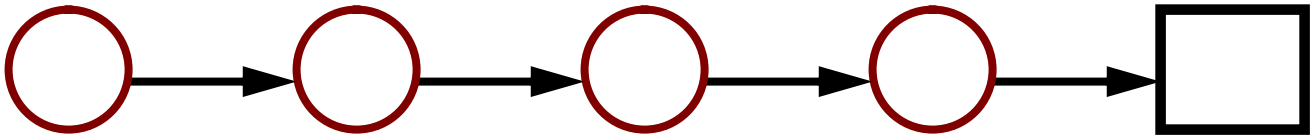
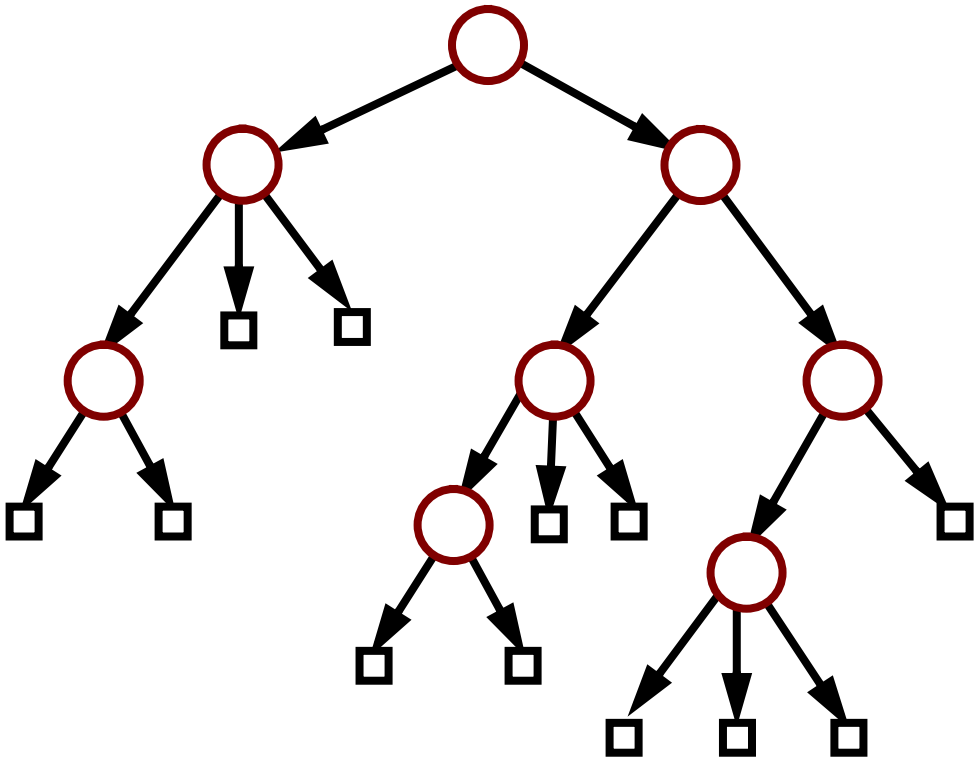
Trees vs. Linked Lists

- Linked list is collection of nodes where each node references *only one* neighbor.



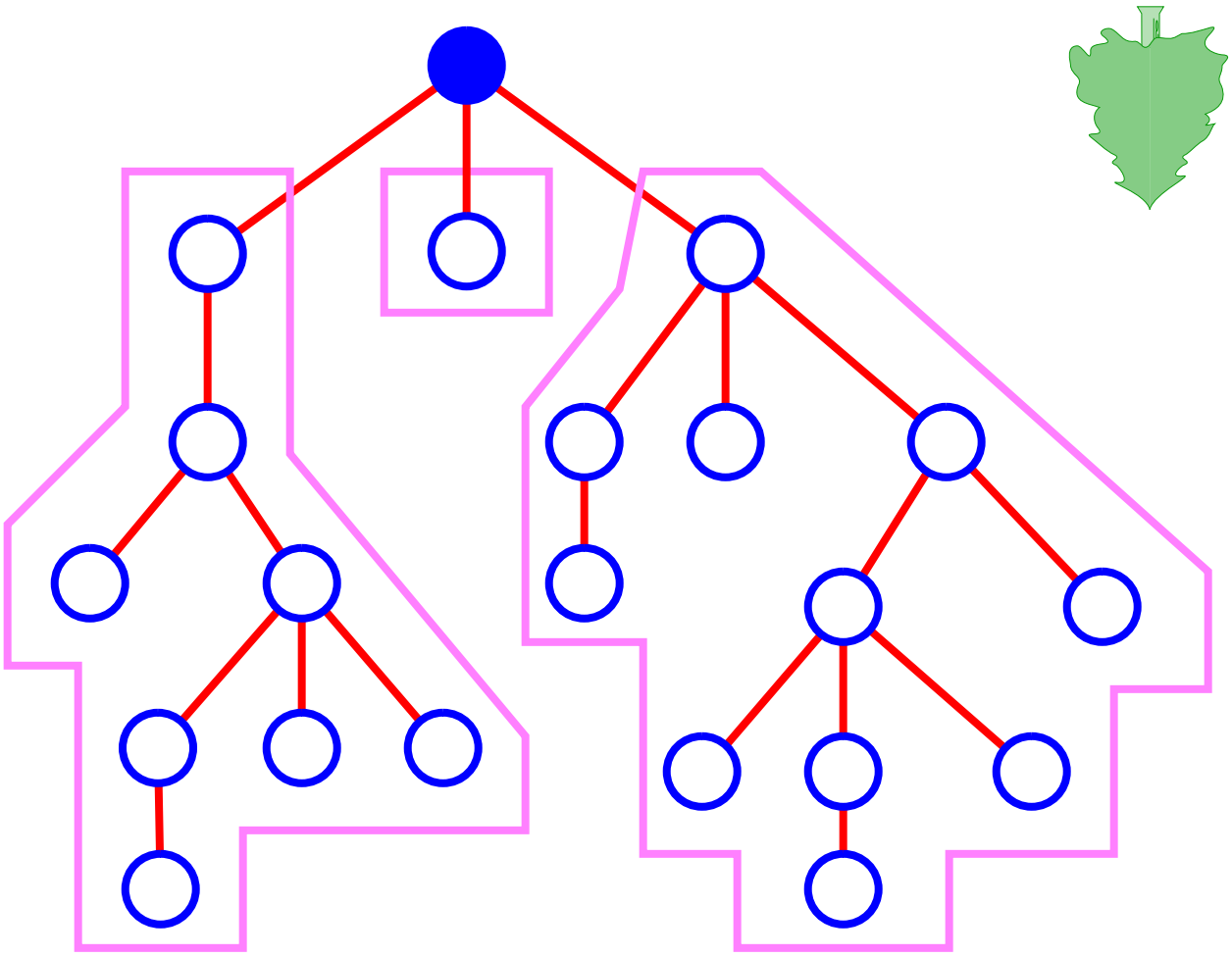
- Tree is also collection of nodes, but each node may *reference* multiple neighbors.



- Tree can be used to model *hierarchical organization* of data.

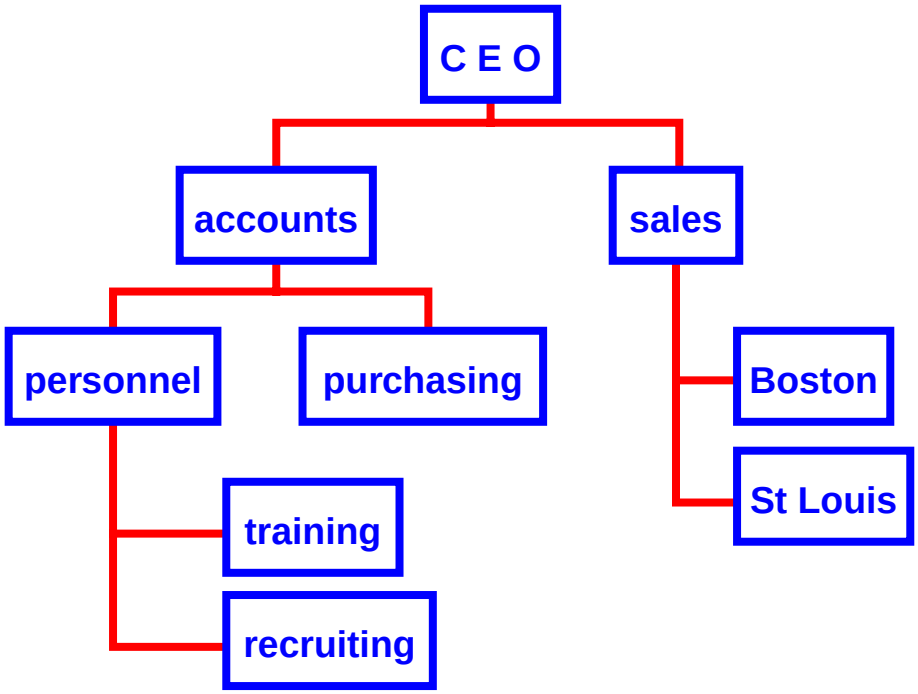
Trees

- A (***rooted***) ***tree T*** is a collection of nodes connected by edges such that:
  - there is a designated ***root*** node
  - the nodes connected by edges to the root are themselves roots of disjoint trees, called ***subtrees*** of ***T***

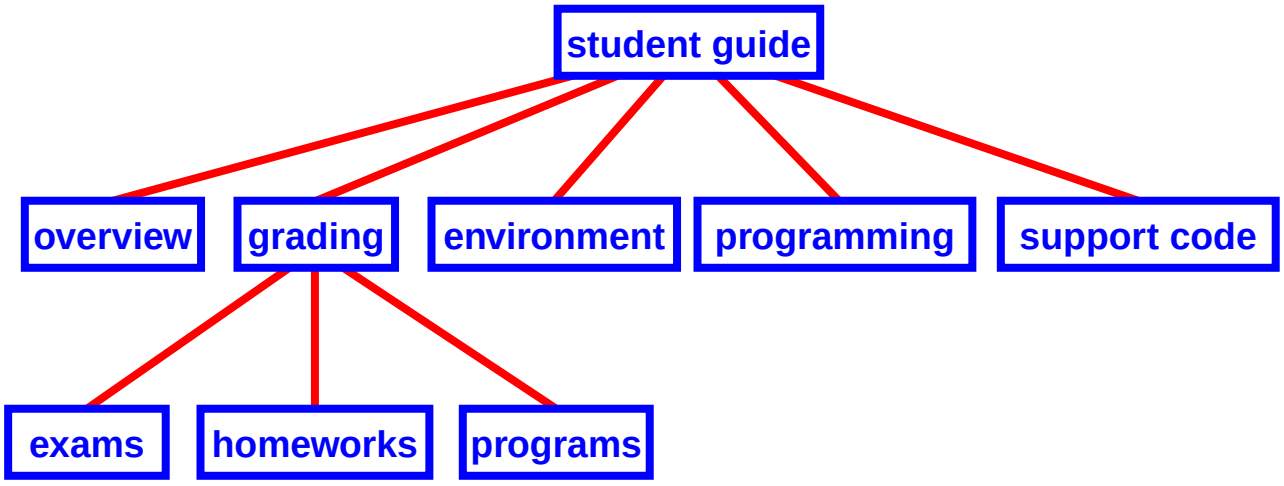


Examples of Trees

- organization chart



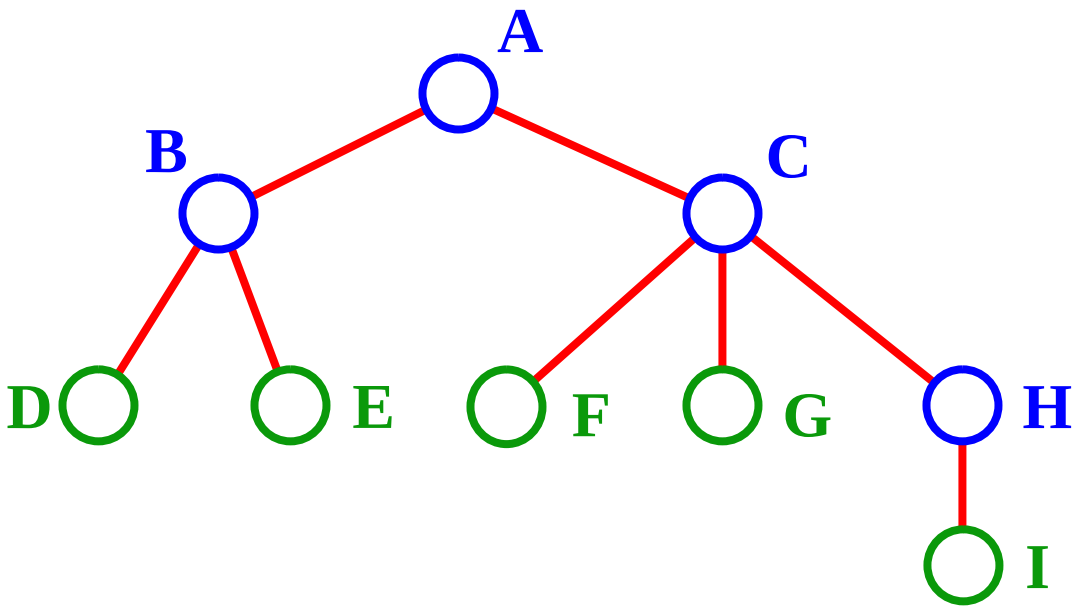
- table of contents



- other ...

Terminology

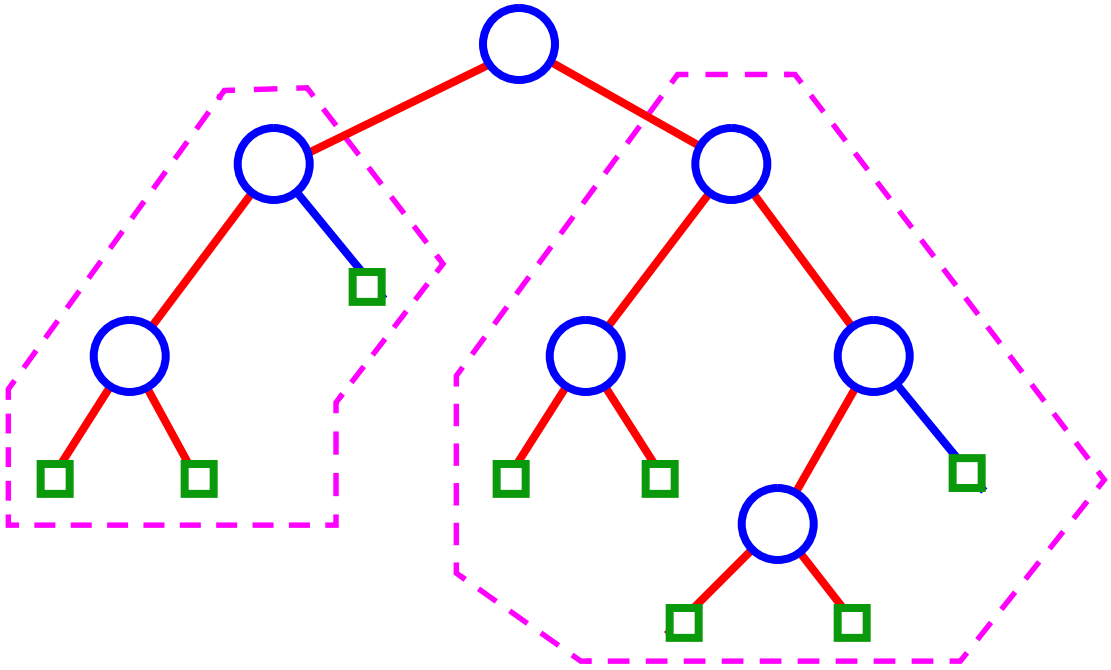
- ***A*** is the ***root*** node.
- ***B*** is the ***parent*** of D and E.
- ***C*** is the ***sibling*** of B
- ***D*** and ***E*** are the ***children*** of B.
- ***(C,F)*** is an ***edge***
- ***D, E, F, G***, I are ***external nodes*****, or** ***leaves*** (i.e., nodes with no children).
- ***A, B, C, H*** are ***internal nodes*****.**
- The ***depth*** (***level***) of ***E*** is ***2***
- The ***height*** of the tree is ***3***.
- The ***degree*** of node ***B*** is ***2***.



Property: (***# edges***) = (***#nodes***) – 1

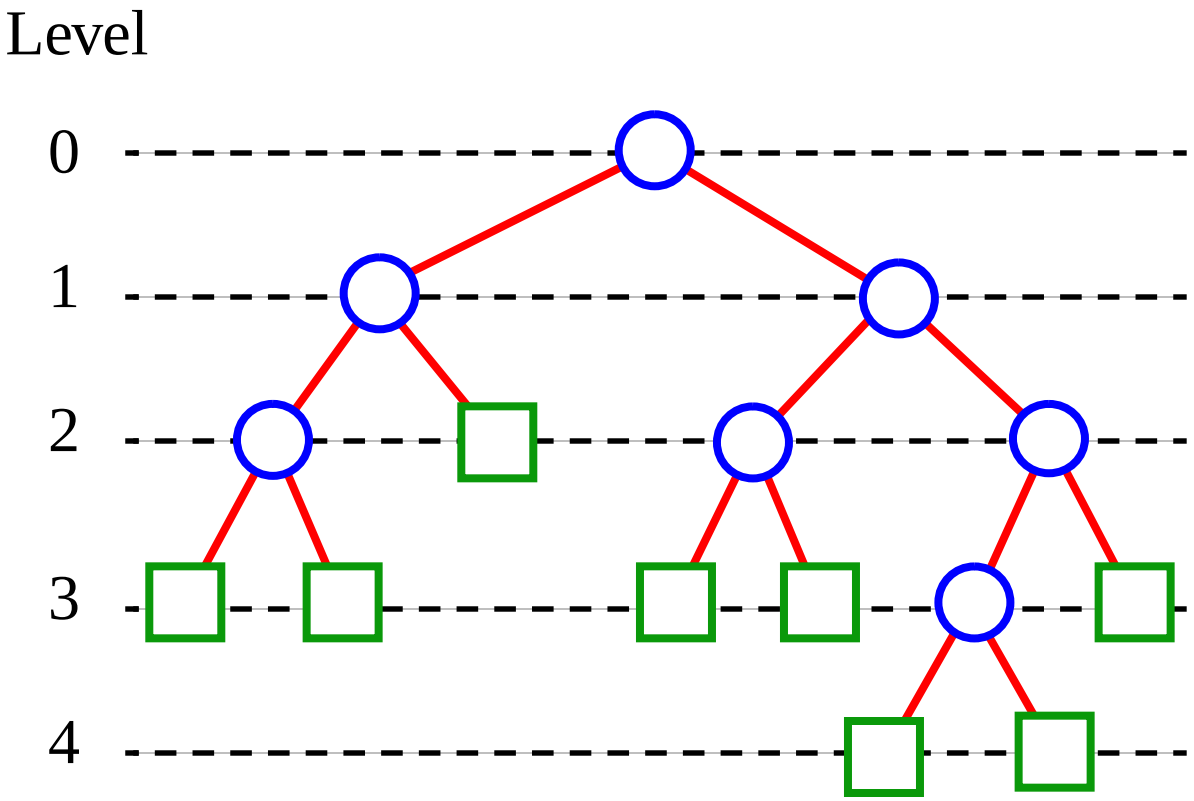
Binary Trees

- ***Ordered tree:*** the children of each node are ordered.
- ***Binary tree:*** ordered tree with all internal nodes of ***degree 2***.
- ***Recursive definition of binary tree:***
- A ***binary tree*** is either
  - an external node (leaf), **or**
  - an internal node (the ***root***) and two binary trees (***left subtree*** and ***right subtree***)



Properties of Binary Trees

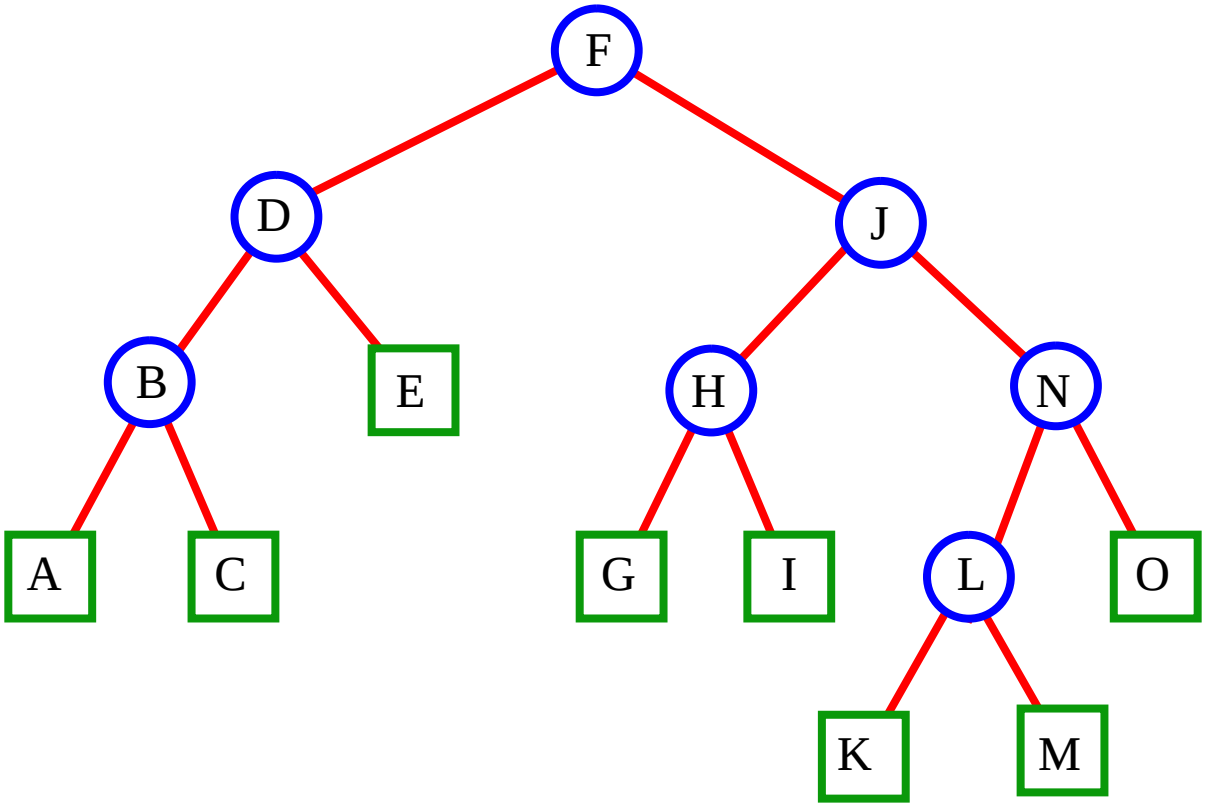
- (# external nodes) = (# internal nodes) + 1
- (# nodes at level **i**) $\leq 2^{\mathbf{i}}$
- (# external nodes) $\leq 2^{(\text{height})}$
- (height) $\geq \log_2$ (# external nodes)



Traversing Binary Trees

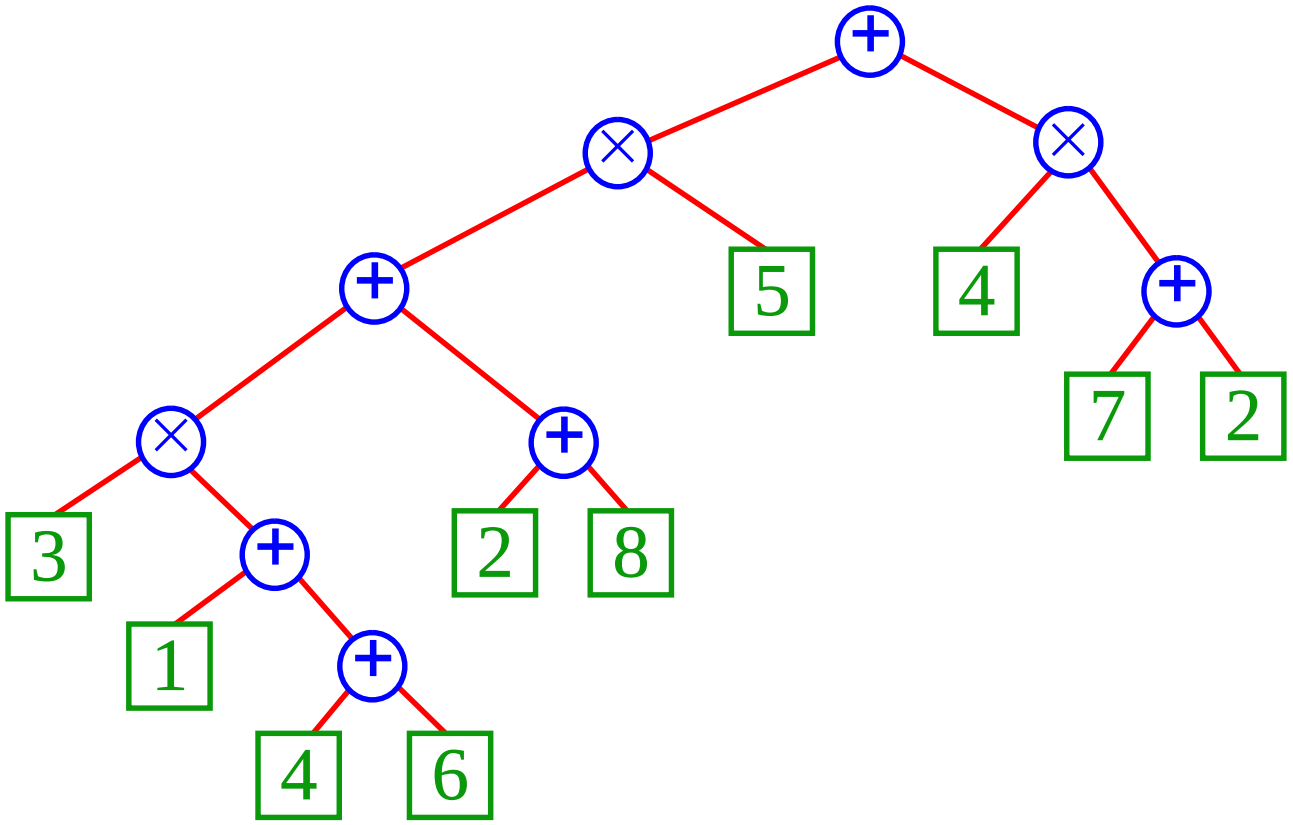
Preorder Traversal:

1. visit root
2. traverse left subtree
3. traverse right subtree



F D B A C E J ···

Printing Arithmetic Expressions



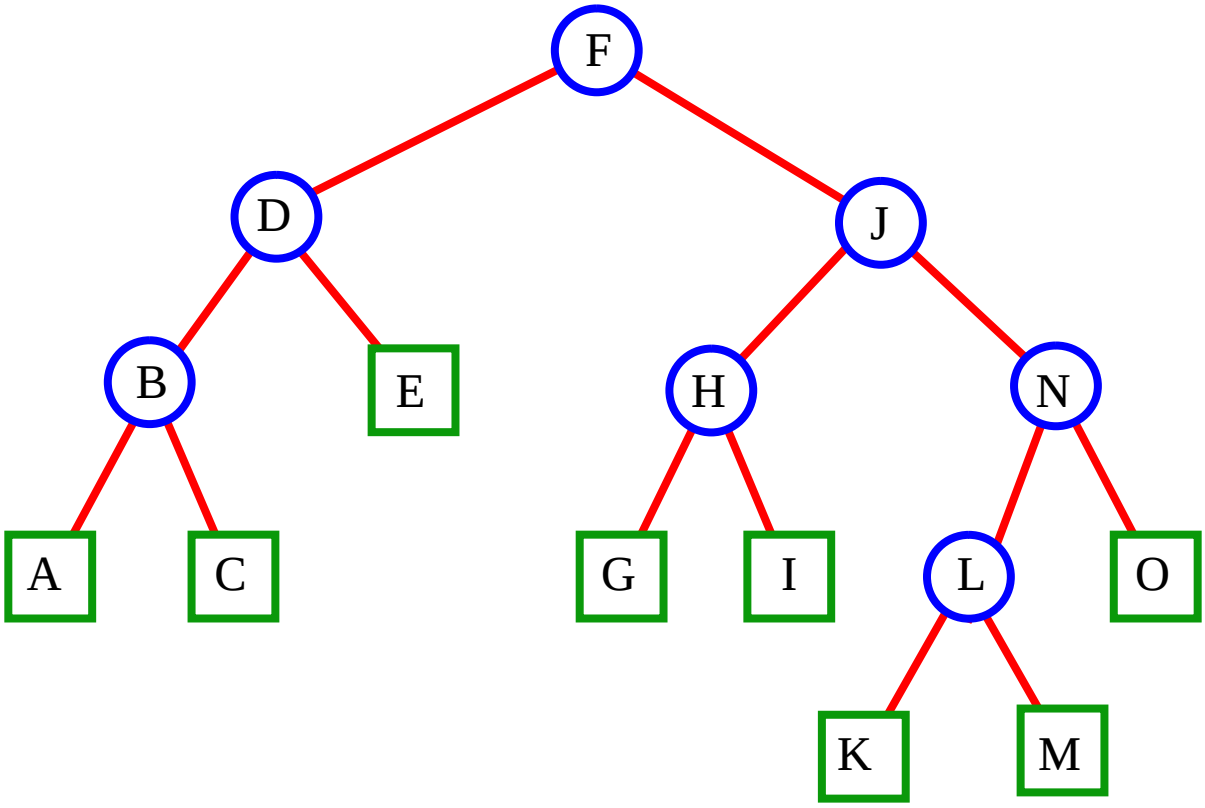
((((3 × (1 + (4 + 6))) + (2 + 8)) × 5) + (4 × (7 + 2)))

- method void print()
  - for a leaf, print() writes its value
  - for an internal node, print() does:
    write "("; left_.print();
    write operator symbol ("+" or "×")
    right_.print(); write ")"
- Inorder Traversal!

Traversing Binary Trees

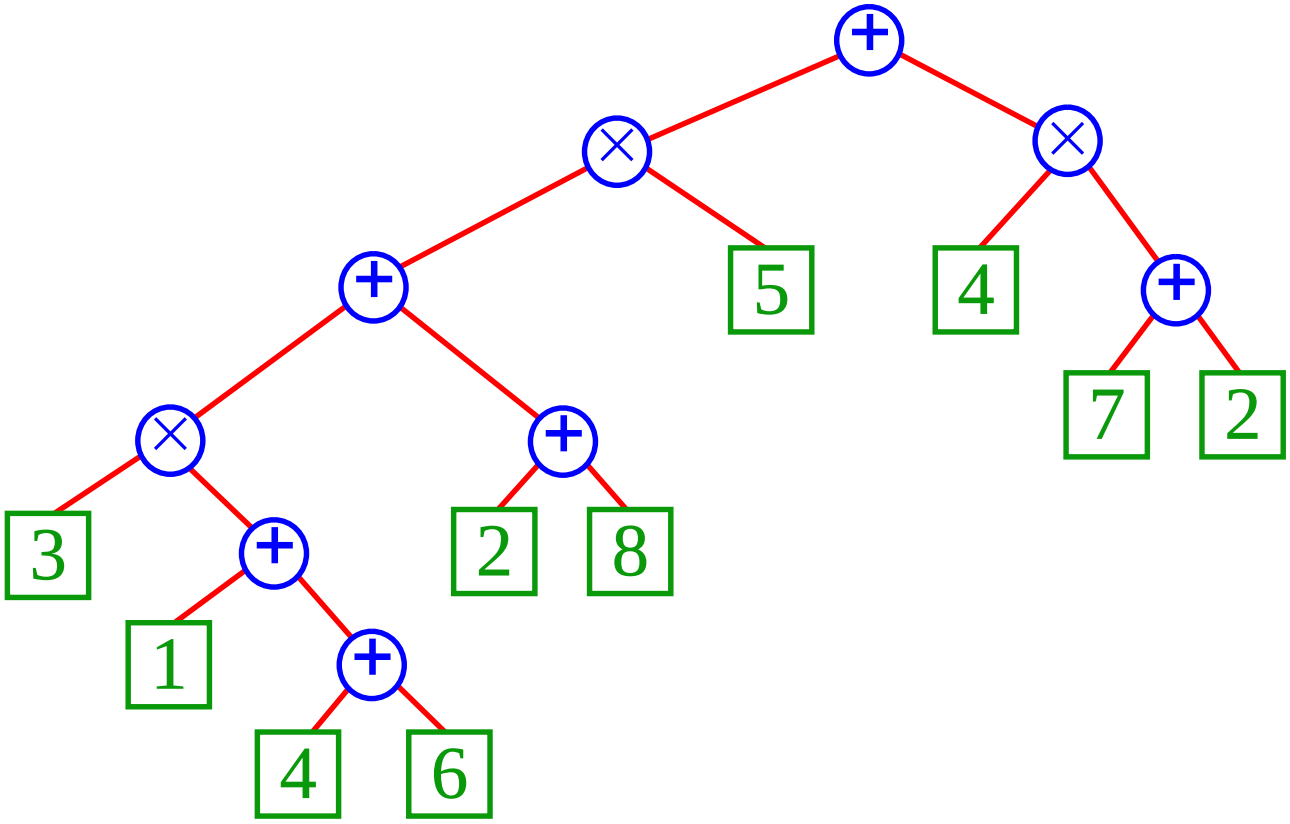
Postorder Traversal:

1. traverse left subtree
2. traverse right subtree
3. visit root



A C B E D ···

Evaluating Arithmetic Expressions



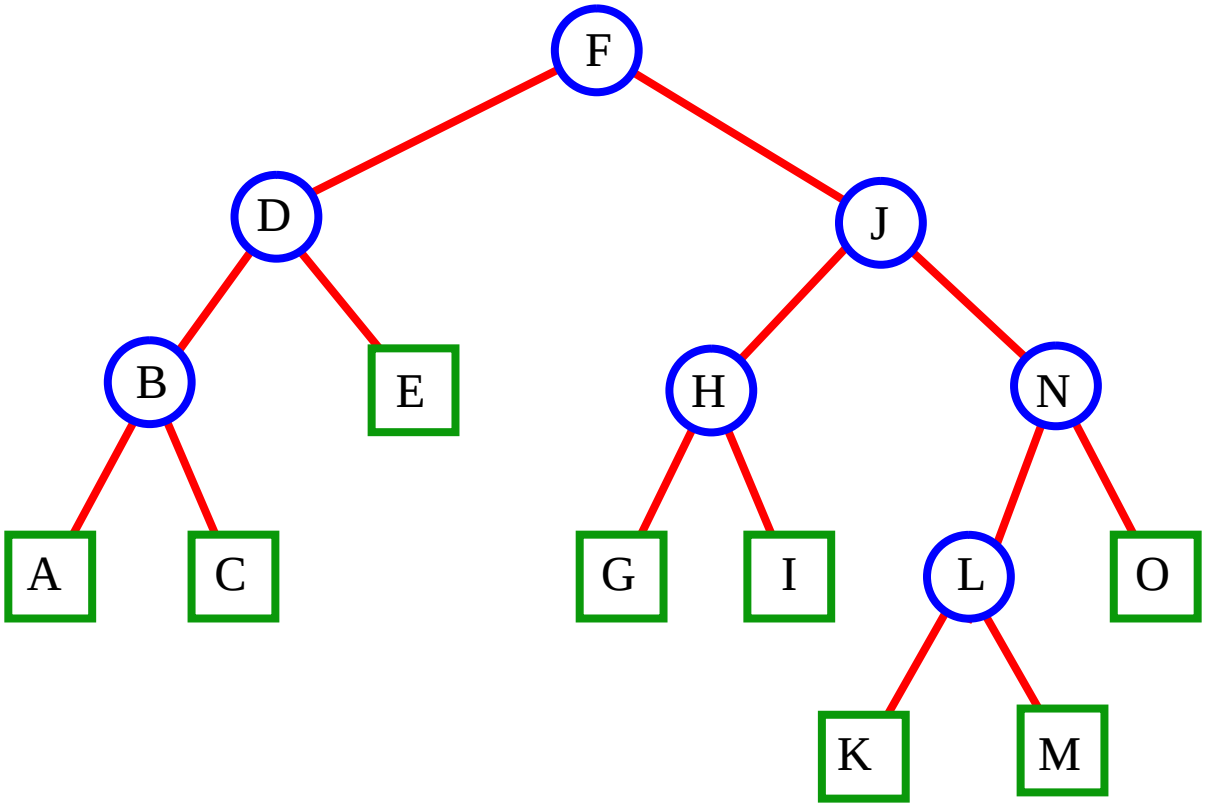
((((3 × (1 + (4 + 6))) + (2 + 8)) × 5) + (4 × (7 + 2)))

- method eval()
  - for a leaf, eval() returns its value
  - for an internal node, eval() returns:
    left_.eval() + right_.eval() or
    left_.eval() × right_.eval()
    depending on the operator stored at the node
- Postorder Traversal!

Traversing Binary Trees

Inorder Traversal:

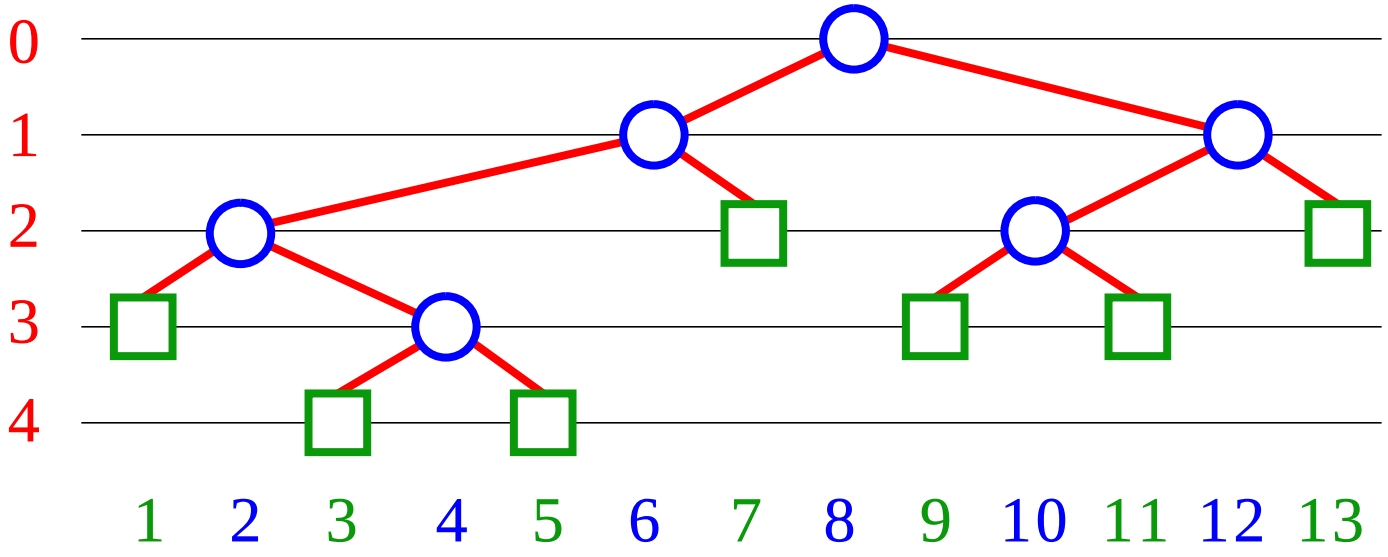
1. traverse left subtree
2. visit root
3. traverse right subtree



A B C D E] ...

Drawing Binary Trees

- y(v) = (depth of v)
- x(v) = (inorder rank of v)



method int layout(int depth, int rank)

- for leaf node:

```
y_ = depth;
x_ = rank;  return rank+1;
```

- for internal node:

```
y_ = depth;
x_ = left_.layout(depth + 1, rank);
rank = x_+1;
return right_.layout(depth + 1, rank);
```

- Inorder traversal for x-coordinates!