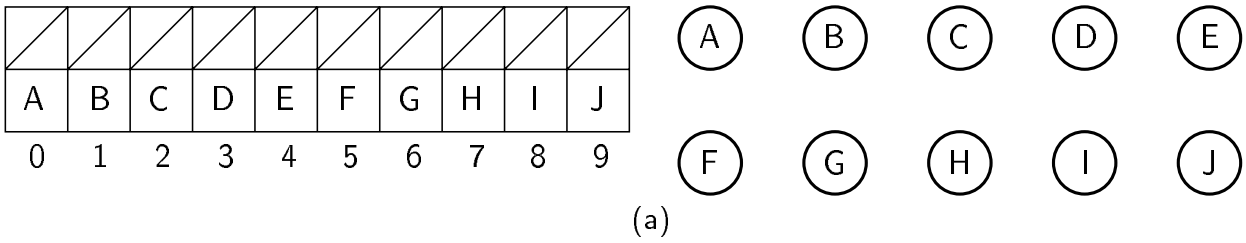
Path Compression: Make all nodes visited point to root.

```
class GTNode {   // General tree node for UNION/FIND
  private GTNode par;              // Parent pointer
  public GTNode() { par = null; } // Constuctor
  public GTNode parent() { return par; }
  public GTNode setParent(GTNode newpar)
  { return par = newpar; }
} // class GTNode
```

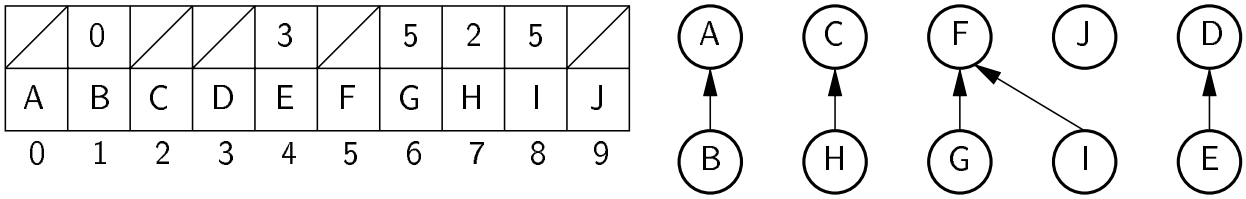
UNION-FIND

```
class GenTree {   // General Tree class for UNION/FIND
  private GTNode[] array;     // Node array

  public GenTree(int size) {  // Constructor
    array = new GTNode[size]; // Create node array
    for (int i=0; i<size; i++)
      array[i] = new GTNode();
  }

  public boolean differ(int a, int b) {
    GTNode root1 = FIND(array[a]);    // Find root of a
    GTNode root2 = FIND(array[b]);    // Find root of b
    return root1 != root2;            // Compare roots
  }

  public void UNION(int a, int b) {   // Merge subtrees
    GTNode root1 = FIND(array[a]);    // Find root of a
    GTNode root2 = FIND(array[b]);    // Find root of b
    if (root1 != root2) root2.setParent(root1);
  }

  private GTNode FIND(GTNode curr) {    // Find root
    while (curr.parent() != null) curr = curr.parent();
    return curr;                        // At root
  }
}
```

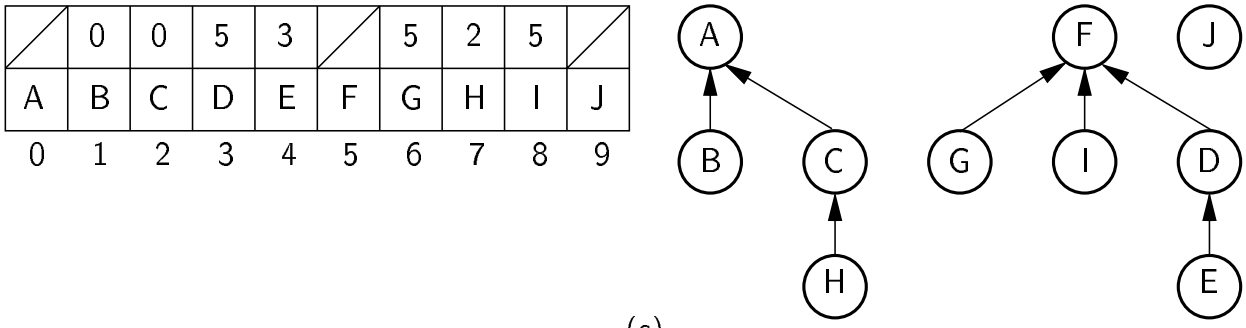
Equivalence Processing Example



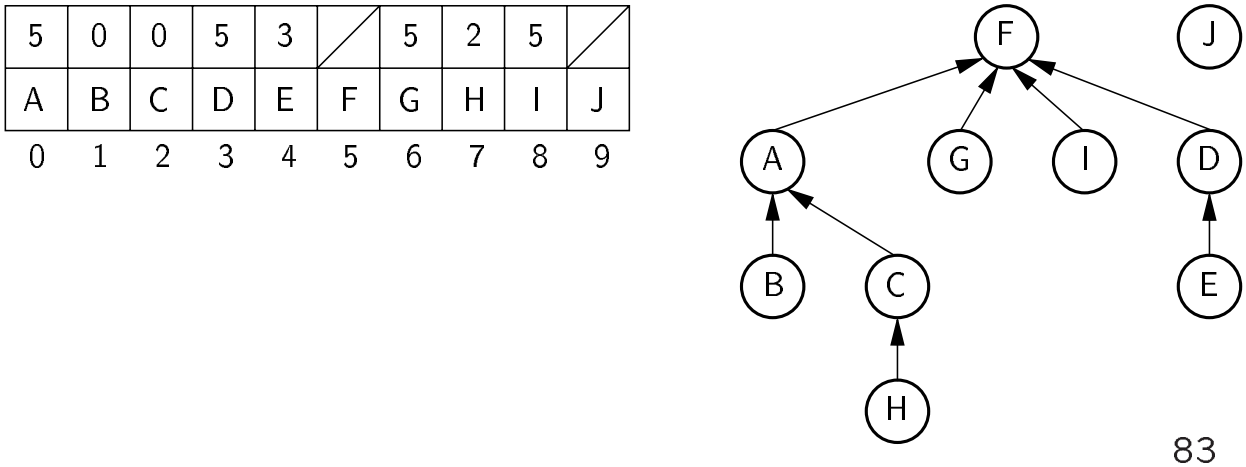
(a)



(b)

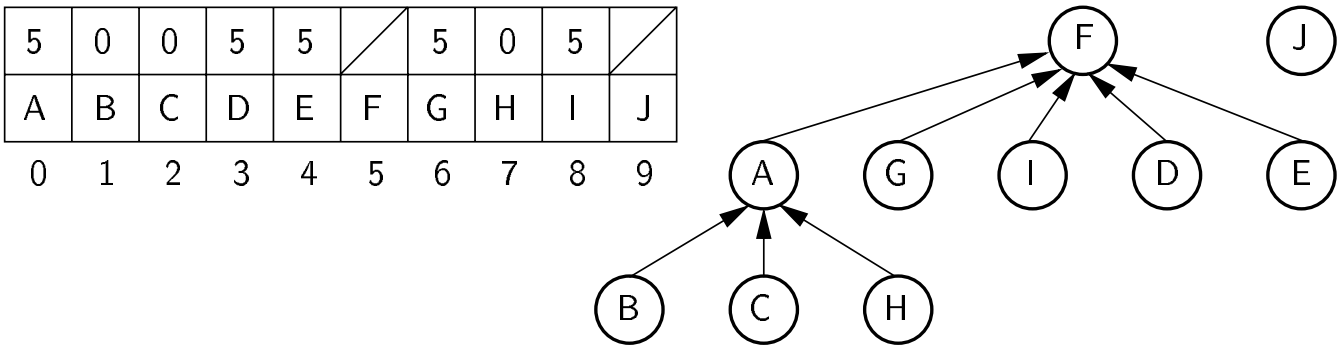


(c)



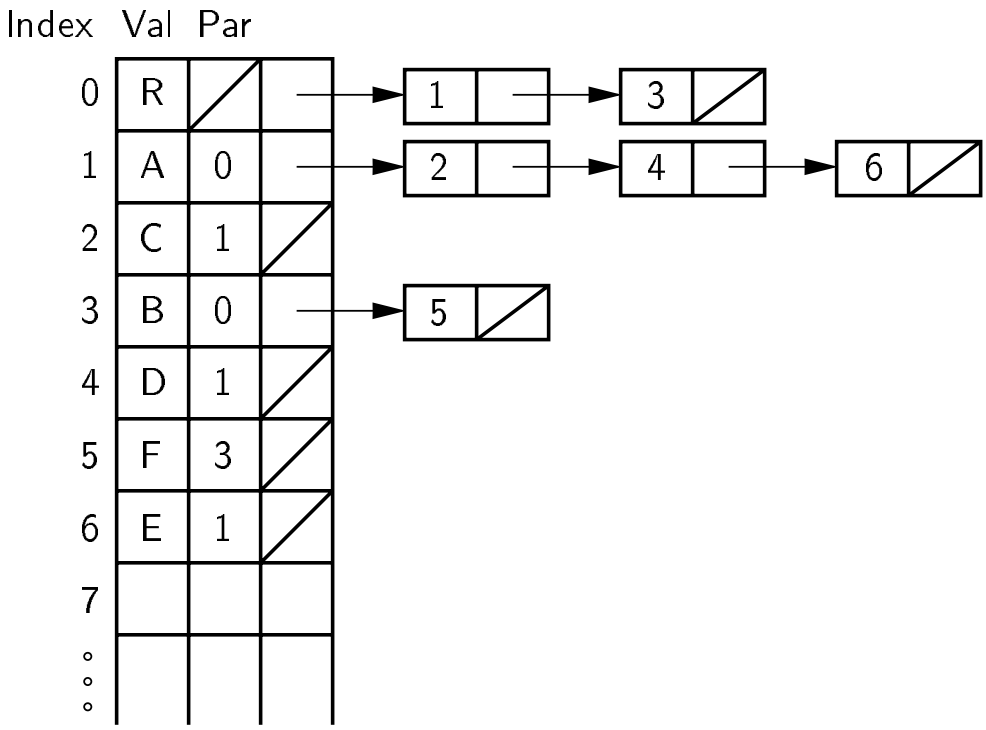
(d)

Path Compression Example

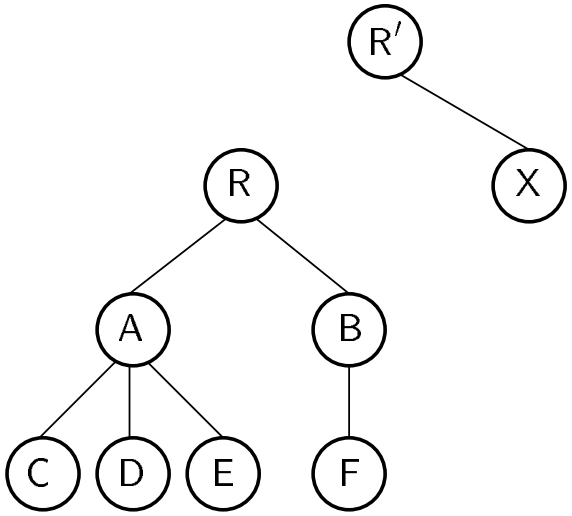



```
GTNode FIND(GTNode curr) {
  if (curr.parent() == null) return curr;  // At root
    return curr.setParent(FIND(curr.parent()));
}
```

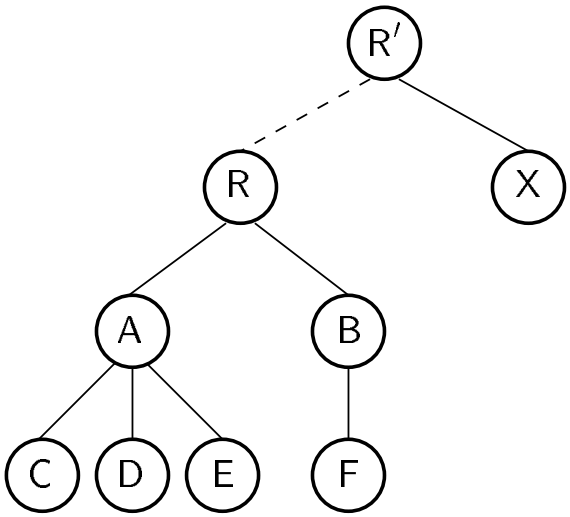
Lists of Children



Leftmost Child/Right Sibling

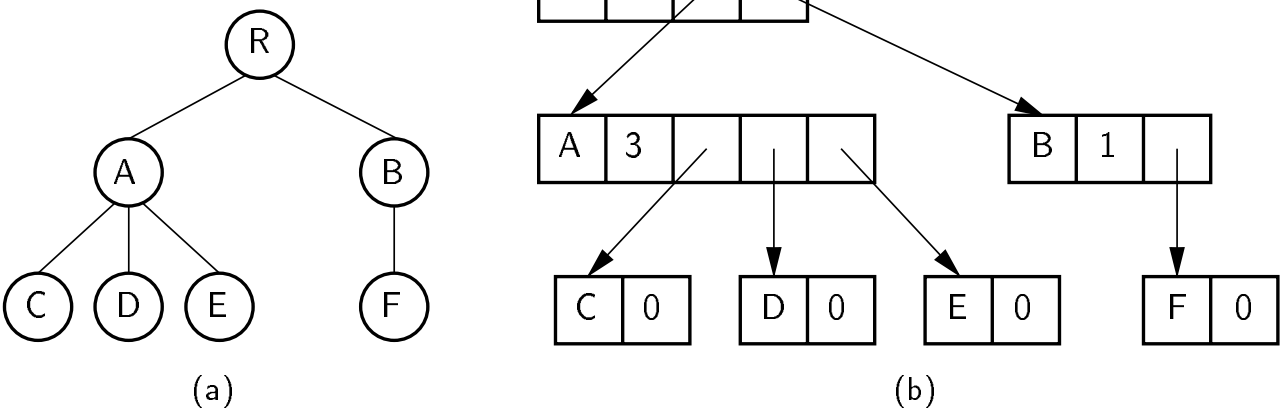


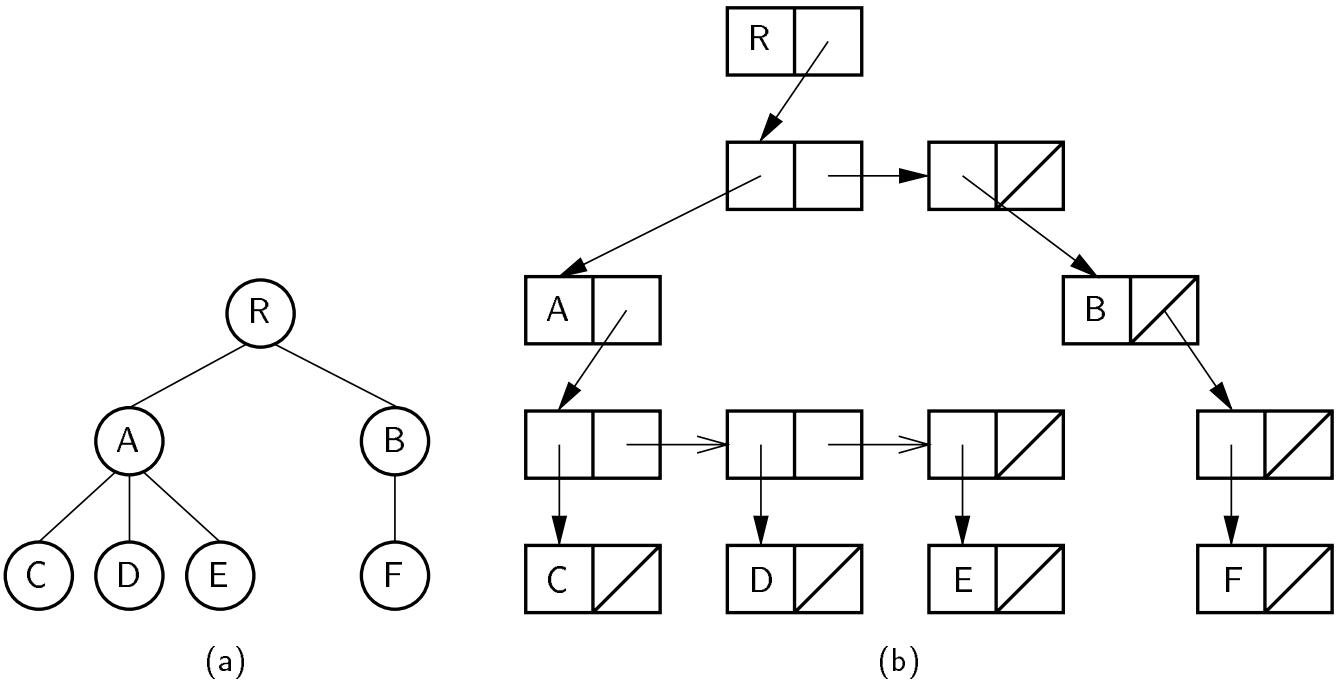
	Left	Val	Par	Right
1	R			
3	A	0	2	
6	B	0		
	C	1	4	
	D	1	5	
	E	1		
	F	2		
8	R′			
	X	7		



	Left	Val	Par	Right
1	R	(7)	(8)	
3	A	0	2	
6	B	0		
	C	1	4	
	D	1	5	
	E	1		
	F	2		
(0)	R′	-1		
	X	7		

Linked Implementations





Sequential Implementations

List node values in the order they would be visited by a preorder traversal.

Saves space, but allows only sequential access.

Need to retain tree structure for reconstruction.

For binary trees: Use symbol to mark `NULL` links.

$$AB/D//CEG///FH//I//$$

Full binary trees: Mark leaf or internal.

$$A'B'/DC'E'G/F'HI$$

General trees: Mark end of each subtree.

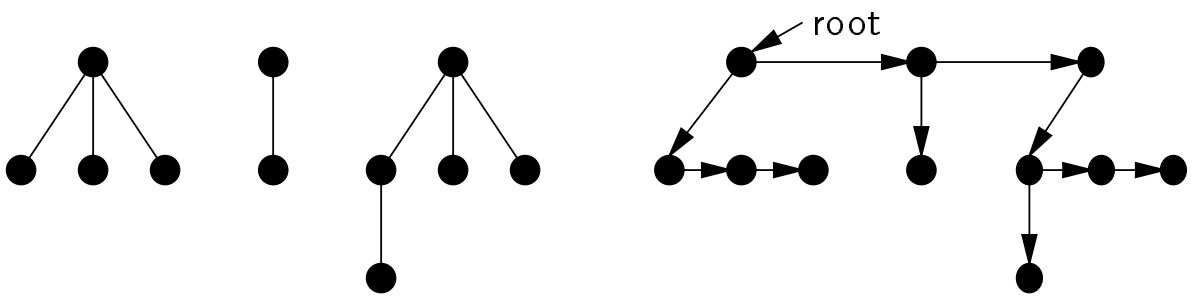
$$RAC)D)E))BF)))$$

Convert to Binary Tree

Left Child/Right Sibling representation essentially stores a binary tree.

Use this process to convert any general tree to a binary tree.

A **forest** is a collection of one or more general trees.



(a) (b)