

Data Structures and Algorithms

CS245-2012S-09

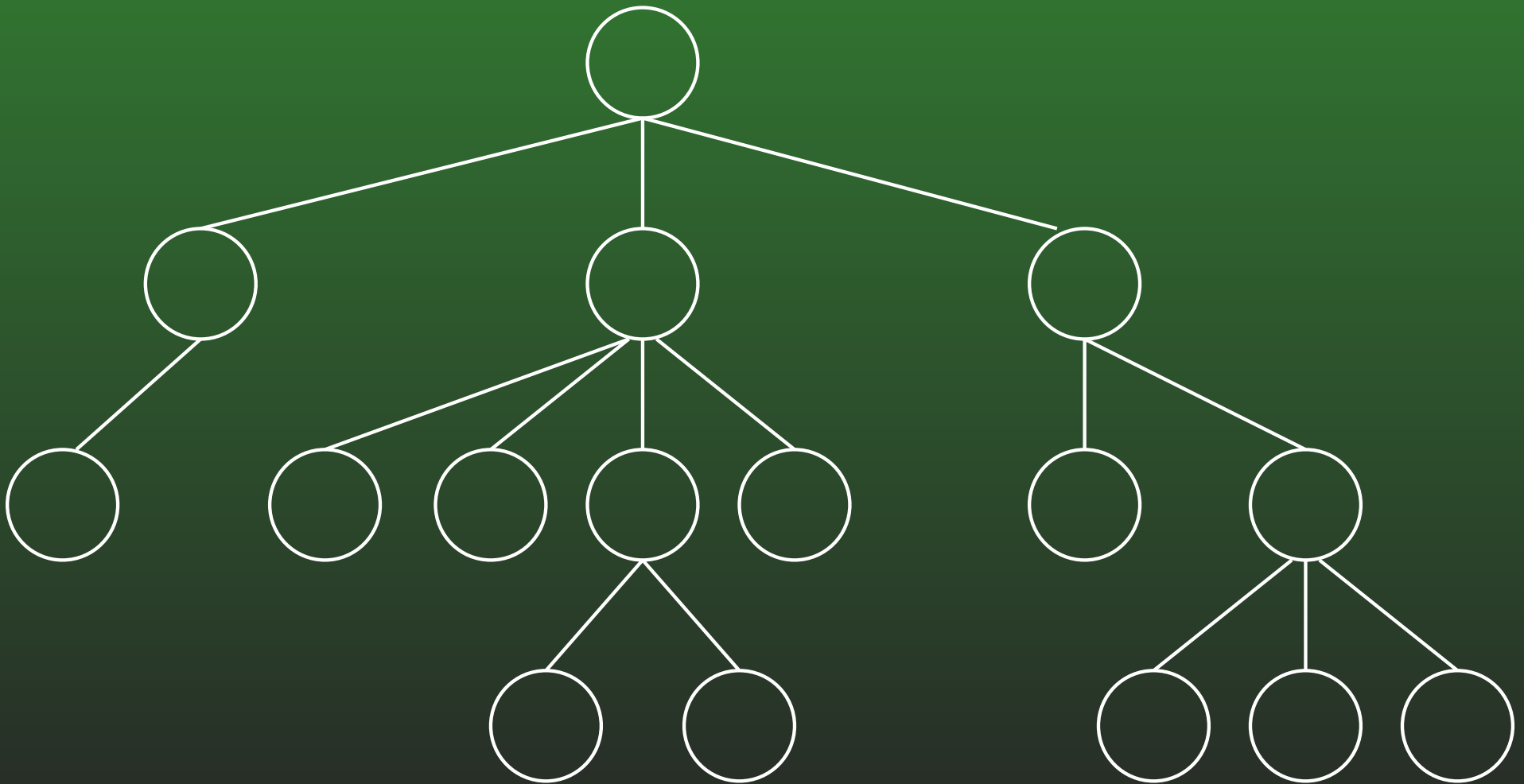
General Trees

David Galles

Department of Computer Science
University of San Francisco

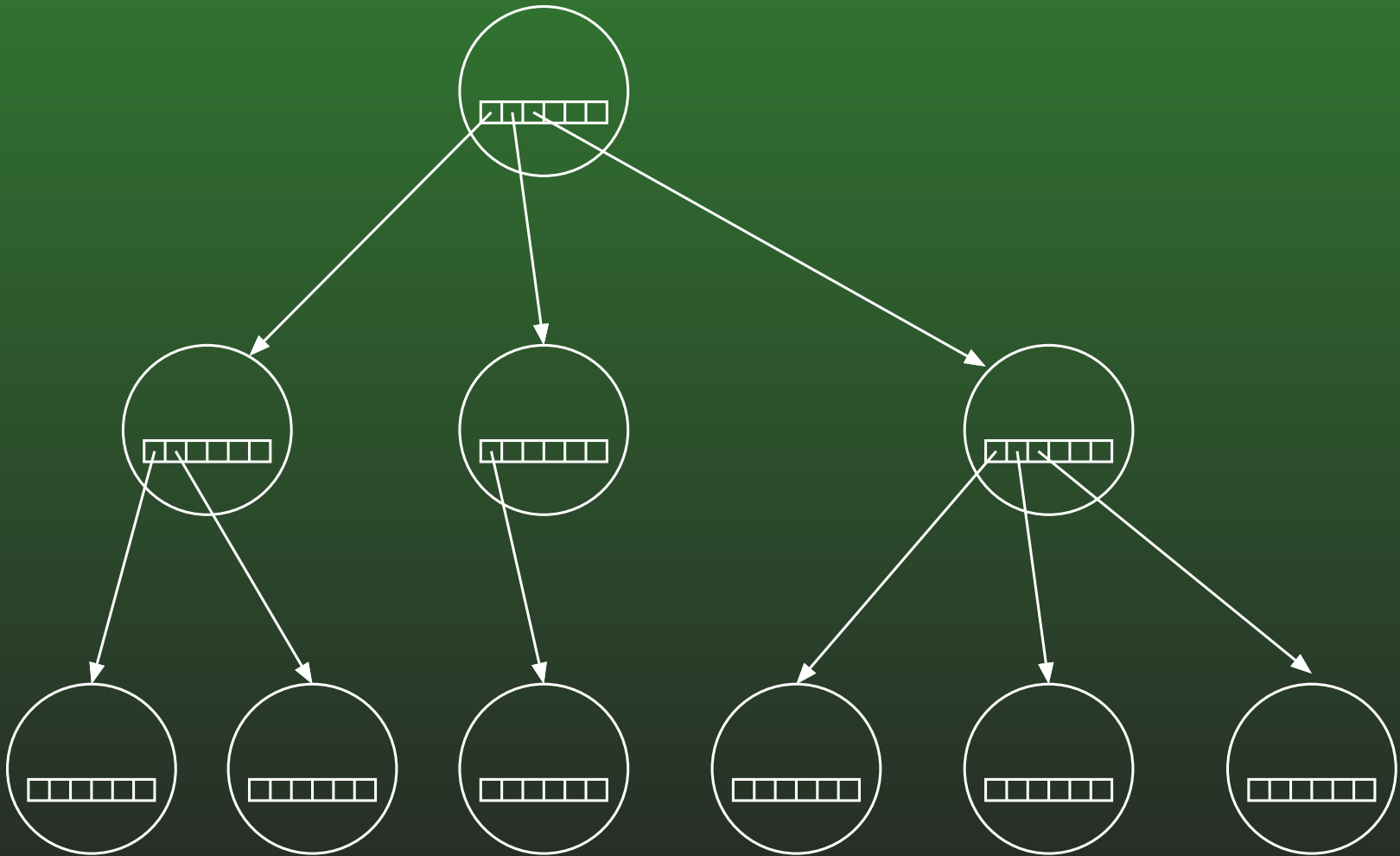
09-0: Trees with > 2 children

How can we implement trees with nodes that have > 2 children?



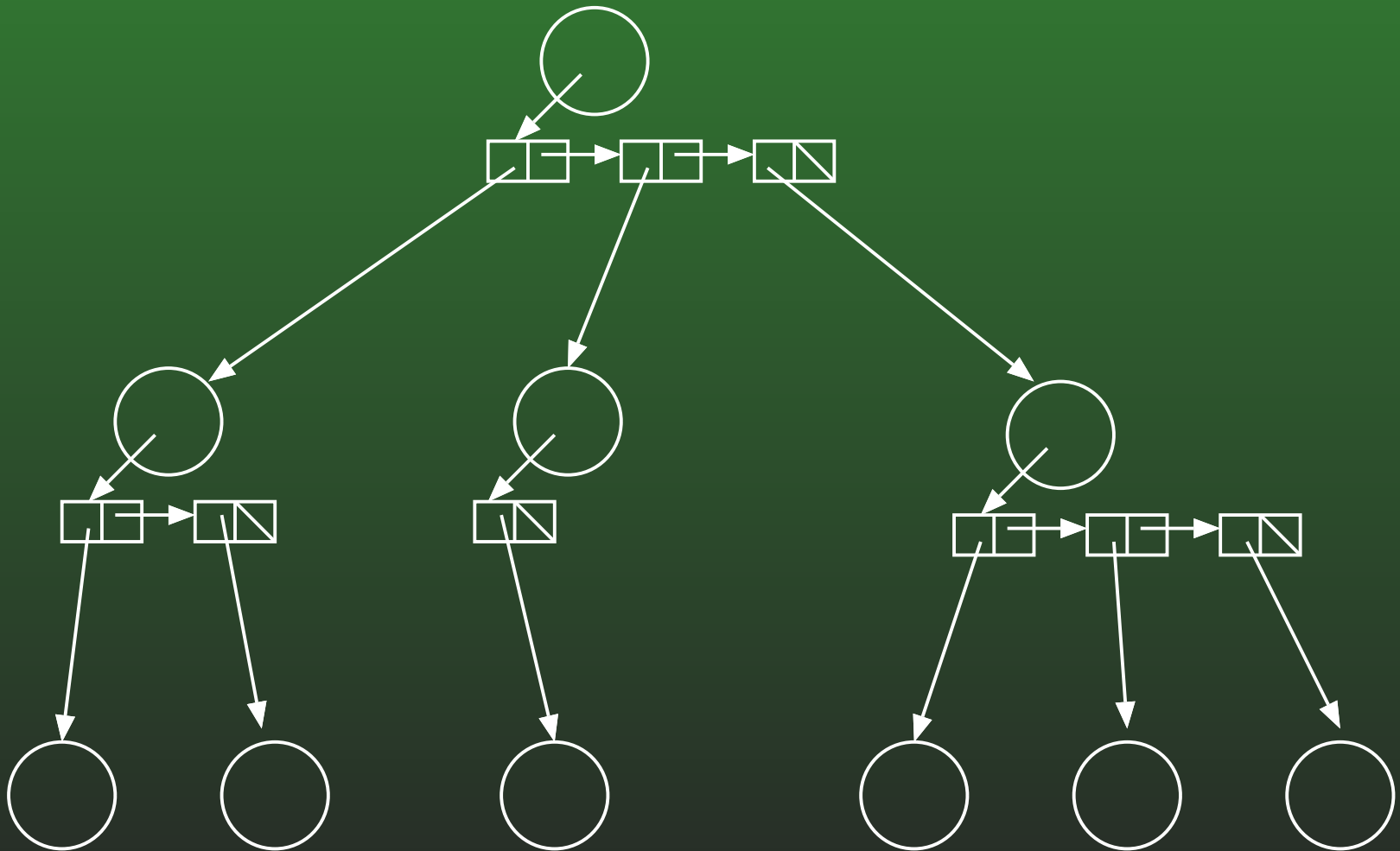
09-1: Trees with > 2 children

- Array of Children



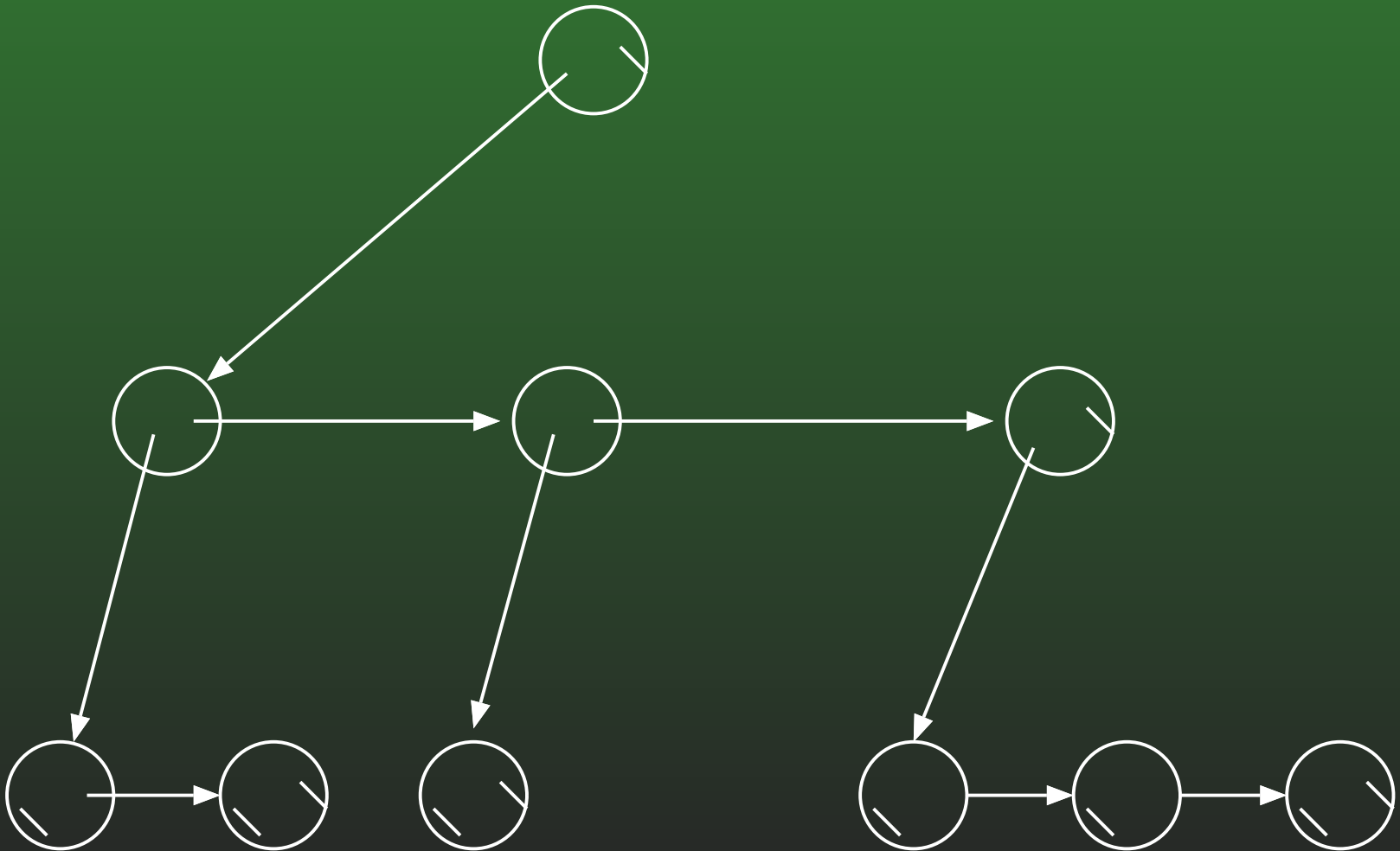
09-2: Trees with > 2 children

- Linked List of Children



09-3: Left Child / Right Sibling

- We can integrate the linked lists with the nodes themselves:



09-4: Working with General Tree

```
class Node {
    private Node leftchild_;
    private Node rightsib_;
    private Object element_;

    Node leftchild() {
        return leftchild_;
    }

    Node rightsib() {
        return rightsib_;
    }

    Node element() {
        return element_;
    }

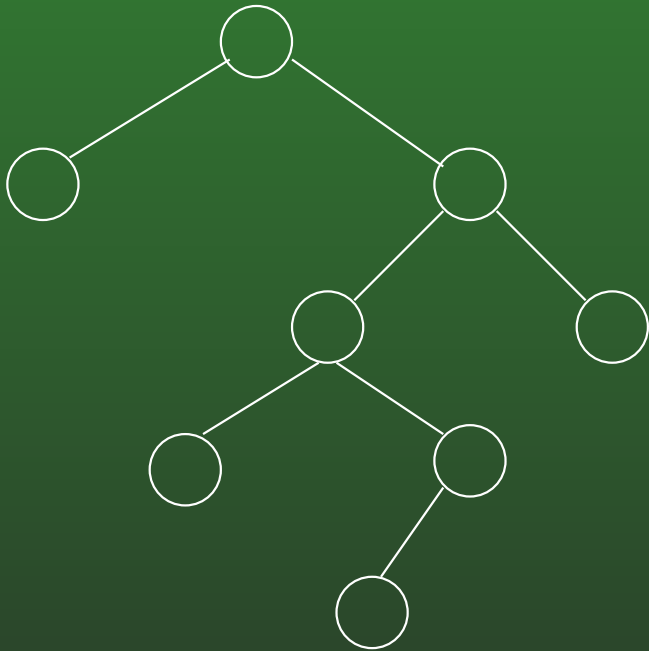
    void setLeftchild(Node leftchild) {
        leftchild_ = leftchild;
    }

    void setRightsib(Node leftchild) {
        rightsib_ = rightsib;
    }

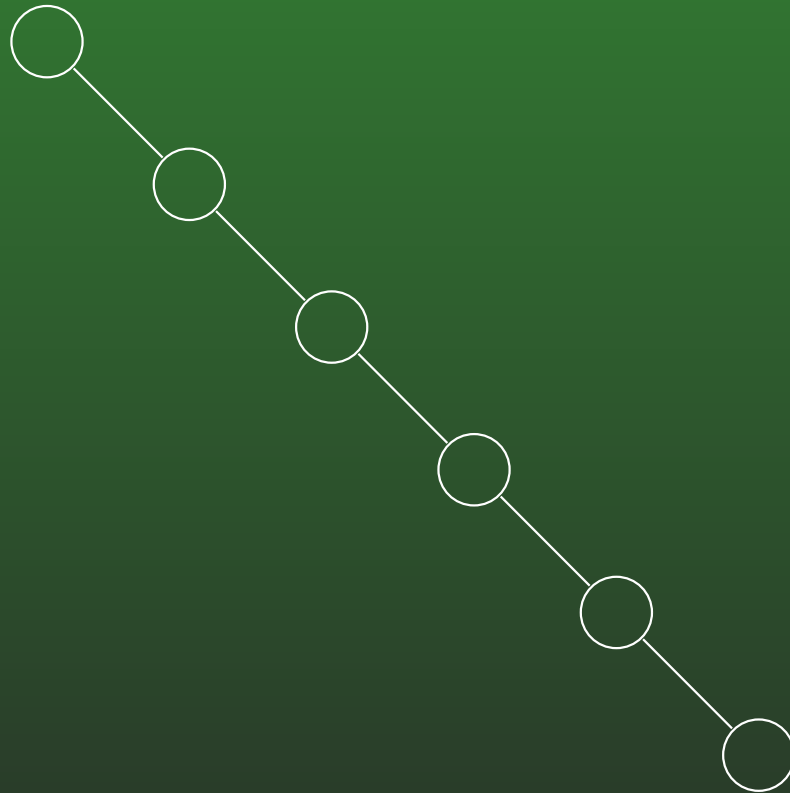
    void setElement(Object element) {
        element_ = element;
    }
}
```

09-5: General Trees – NumNodes

- Returns the number of nodes in a tree



Number of Nodes = 8



Number of Nodes = 6

09-6: General Trees – NumNodes

```
int numnodes(Node tree) {
    int descendants = 0;
    Node tmp;

    if (tree == null)
        return 0;
    for (tmp = tree.leftchild(); tmp != null;
         tmp = tmp.rightsib())
        descendants = descendants + numnodes(tmp);

    return descendants + 1;
}
```

09-7: General Trees – NumNodes II

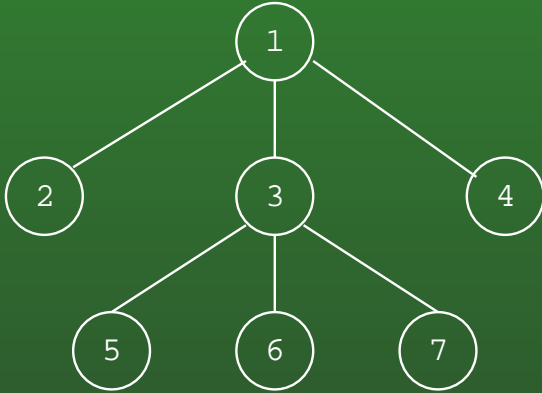
```
int numnodes(Node tree) {  
    if (tree == null)  
        return 0;  
    return 1 + numnodes(tree.leftchild())  
        + numnodes(tree.rightsib());  
}
```


09-9: General Trees – Height

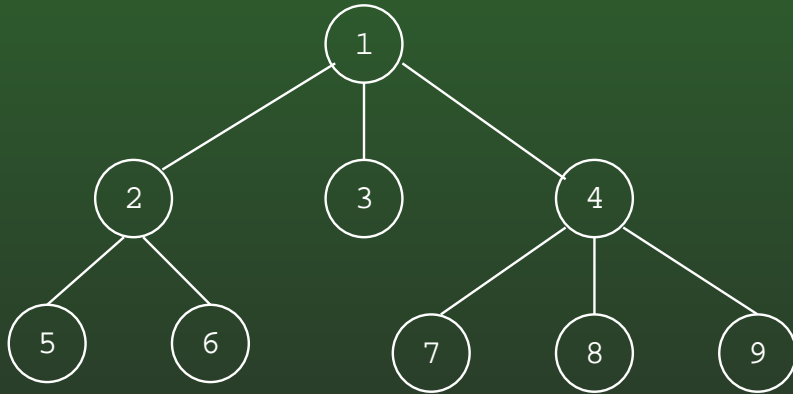
```
int height(Node tree) {
    if (tree == null)
        return 0;
    return MAX((1 + height(tree.leftchild())),
               height(tree.rightsib()));
}
```

09-10: General Trees

Tree 1



Tree 2



Tree 3

Write `numLeaves` and `print`

09-11: General Trees – numLeaves

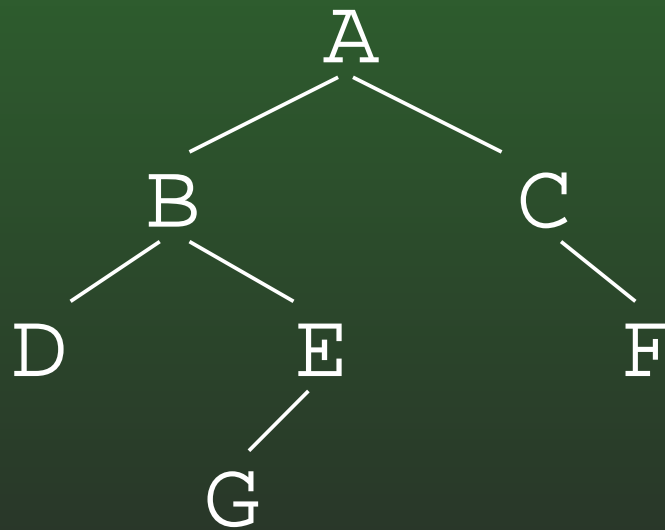
```
int numLeaves(Node tree) {  
  
    if (tree == null)  
        return 0;  
    if (tree.leftchild() == null)  
        return 1 + numLeaves(tree.rightsib());  
    return numLeaves(tree.leftchild()) +  
           numLeaves(tree.rightsib());  
}
```

09-12: General Trees – numLeaves

```
void print(Node tree, int offset) {  
  
    if (tree != null)  
    {  
        for (int i = 0; i < offset; i++)  
            System.out.print("\t");  
        System.out.println(tree.element());  
        print(tree.leftchild(), offset+1);  
        print(tree.rightsib(), offset);  
    }  
}
```

09-13: Serializing Binary Trees

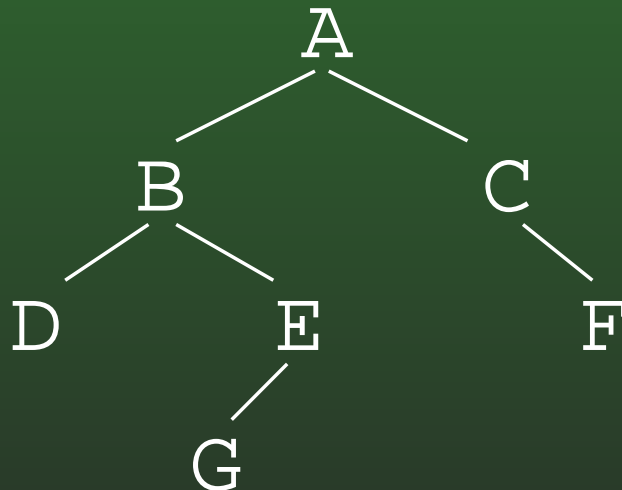
- Print a tree to a file, saving structure information
- First Try: Print out nodes, in order that they would appear in a PREORDER traversal.
 - Why doesn't this work?



ABDEGCF

09-14: Serializing Binary Trees

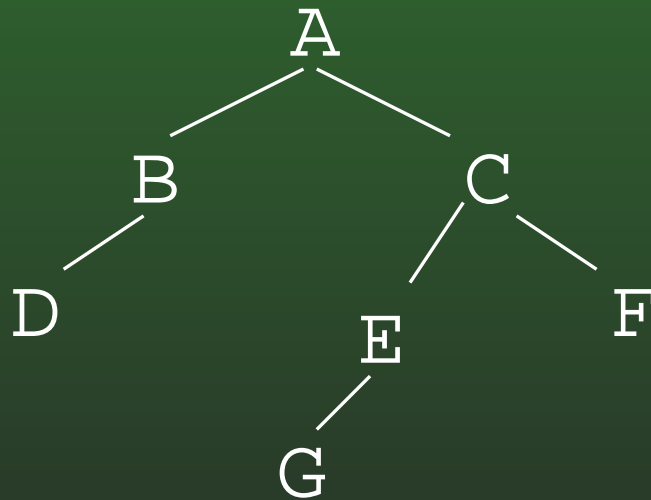
- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don't know when we've hit a null pointer
- Store null pointers, too!



ABD//EG///C/F//

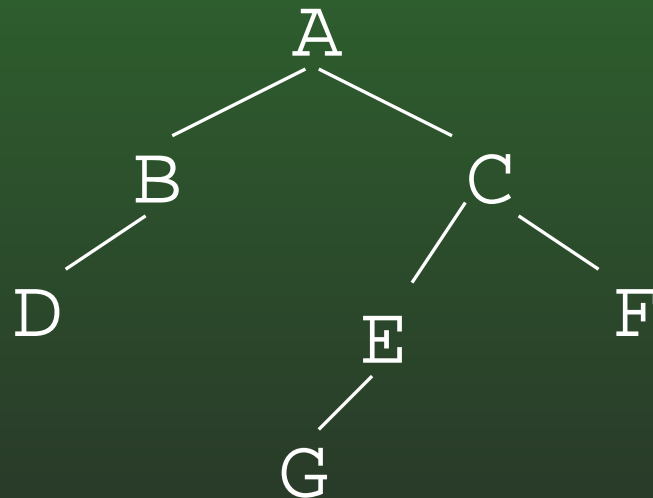
09-15: Serializing Binary Trees

- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don't know when we've hit a null pointer
- Store null pointers, too!



09-16: Serializing Binary Trees

- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don't know when we've hit a null pointer
- Store null pointers, too!



ABD///CEG///F//

09-17: Serializing Binary Trees

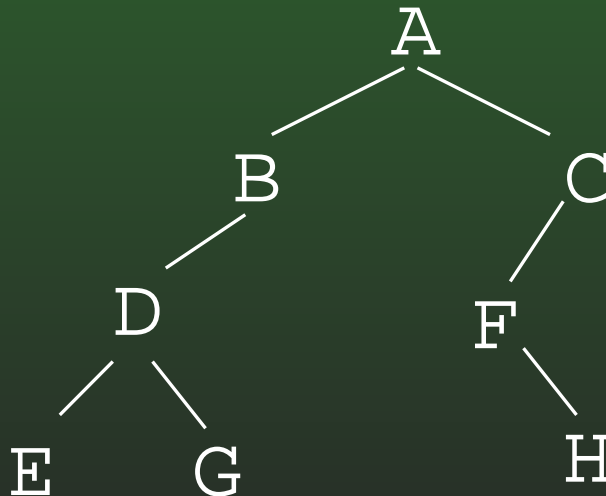
- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don't know when we've hit a null pointer
- Store null pointers, too!

ABDE//G///CF/H///

09-18: Serializing Binary Trees

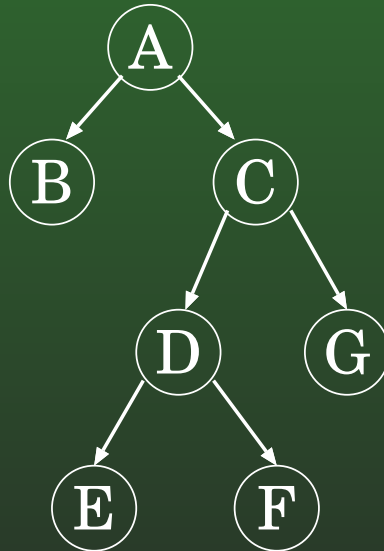
- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don't know when we've hit a null pointer
- Store null pointers, too!

ABDE//G///CF/H///



09-19: Serializing Binary Trees

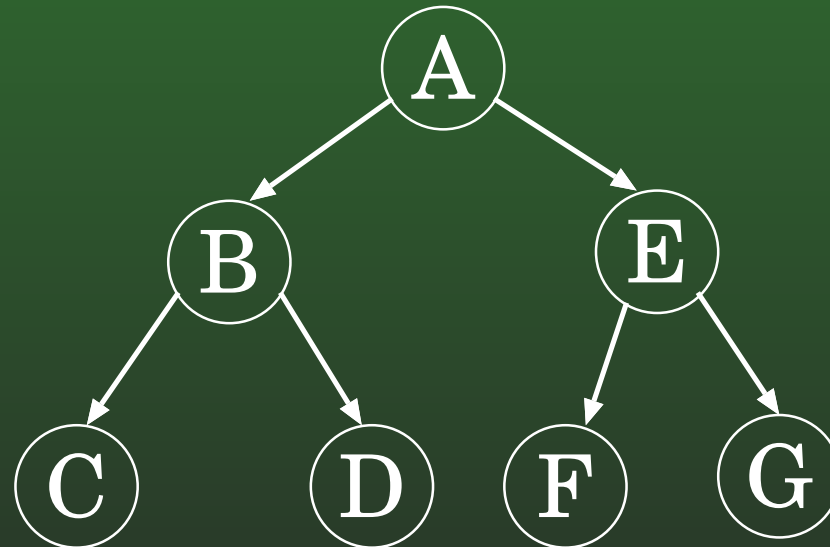
- If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:



$A_0 B_1 C_0 D_0 E_1 F_1 G_1$

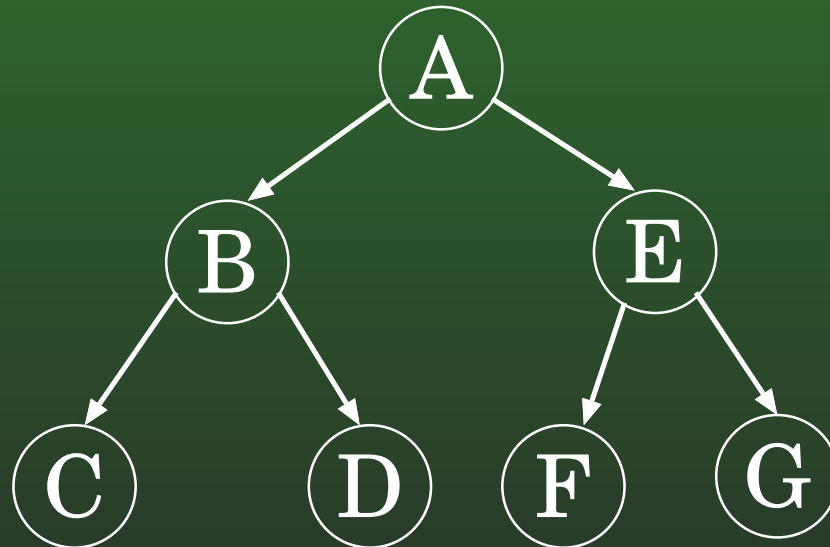
09-20: Serializing Binary Trees

- If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:



09-21: Serializing Binary Trees

- If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:



$A_0 B_0 C_1 D_1 E_0 F_1 G_1$

09-22: Serializing Binary Trees

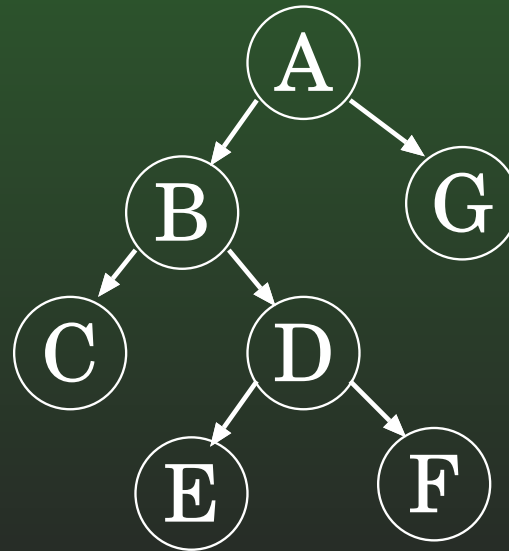
- If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:

$A_0 B_0 C_1 D_0 E_1 F_1 G_1$

09-23: Serializing Binary Trees

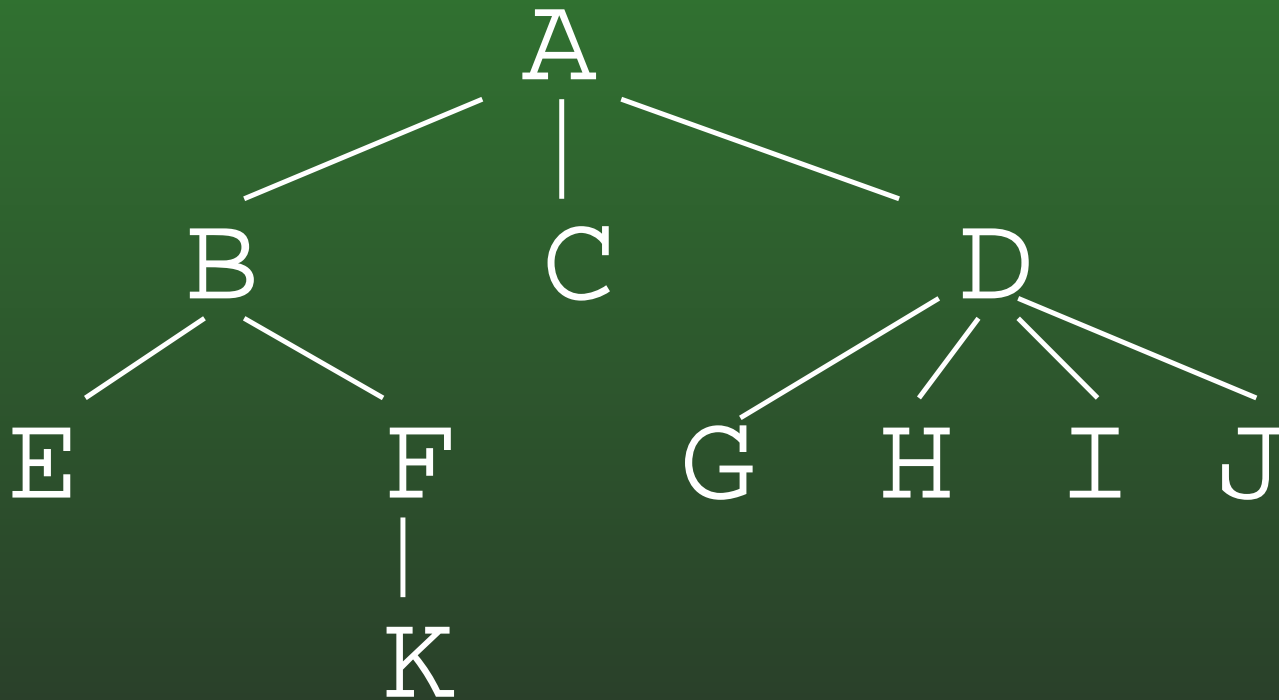
- If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:

$A_0 B_0 C_1 D_0 E_1 F_1 G_1$



09-24: Serializing General Trees

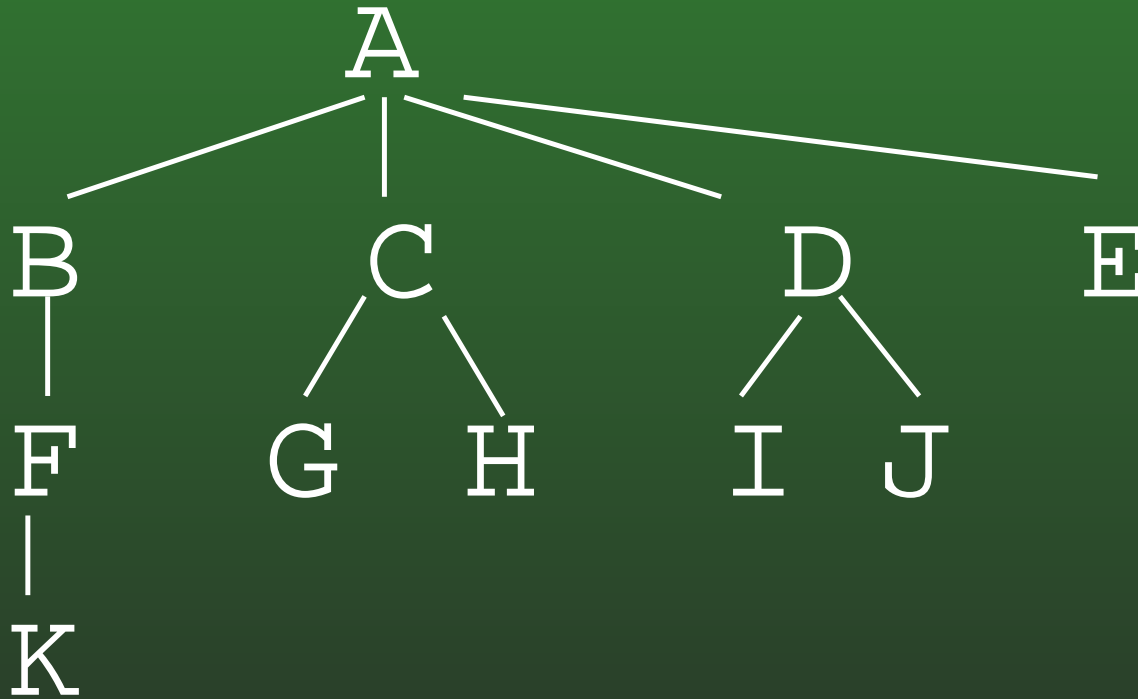
- Store an “end of children” marker



ABE)FK)))C)DG)H)I)J))))

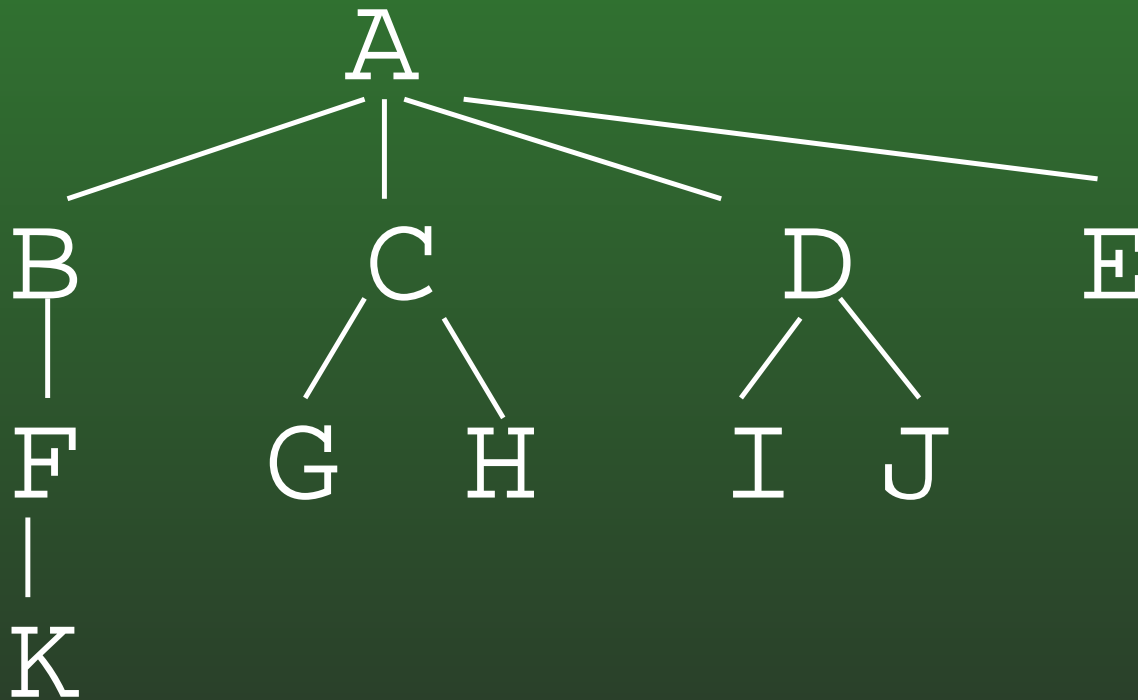
09-25: Serializing General Trees

- Store an “end of children” marker



09-26: Serializing General Trees

- Store an “end of children” marker



ABFK)))CG)H))DI)J))E))

09-27: Serializing General Trees

- Store an “end of children” marker

ABDK)))CE)F)GI)J))H)))

09-28: Serializing General Trees

- Store an “end of children” marker

ABDK)))CE)F)GI)J))H)))

