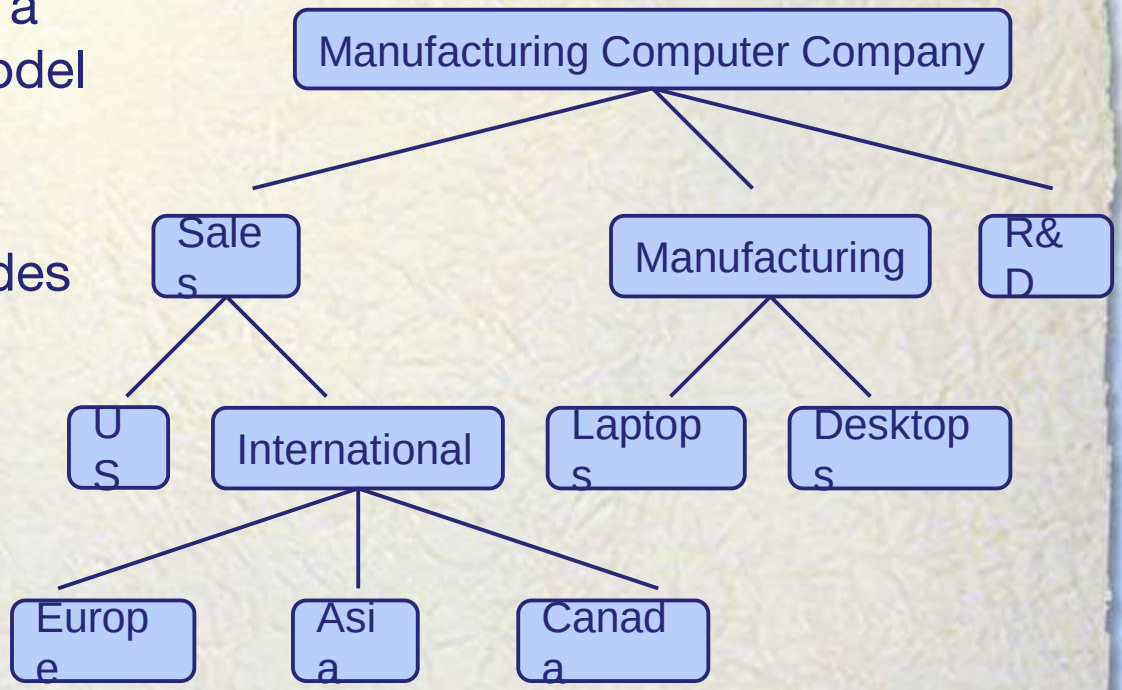
Data Structures and Algorithms

Trees

University of Technology and Engineering
Vietnam National University Hanoi

What is a Tree

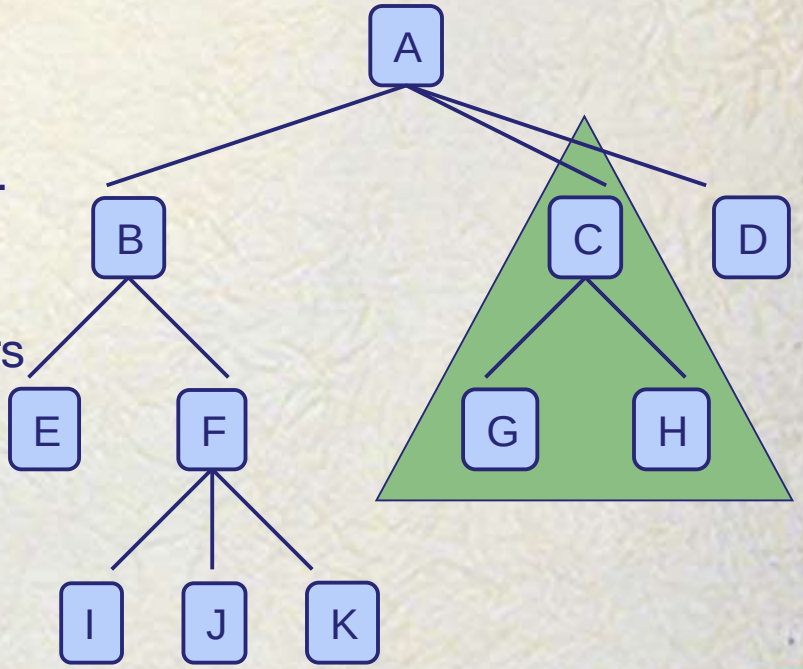
- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Tree Terminology

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Descendant of a node: child, grandchild, grand-grandchild, etc.
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3)
- Siblings: same parent.
- Edge: (u, v): u is the parent of v.
- Path

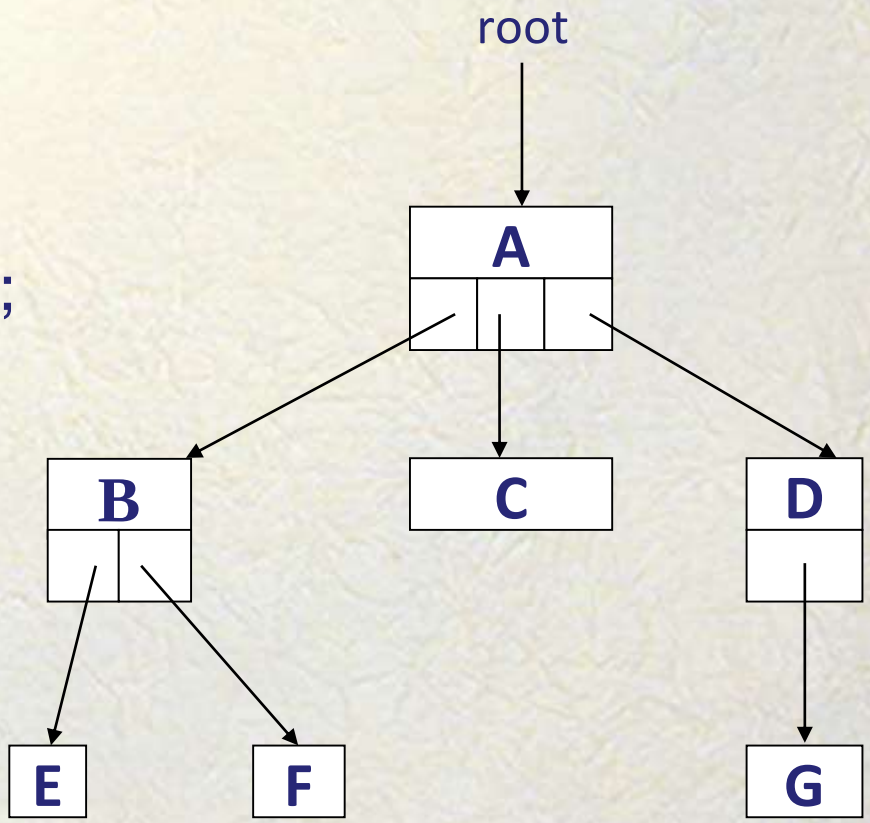
- Subtree: tree consisting of a node and its descendants



List of Children Tree Presentation

```
Template <class Item>
class Node {
    Item data;
    List<Node*> children;
}
```

```
Node<Item>* root;
```



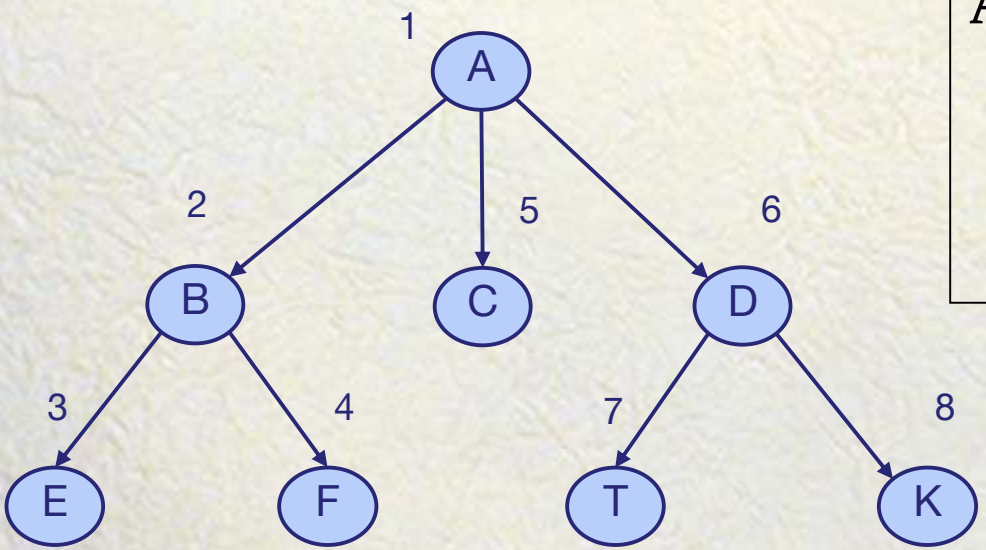
Depth

Depth(v): number of ancestors of v.

Algorithm depth(*T, v*):
    if *v* is the root of *T* **then**
        return 0;
    else
        return 1 + depth(*T, w*), where *w* is the parent of *v* in *T;*

Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Example: Clone a tree



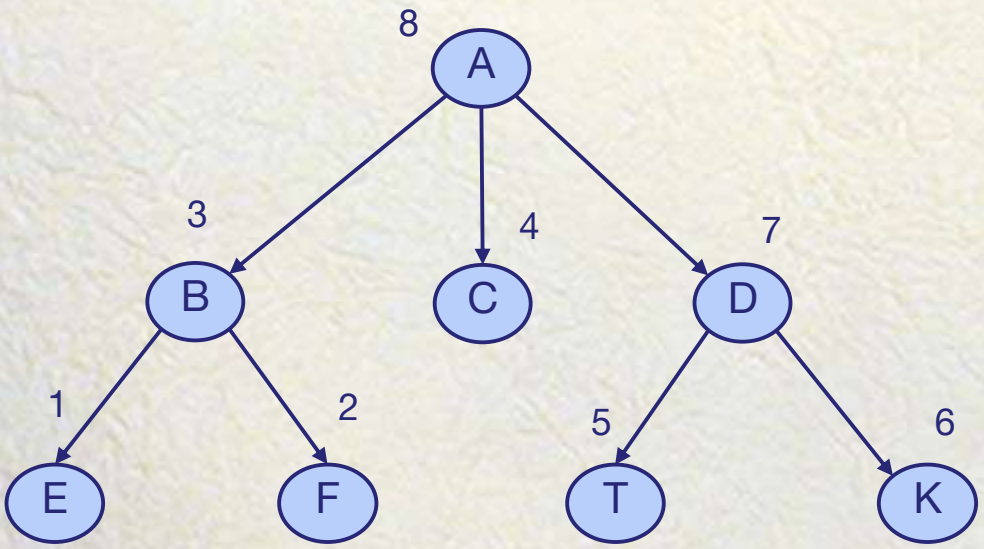
A, B, E, F, C, D, T, K

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

Postorder Traversal

In a postorder traversal, a node is visited after its descendants

Application: Delete a tree from leaves to root

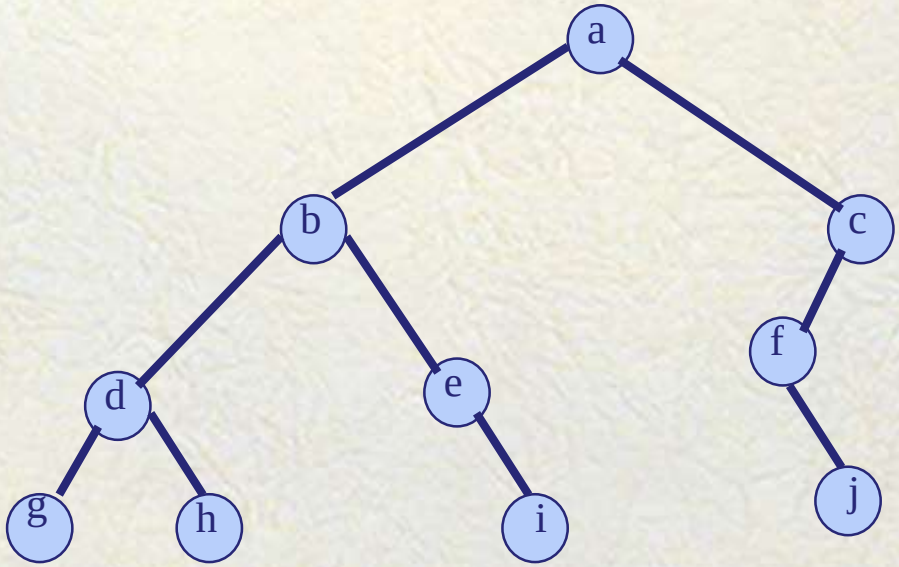


E, F, B, C, T, K, D, A

Algorithm ***postOrder(v)***
  for each child *w* of *v*
    postOrder (*w*);
  visit(*v*);

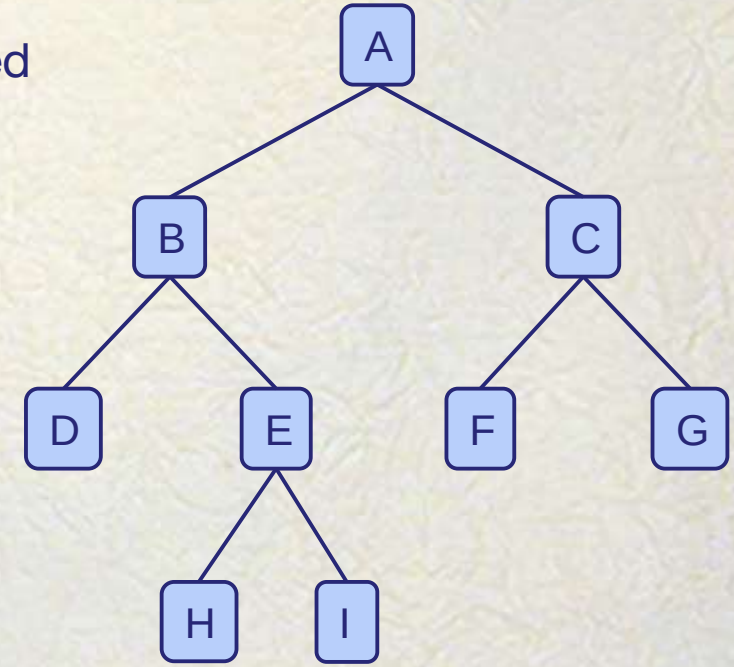
Exercise 1

Show the node orders in preorder and postorder traversals.



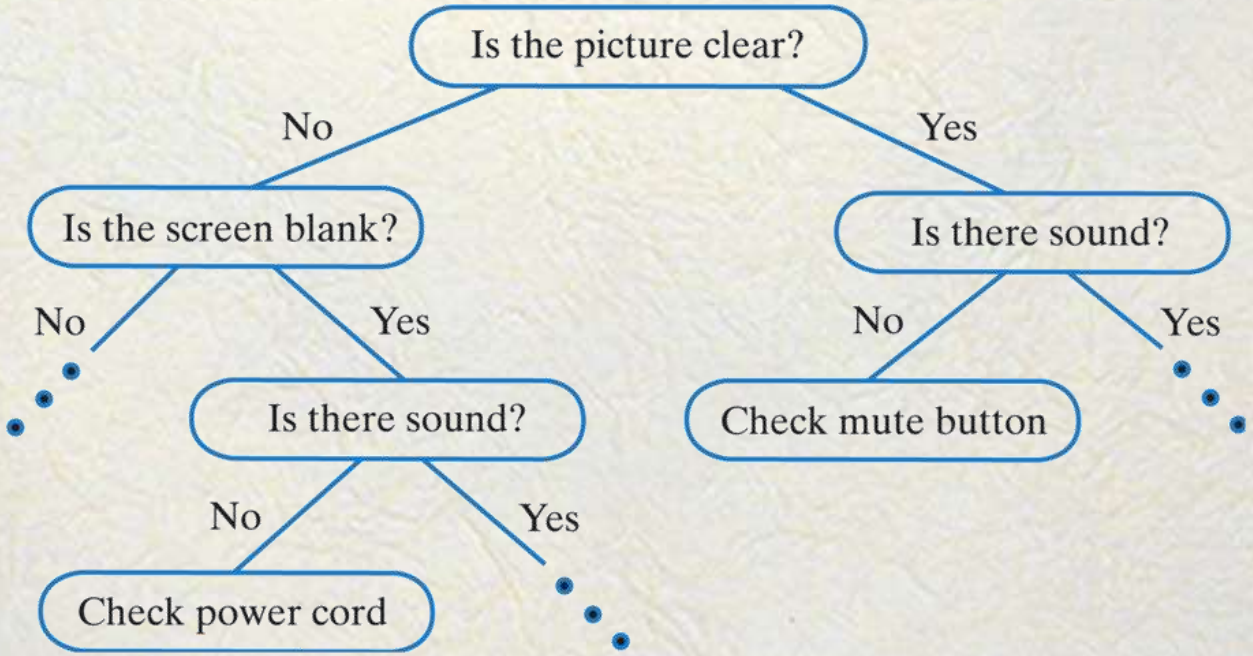
Binary Trees

- A binary tree is a tree each internal node has at most two children, called left child and right child
- Applications:
  - decision processes
  - searching



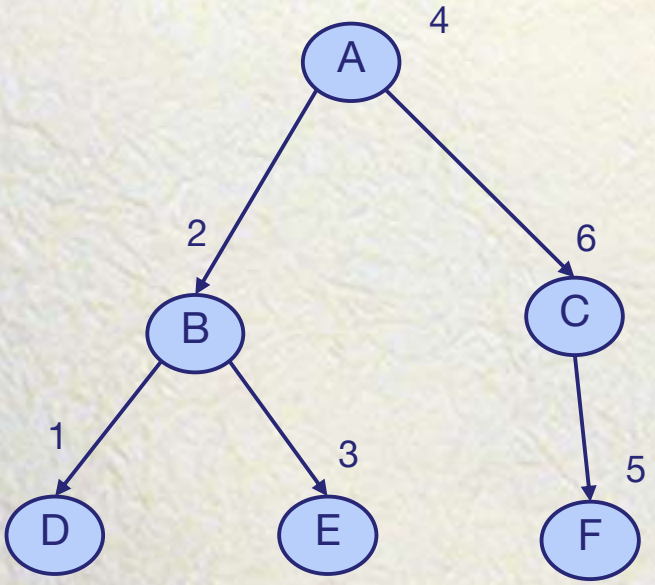
Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions



Inorder Traversal

In an inorder traversal a node is visited after its left subtree and before its right subtree

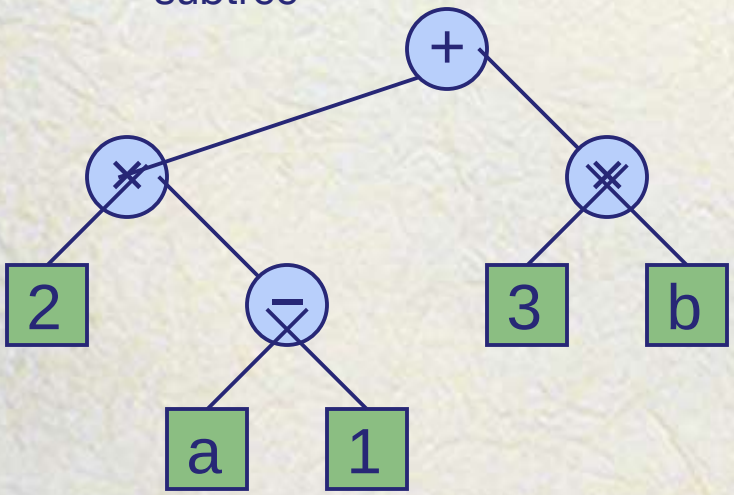


D, B, E, A, F, C

```
Algorithm inOrder(v)
    if hasLeft (v)
        inOrder (left (v));
    visit(v);
8   if hasRight (v)
        inOrder (right (v));
```

Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree



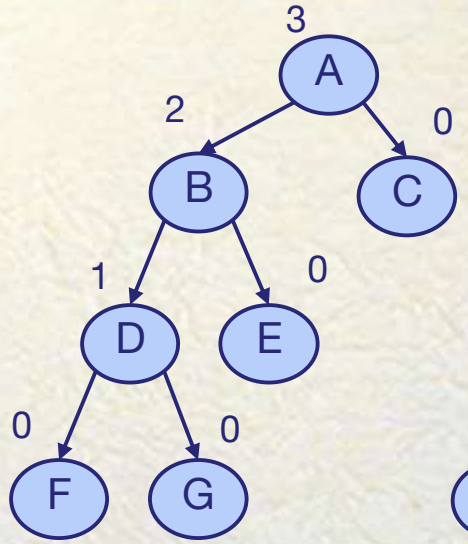
Algorithm *printExpression*(*v*)
  if *hasLeft* (*v*)
    print("(");
    inOrder (*left*(*v*));
  print(*v.element* ());
  if *hasRight* (*v*)
    inOrder (*right*(*v*));
    print (")");

((2 × (a − 1)) + (3 × b))

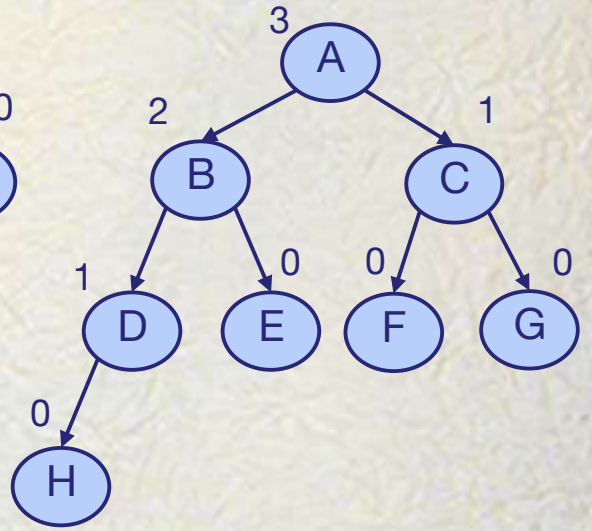
Balanced binary tree

A ***binary tree*** is balanced if for every internal node v of T, the *heights of the children of v can differ by at most 1.*

Note: the height of a balanced binary tree is O(log n)



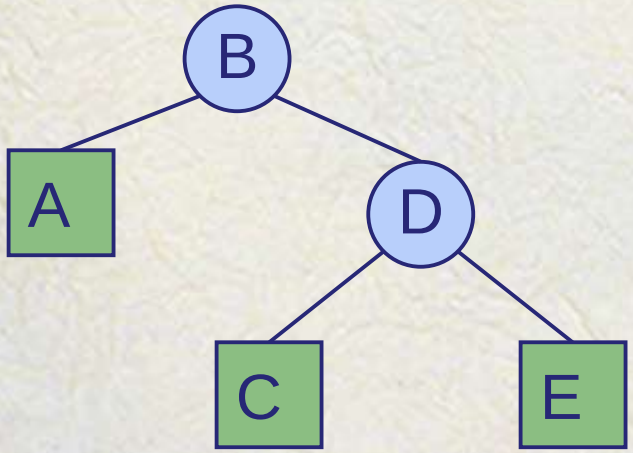
Unbalanced

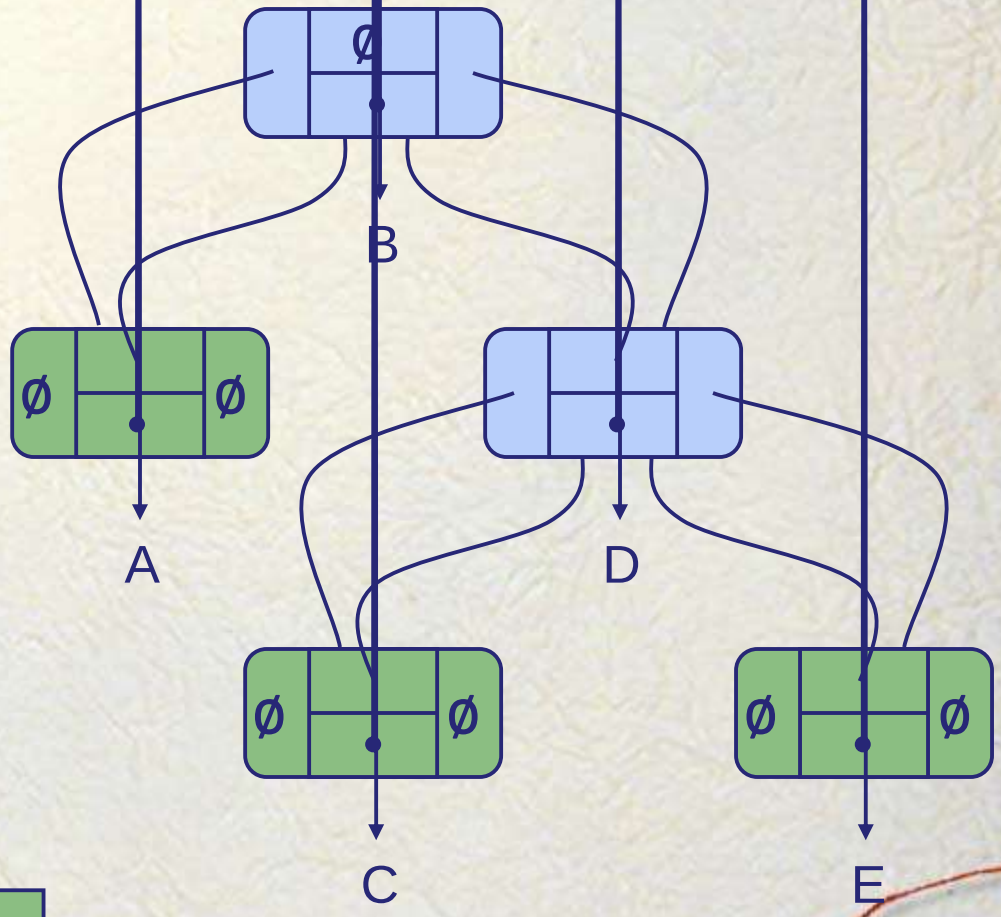


Balanced

Linked Structure for Binary Trees

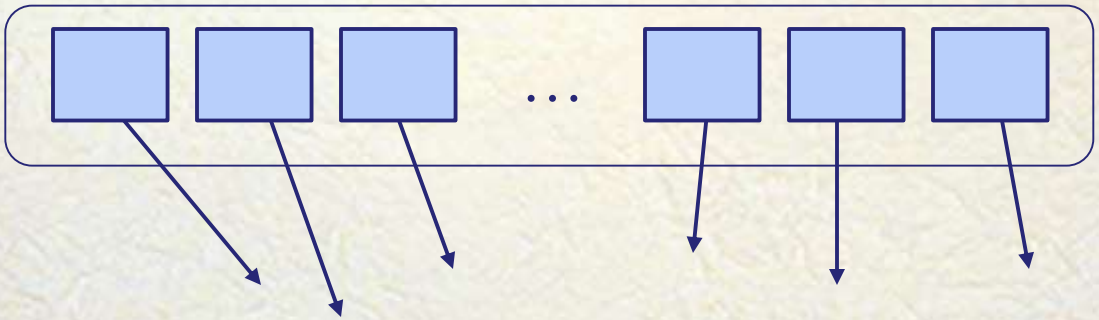
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT





Array-Based Representation of Binary Trees

- nodes are stored in an array



- let rank(node) be defined as follows:
  - rank(root) = 1
  - if node is the left child of parent(node),
    rank(node) = 2*rank(parent(node))
  - if node is the right child of parent(node),
    rank(node) = 2*rank(parent(node))+1