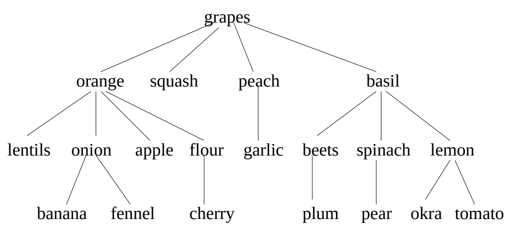
CPS 100 - Program Design and Analysis II
Dr. Susan Rodger
Section: Trees (handout)

Read Chap 6.5-6.6, Chap 17. We will not use the class implementation of trees in the book, but you should read about them.

General Tree

- set of nodes (vertices)
- set of edges
- no cycles

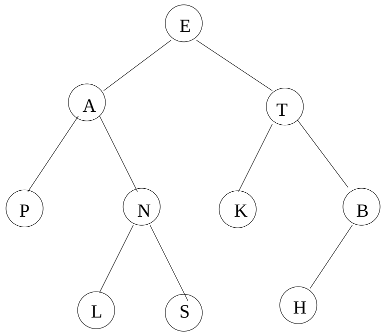


Terms

- path - list of successive vertices connected by edges
- root - (must designate - usually this is the node at the top)
- family terms for nodes
  - parent
  - child
  - sibling
    Siblings of squash? garlic?
  - ancestor
    orange is ancestor of
  - descendant
- leaf (or terminal) node - no children
- nonleaf (nonterminal) node - at least one child
- subtree - any node together with all its descendants
- forest - set of trees
- level of a node: 1 + length of the path from the root to this node.
  Level of grapes? beets?
  NOTE: Definition of level in other books may be without the (+1).
- height of a tree - max level of all its nodes (or 0 if no nodes).

Binary Tree

- each node has at most 2 children
- children are ordered - either left node or right node



Full binary tree

A binary tree of height h with no missing nodes on a level.

Complete binary tree

A binary tree of height h that is full to level h-1 and level h is filled from left to right.

Properties

1. A tree with N nodes has N-1 edges.

2. The height of a full binary tree with N nodes is $\log_2(N)$.

Implementation of binary tree

```
struct TreeNode
{
   int info;
   TreeNode * left;
   TreeNode * right;
// TreeNode * parent;   // optional
};
```

We won't use the parent pointer very much. It is not needed if recursion is used.

Binary Tree Traversals

Preorder

- visit root (print value)
- visit left subtree
- visit right subtree

Recursive Algorithm

```
void Preorder(TreeNode * Tree)
{  if (Tree != NULL) {
     Visit(Tree->Info);
     Preorder(Tree->left);
     Preorder(Tree->right);
   }
}
```

Preorder:

Stack-based implementation

```
void Traverse(TreeNode * t)
{
   Stack <TreeNode *> S;
   S.Push(t);
   while (!S.IsEmpty())
   {
      t = S.Top();
      S.Pop();









   }
}
```

Inorder Traversal

- visit left subtree
- visit root (print value)
- visit right subtree

Inorder:

Postorder Traversal

- visit left subtree
- visit right subtree
- visit root (print value)

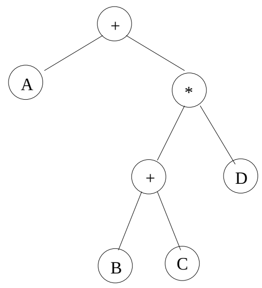
Postorder:

Problem: How many leaves?

Write a recursive function to return the number of leaves in a tree.

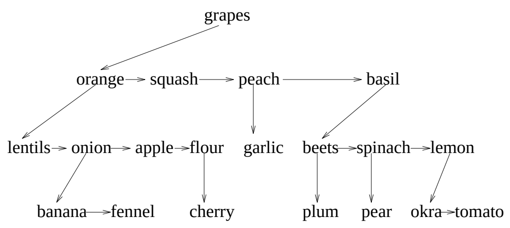
```
int NumberLeaves(TreeNode * t)
{
```

Example: Parse Tree (Expression Tree)



Represents

Implementation of General Tree




```
struct GenTree
{
   string info;
   GenTree * sibling;
   GenTree * child;
};
```

Problem:

Print the number of leaves in the tree.

```
int NumberLeaves(GenTree * t)
{
```