Lecture 10
Trees

Why Trees?

-trees enable a host of faster algorithms
-provide natural organization of the data

Trees are…

- hierarchical data structures
- appropriate for more search less update
- are made of nodes and edges

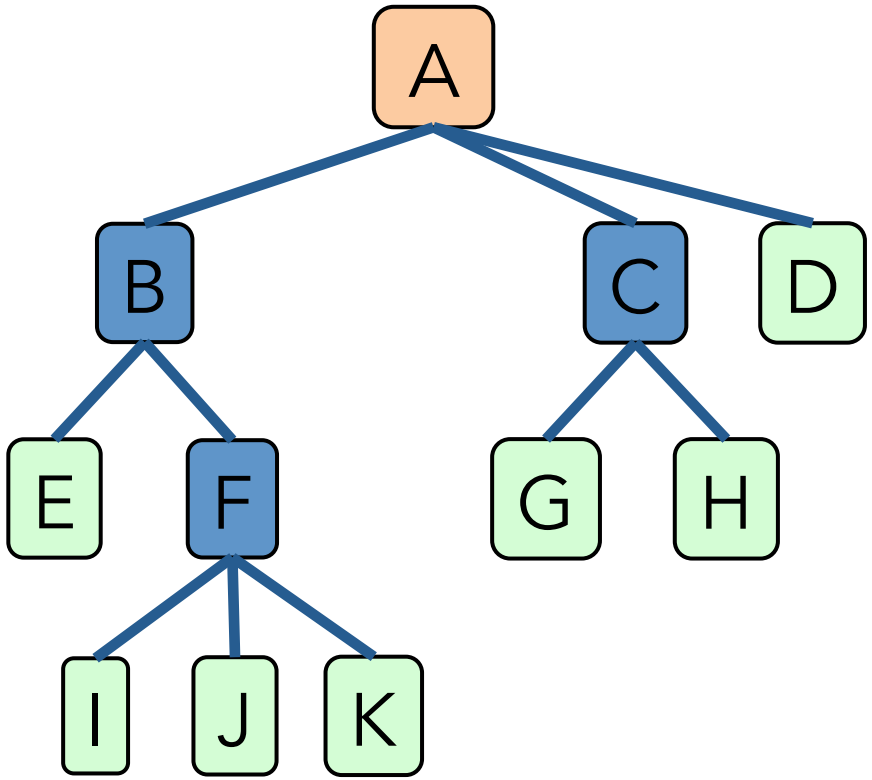
- a non-empty tree always has a root
- root has no parent
- each node has a unique parent

Nodes

Edges

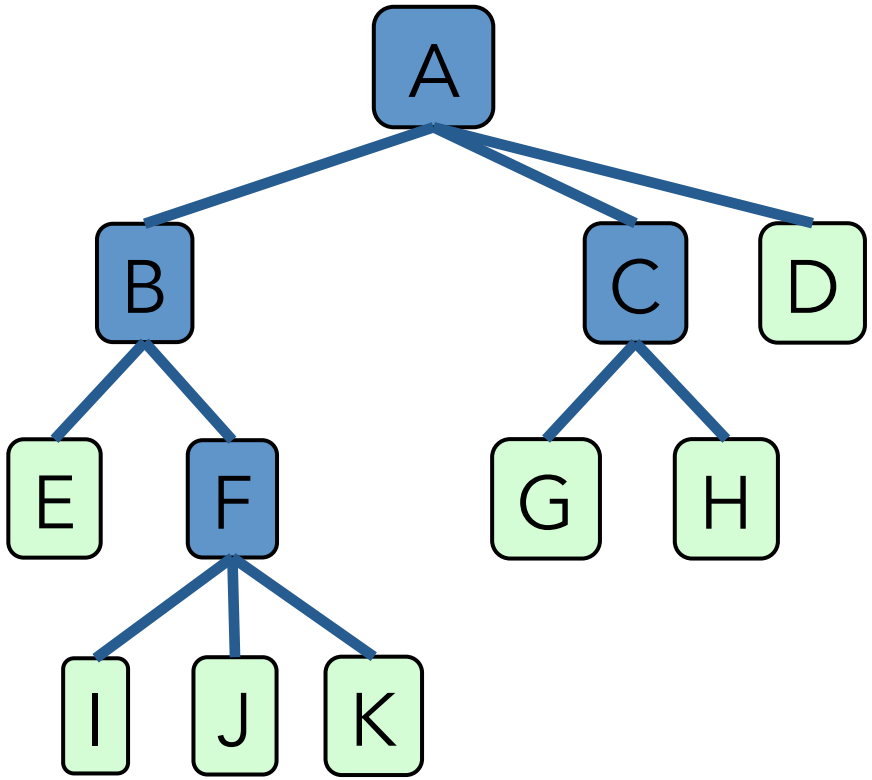
Root

Leaves



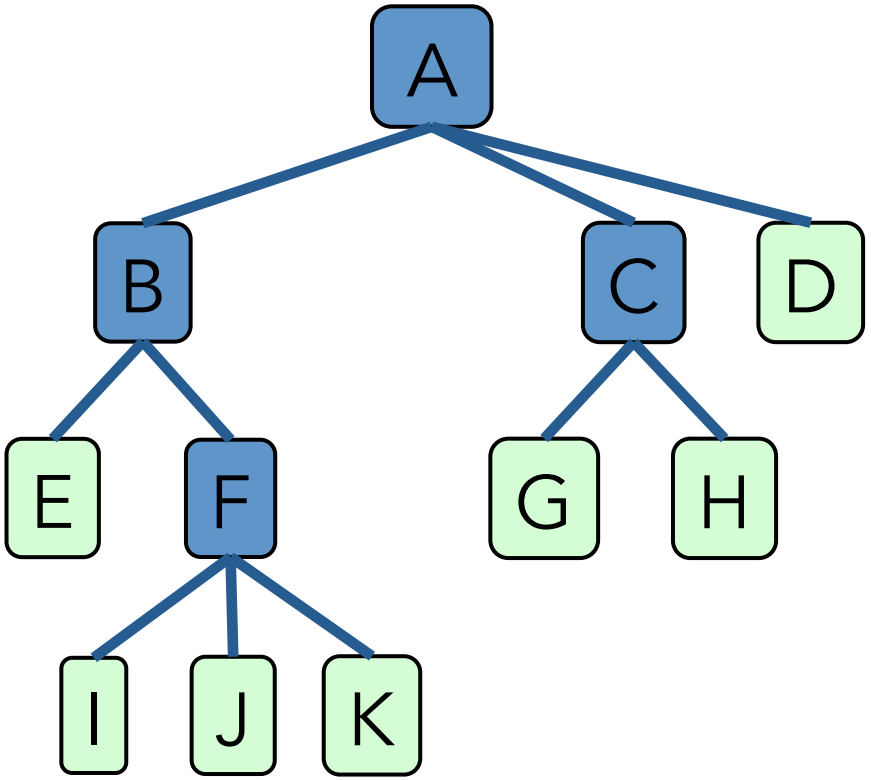
Internal

External

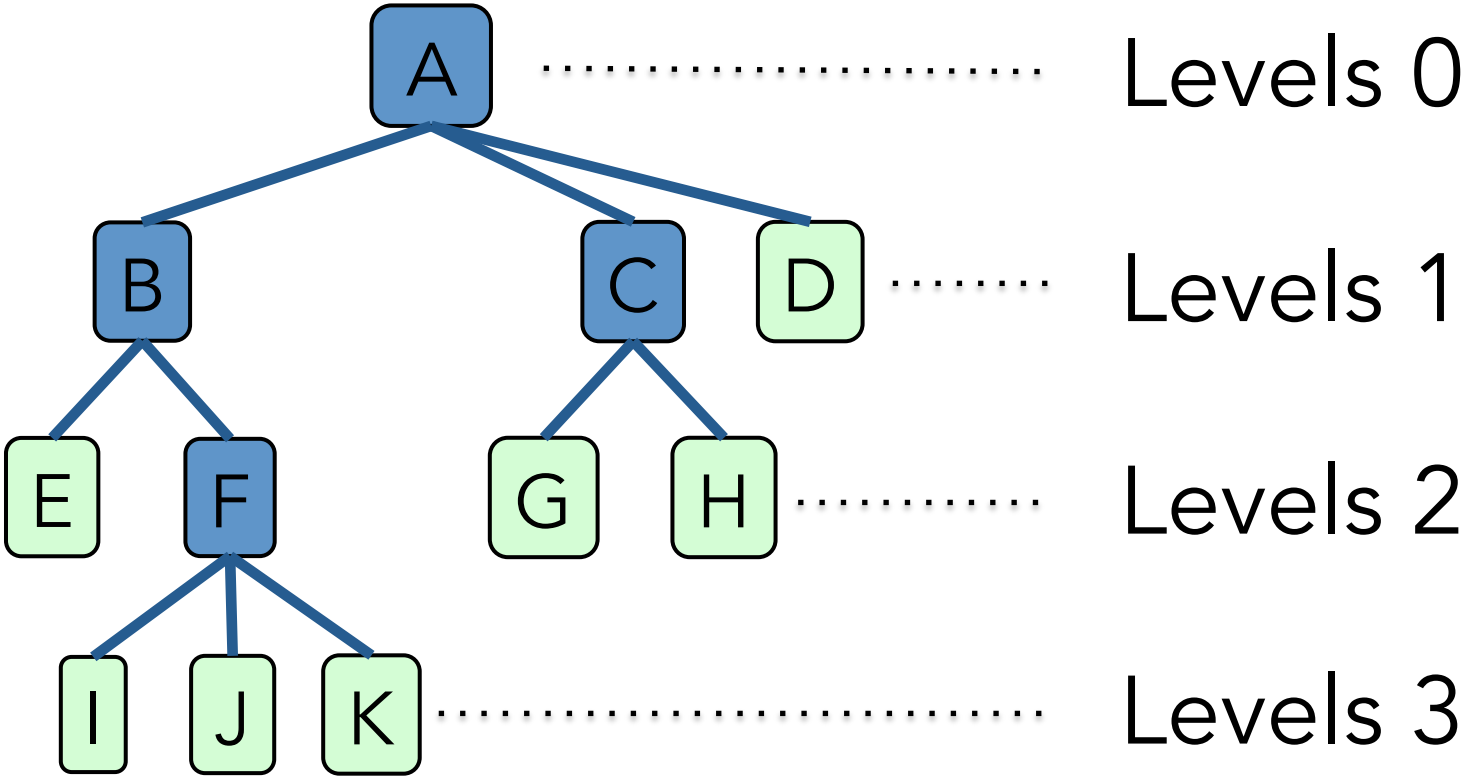


Relations

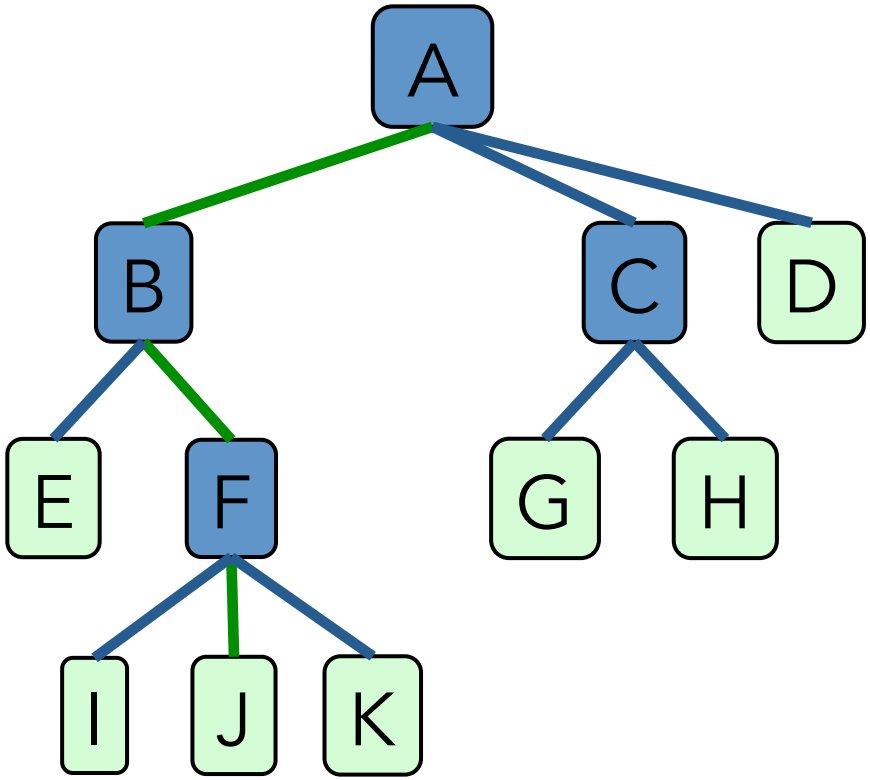
1. Parent
2. Child
3. Sibling
4. Ancestors
5. Descendants



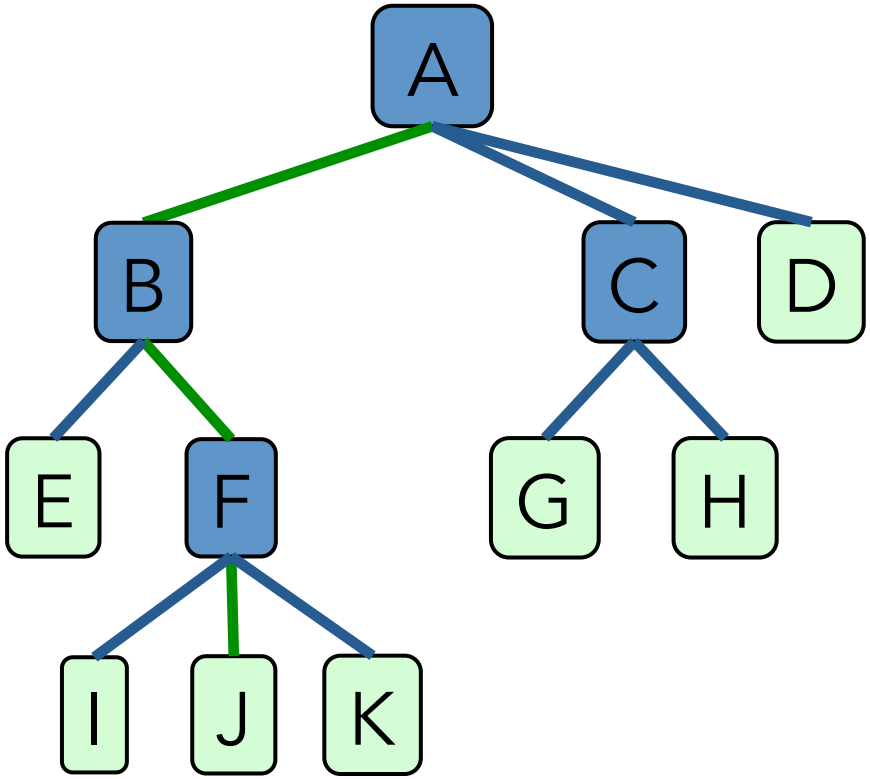
Depth (Level) & Height



Path (A to J)



Subtree



Tree Implementation

Node

```
struct Node{
Elem elt;
Node* par;
Node* children;
}
```

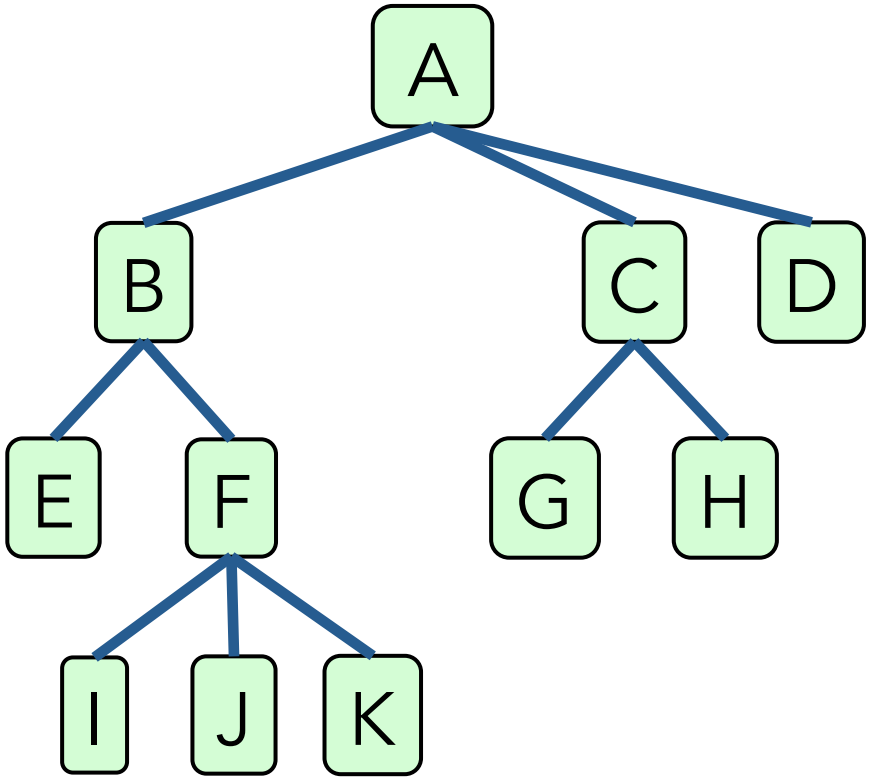
Position for the Node

```
template <typename E>
class Position<E> {
    private:
        Node* v;
    public:
        E& operator*();
        Position parent();
        PositionList children();
        bool isRoot();
        bool isExternal();
friend class Tree;
};
```

Tree Implementation

```
template <typename E> class Tree<E> {
    private:
        Node* _root;
    public:
        int size();
        bool empty();
        Position root();
        PositionList positions();
};
```

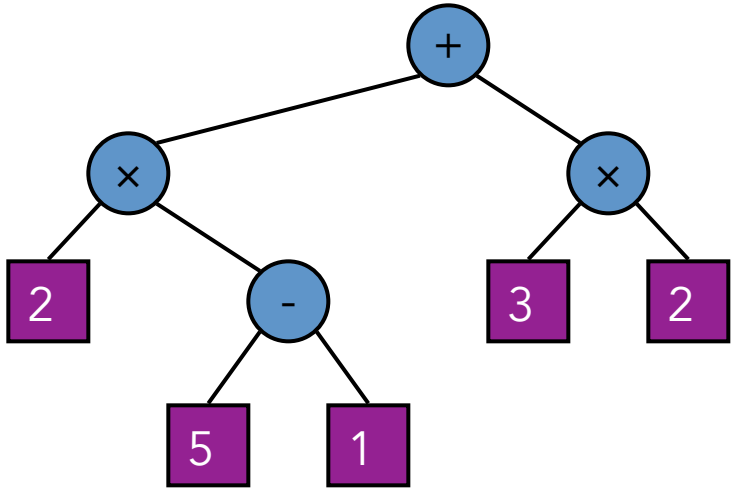
How do we obtain a PositionList?



“Visiting” a Node

Using the node element for some purpose!

If visit is computing the expression, what is the value of this tree?



Postorder Traversal

1. visit children
2. visit parent

Postorder

Algorithm ***postOrder(v)***

for each child ***w*** of ***v***

postorder (w)

visit(v)

Preorder Traversal

1. visit parent
2. visit children

Preorder

Algorithm *preOrder*(*v*)
    visit(*v*)
    for each child *w* of *v*
        preorder (*w*)

Inorder Traversal

1. Visit left subtree
2. Visit node
3. Visit right subtree

Inorder Traversal

```
Algorithm inOrder(v)
    if ! v.isExternal()
        inOrder(v.left())
    visit(v)
    if ! v.isExternal()
        inOrder(v.right())
```

Tree Traversal

1. Postorder
2. Preorder
3. Inorder (Binary Trees Only)

Write a program to calculate the depth of a given node?

```
depth(p):
    if p.isRoot() then
        return 0
    else
        return 1 + depth(p.parent())
```

Write a program to calculate the height of a given tree?

```
height1(T ):
    h=0
    for each p ∈ T.positions() do
        if p.isExternal() then
            h = max(h, depth(T , p))
return h
```

Write a faster algorithm to calculate height!

```
height2(T, p):
    if p.isExternal() then
        return 0
    else
        h=0
        for each q ∈ p.children() do
            h = max(h,height2(T,q))
return 1 + h
```