CHAPTER 11 Trees

11.1 Introduction to Trees

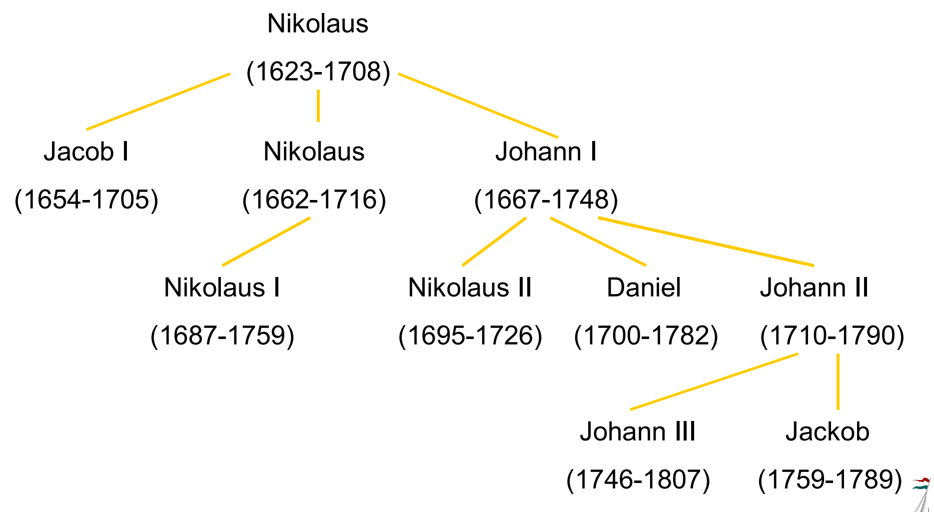
11.2 Applications of Trees

11.3 Tree Traversal

11.4 Spanning Trees

11.5 Minimum Spanning Trees

The Bernoulli family of mathematicians



tree

【Definition 1】 A *tree* is a **connected undirected** graph with **no simple circuits**.

Forest is an undirected graph with no simple circuits.

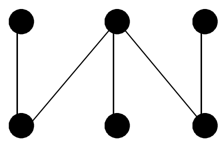
Note:

① Any tree must be a simple graph.

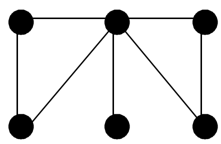
② Each connected components of forest is a tree.

〖Example 1〗 Which graphs are trees?

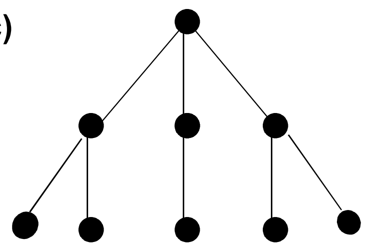
a)



b)



c)



【 Theorem 1 】 An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Proof:

(1) $\Rightarrow$

- ✓ there is a simple path between any two of its vertices
- ✓ unique

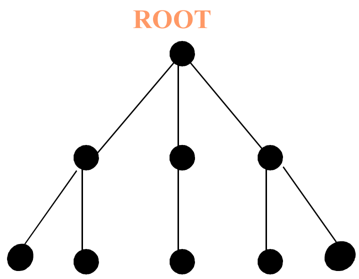
(2) $\Leftarrow$

- ✓ connected
- ✓ no simple circuits

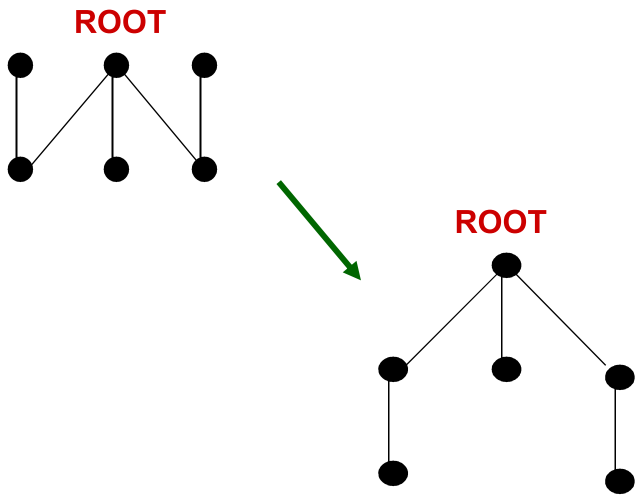
Rooted tree

- In many applications of trees a particular vertex of a tree is designated as the *root*.
- Once we specify a root, we direct each edge away from the root.
- Thus, a tree together with its root produces a directed graph called a *rooted tree*.

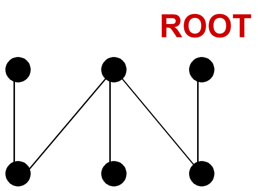
For example,



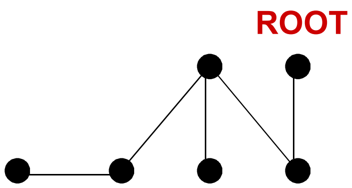
〖Example 2〗 Change an unrooted tree into a rooted tree.



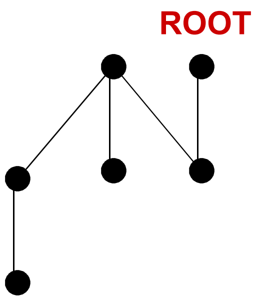
Problem: What if a different root is chosen?



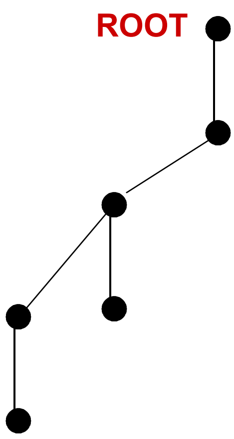
Problem: What if a different root is chosen?



Problem: What if a different root is chosen?



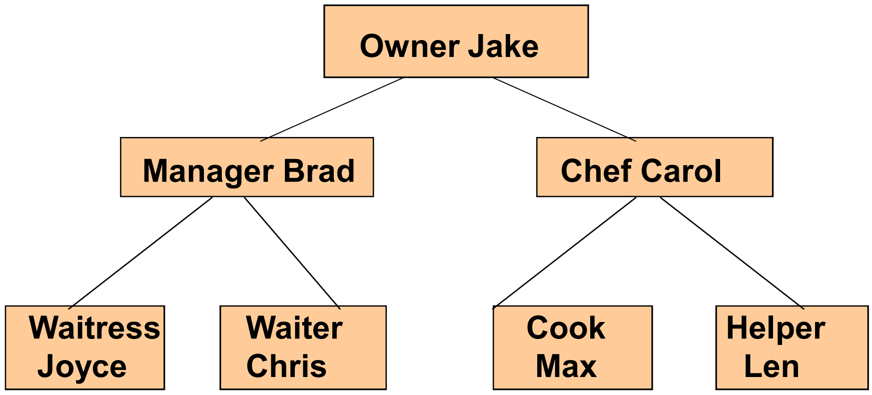
Problem: **What if a different root is chosen?**



Some concepts in tree

- **Parents vs. Children**
- **Siblings**
- **Ancestor vs. Descendants**
- **Root, leaf, and internal vertices**
- **Subtrees**

Jake’s Pizza Shop Tree

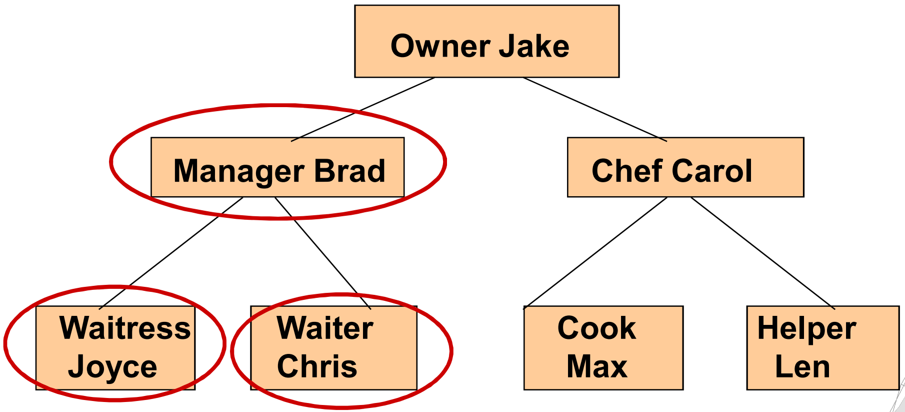


Parent

The *parent of a non-root vertex* v is the *unique* vertex u with a directed edge from u to v.

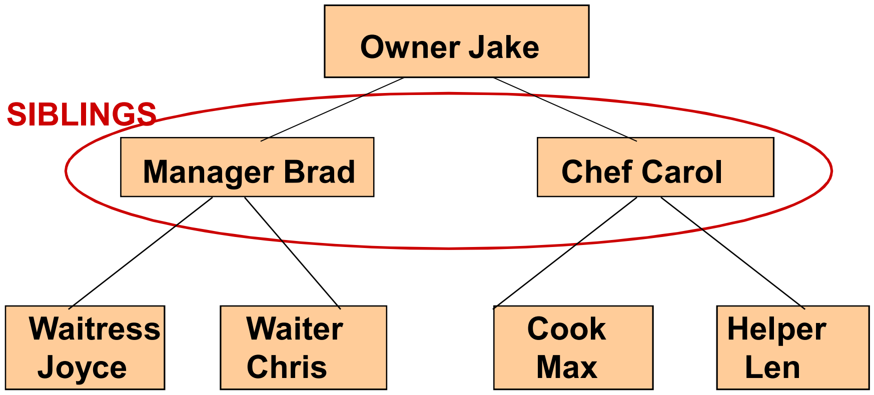
Child

When u is the parent of v, v is called a *child* of u.



Sibling

Vertices with the same parent are called *siblings* .

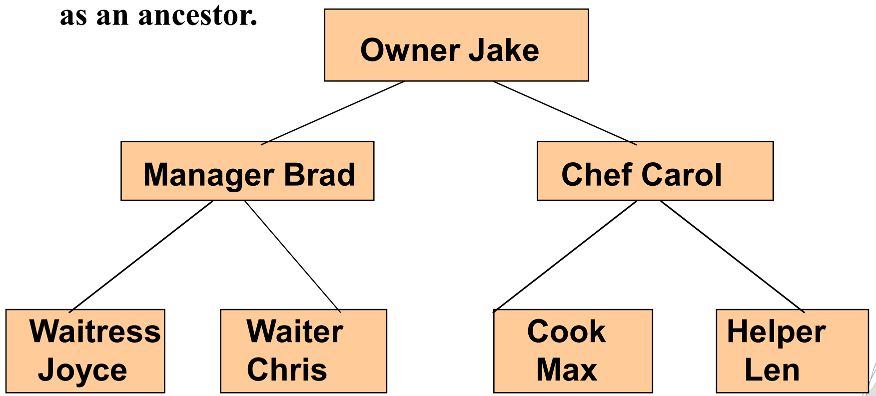


Ancestors

The *ancestors of a non-root vertex* are all the vertices in the path from root to this vertex.

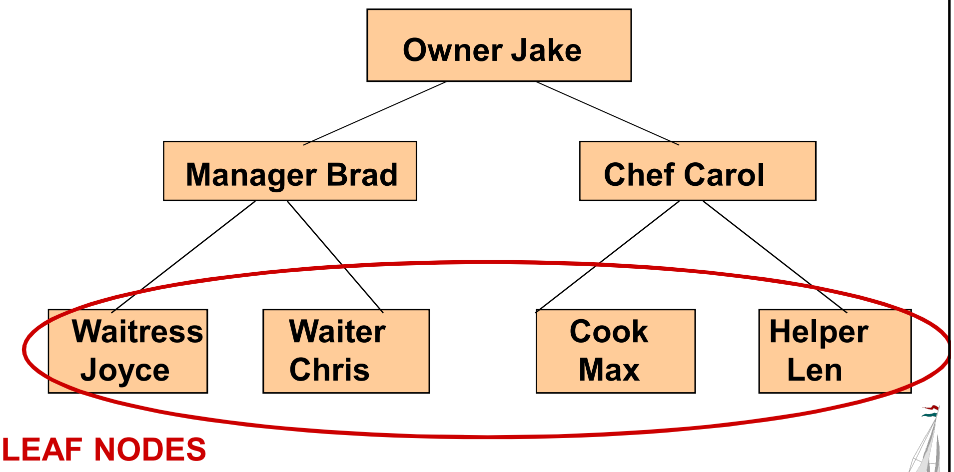
Descendants

The *descendants of vertex v* are all the vertices that have *v* as an ancestor.



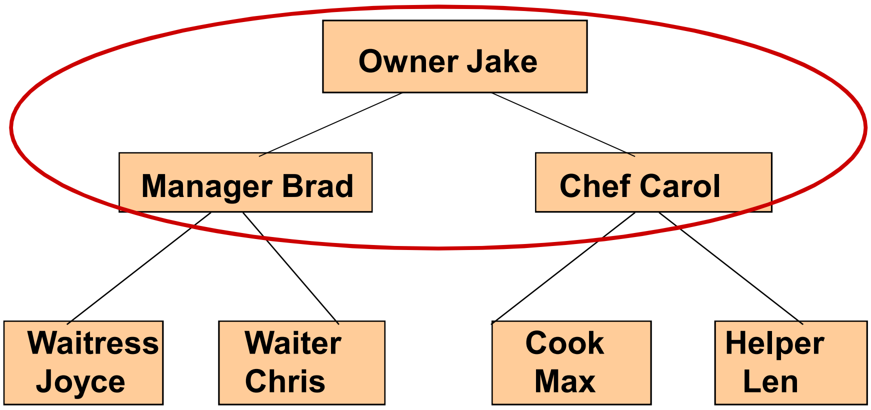
Leaf

A vertex is called a *leaf* if it has no children.



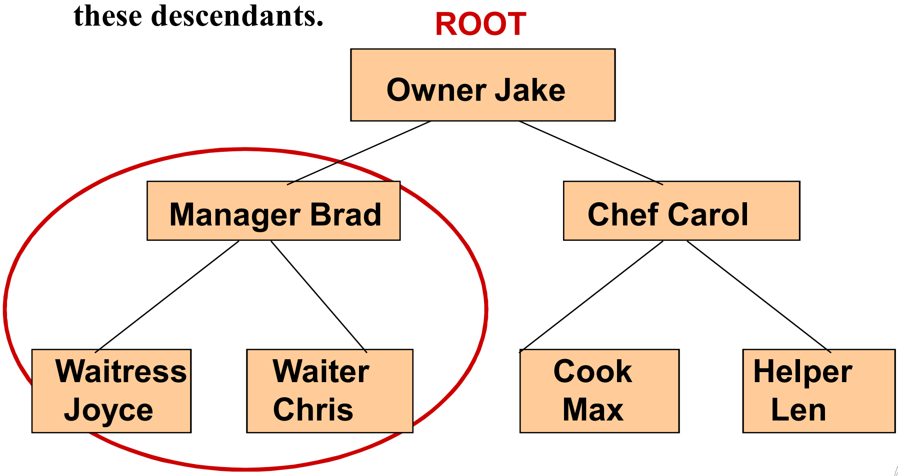
Internal Vertex

A vertex that has children is called an *internal vertex.*



Subtree

The *subtree at vertex v* is the subgraph of the tree consisting of vertex v and its descendants and all edges incident to these descendants.

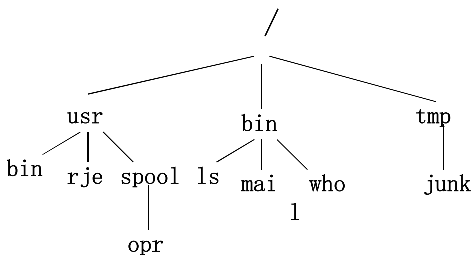


Trees as Models

Trees are used as models in such diverse areas as computer science, chemistry, geology, botany and psychology.

For example , Computer File Systems

- **A file system may be represented by a rooted tree**
- **the root represents the root directory**
- **internal vertices represent subdirectories**
- **leaves represent ordinary files or empty directories.**

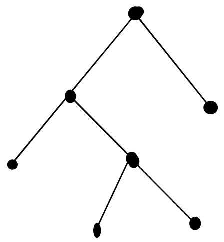


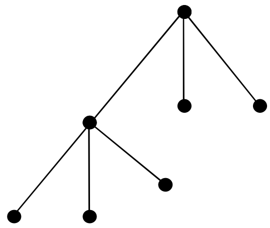
Binary Tree

【Definition 】 A rooted tree is called a *m-ary tree* if every internal vertex has no more than *m* children.

It is a *binary tree* if *m* = 2.

The tree is called a *full m-ary tree* if every internal vertex has exactly *m* children.





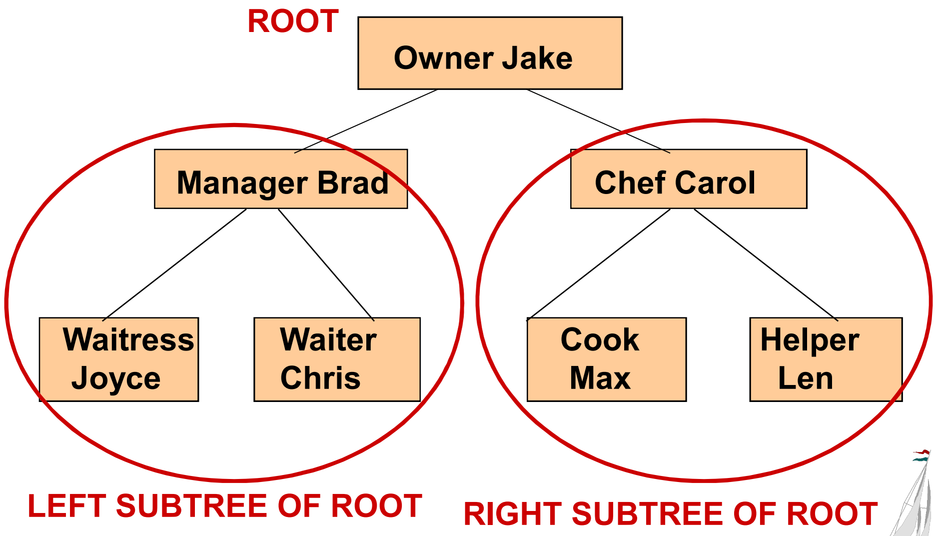
Ordered rooted tree

【Definition 】 An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.

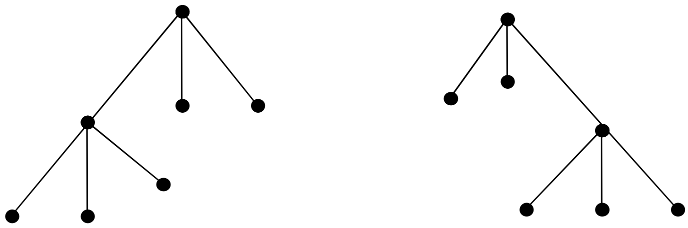
In an ordered binary tree, the two possible children of a vertex are called the *left child* and the *right child*, if they exist.

The tree rooted at the left child is called the *left subtree*, and that rooted at the right child is called the *right subtree*.

Left Subtree and right subtree

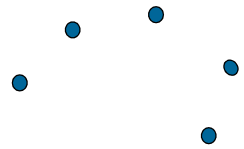


IS these two trees isomorphic?

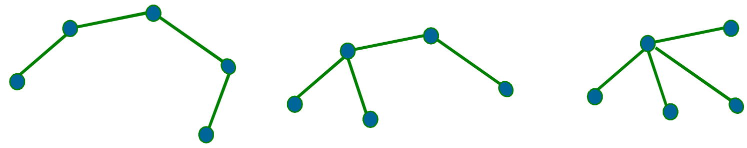


〖Example 3〗(1) How many nonisomorphic unrooted trees are there with n vertices if n=5 ?

Solution:



A tree must be connected and have no simple circuits, and have 4 edges.



〖Example 3〗(2) How many nonisomorphic rooted trees are there with n vertices if n=5 ?

Solution: **Nine**

Tree Properties

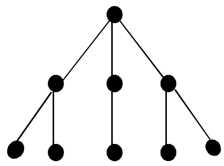
【 Theorem 2】 **A tree with n vertices has n-1 edges.**

Proof (1):

Choose the vertex r as the root of the tree.

We set up a one-to-one correspondence between the edges and the vertices other than r by associating the terminal vertex of an edge to that edge.

For example,



Since there are n-1 vertices other than r, there are n-1 edges in the tree.

Tree Properties

【 Theorem 2】 **A tree with n vertices has n-1 edges.**

Proof (2):

$$T=(V,E),|V|=n,|E|=e$$

Any tree must be planar and connected. Then

$$r=e-n+2$$

Any tree have no circuits. Then

$$r=1$$

It follows that,

$$e=n-1$$

〖Example 4〗 A tree has two vertices of degree 2, one vertex of degree 3, three vertices of degree 4. How many leafs does this tree has?

Solution:

Suppose that there are x leafs.

$v = 2+1+3+x$

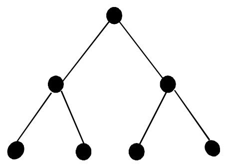
$e = \frac{1}{2}\,(2\times2 + 1\times3 + 3\times4 + x\times1) = v - 1$

$x = 9$

Tree Properties

【 Theorem 3】 A full m-ary tree with i internal vertices contains $n=mi+1$ vertices.

Proof :



Every vertex, except the root, is the child of an internal vertex.

Since each of the i internal vertices has m children, there are mi vertices in the tree other than the root.

Therefore, the tree contains $n=mi+1$ vertices.

Tree Properties

【Theorem 4】 **A full m-ary tree with**

- **n vertices has $i=(n-1)/m$ internal vertices and $l=[(m-1)n+1]/m$ leaves**
- **i internal vertices has $n=mi+1$ vertices and $l=(m-1)i+1$ leaves**
- **l leaves has $n=(ml-1)/(m-1)$ vertices and $i=(l-1)/(m-1)$ internal vertices**

Proof:

$$n = mi + 1$$
$$n = i + l$$

Note:

For a full binary tree, $l= i +1$, $e = v - 1$.

〖Example 5〗 A chain letter starts when a person sends a letter to five others. Each person who receives the letter either sends it to five other people who have never received it or does not send it to anyone. Suppose that 10000 person send out the letter before the chain ends and that no one receives more then one letter. How many people receive the letter, and how many do not send it out?

Solution:

The chain letter can be represented using a full 5-ary tree.

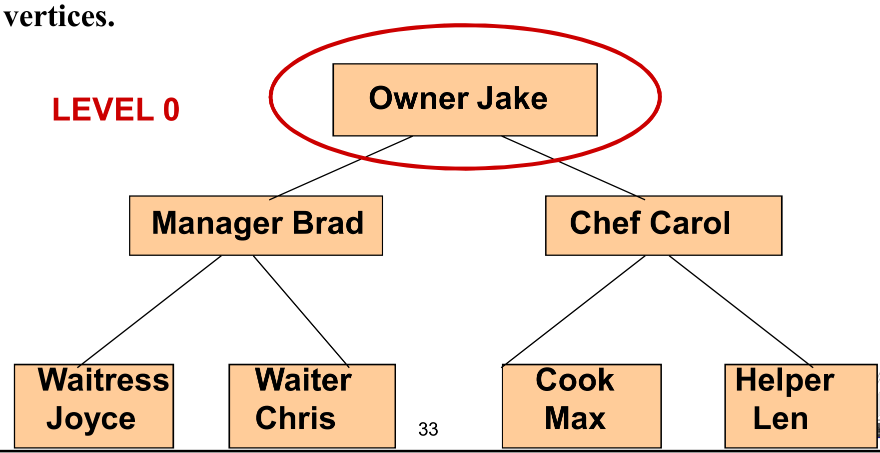
$$i = 10000$$
$$n = 5i + 1$$
$$n = i + l$$
$$l = 40001$$
$$n\text{-}1 = 50000$$

Level

The *level of vertex v* in a rooted tree is the length of the unique path from the root to *v*.

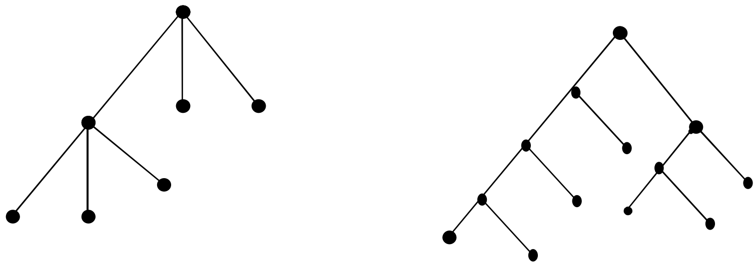
Height

The *height of a rooted tree* is the maximum of the levels of its vertices.



Balanced

A rooted *m*-ary tree of height *h* is called *balanced* if all its leaves are at levels *h* or *h*-1. （书上定义，我认为不合适）



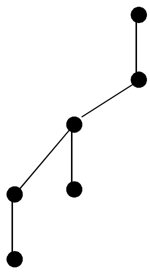
Balanced

A tree is balanced if:　（**Binary Tree)**

The left and right subtrees' heights differ by at most one, AND
The left subtree is balanced, AND
The right subtree is balanced

```
      A
     / \
    B   C
   /   / \
  D   E   F
     /
    G
```

```
    A
   / \
  B   C    <-- difference = 2
 /   /
D   E
   /
  G
```



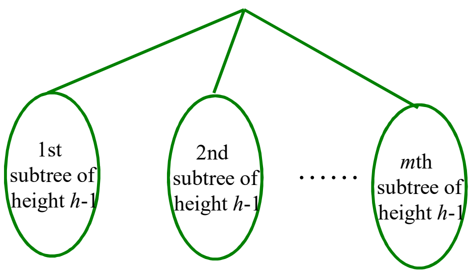
Tree Properties

【 Theorem 5】 There are at most m^h leaves in an m-ary tree of height h.

Proof:

(1) h=1

(2) Assume that the result is true for all m-ary tree of height less than h. Let T be an m-ary tree of height h.



Tree Properties

【 Corallary 】 **If an m-ary tree of height h has l leaves, then**

$$h \ge \lceil \log_m l \rceil .$$

If the m-ary tree is full and balanced, then

$$h = \lceil \log_m l \rceil .$$

Proof:

(1) $l \le m^h$

(2) Since the tree is balanced. Then each leaf is at level h or h-1, and since the height is h, there is at least one leaf at level h. It follows that,

$$\left.\begin{array}{l} m^{h-1} < l \\ l \le m^h \end{array}\right\} \Rightarrow \quad h-1 < \log_m l \le h$$

Question

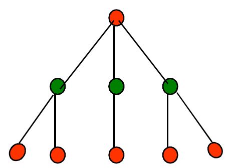
Every tree is a bipartite?

Yes.

Every tree can be colored using two colors.

Method:

We choose a root and color it red. Then we color all the vertices at odd levels blue and all the vertices at even levels red.



Homework:

Sec. 11.1 12, 20, 21, 28

CHAPTER 11 Trees

Problems:

- **How should items in a list be stored so that an item can be easily located?**

  Binary search trees

- **What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type?**

  Decision trees

- **How should a set of characters be efficiently coded by bit strings?**

  Prefix codes

1. Binary Search Trees

- **The problem of search**
- **The concept of binary search tree**
- **How to construct a binary search tree**
- **Binary search tree algorithm**
- **The computational complexity**

☸The Concept of Binary Search Trees

- A binary search tree can be used to store items in its vertices. It enables efficient searches.
- *Binary search tree*
  - An ordered rooted binary tree
  - Each vertex contains a distinct key value
  - The key values in the tree can be compared using "greater than" and "less than", and
  - The key value of each vertex in the tree is less than every key value in its right subtree, and greater than every key value in its left subtree.

☸ Construct the binary search tree

- The shape of a binary search tree depends on its key values and their order of insertion.

For example, Insert the elements 'J' 'E' 'F' 'T' 'A' in that order.

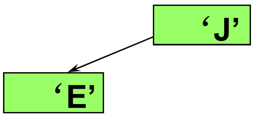
-- The first value to be inserted is put into the root.

'J'

-- Inserting ‘E’ into the BST

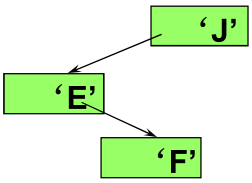
Thereafter, each value to be inserted begins by comparing itself to the value in the root, moving left it is less, or moving right if it is greater.

This continues at each level until it can be inserted as a new leaf.



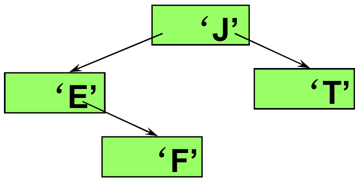
-- Inserting ‘F’ into the BST

Begin by comparing ‘F’ to the value in the root, moving left it is less, or moving right if it is greater. This continues until it can be inserted as a leaf.



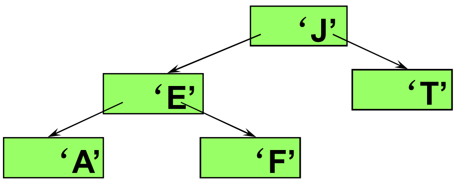
-- Inserting 'T' into the BST

Begin by comparing 'T' to the value in the root, moving left it is less, or moving right if it is greater. This continues until it can be inserted as a leaf.



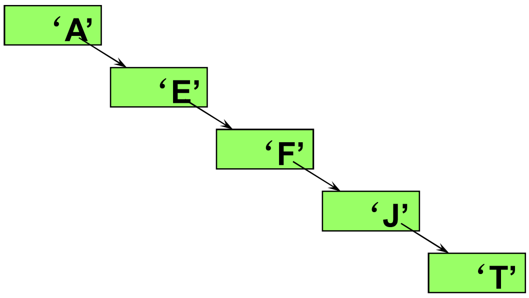
--Inserting 'A' into the BST

Begin by comparing 'A' to the value in the root, moving left if it is less, or moving right if it is greater. This continues until it can be inserted as a leaf.

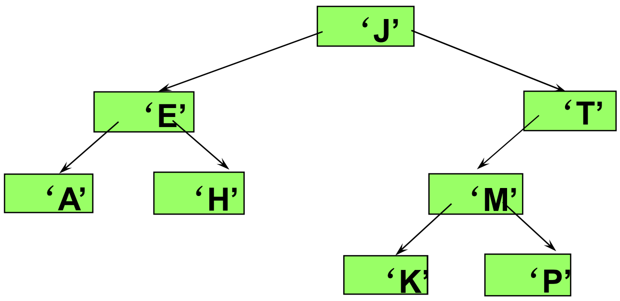


What binary search tree . . .

is obtained by inserting the elements ‘A’ ‘E’ ‘F’ ‘J’ ‘T’ in that order.



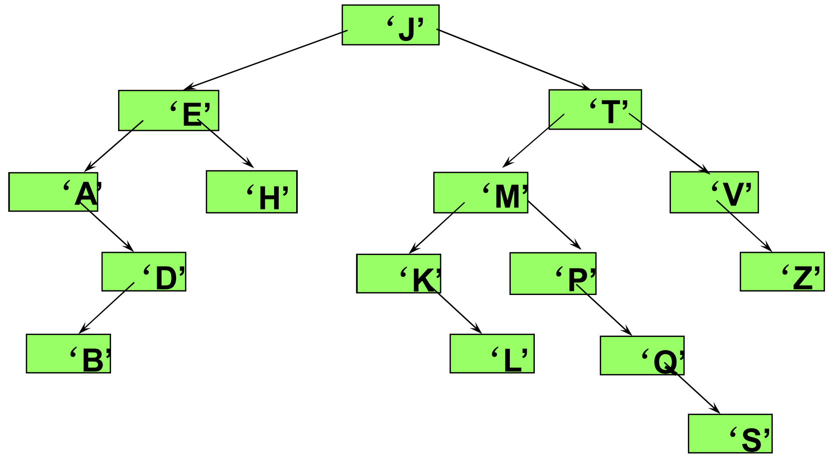
Another binary search tree



Add nodes containing these values in this order:

‘D’ ‘B’ ‘L’ ‘Q’ ‘S’ ‘V’ ‘Z’

Is ‘F’ in the binary search tree?



❁ Binary Search Tree Algorithm

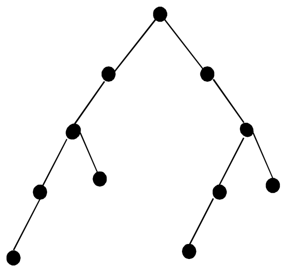
Algorithm 1 Locating and adding items to a binary search tree

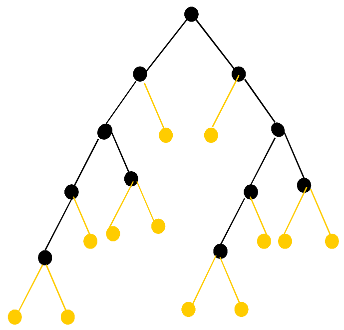
Procedure insertion (T: binary search tree, x: item)
v:=root of T
While $v \neq$ null *and* label(v) $\neq x$
Begin
  if x<label(v) *then*
    if left child of $v \neq$null *then* v:=left child of v
    else add new vertex as a left child of v and set v:=null
  else
    if right child of $v \neq$null *then* v:=right child of v
    else add new vertex as a right child of v and set v:=null
end
if root of T =null *then* add a vertex r to the tree and label it with x
else if v is null *or* label(v) $\neq x$ *then* label new vertex with x and let v be this new vertex
{ v = location of x}

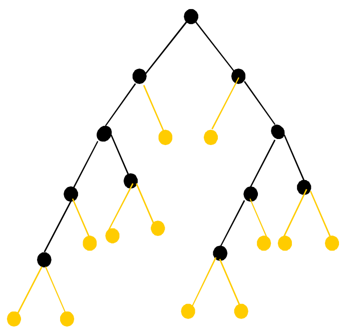
☸ The computational complexity

Suppose we have a binary search tree *T* for a list of *n* items.

We can form a full binary tree *U* from *T* by adding unlabeled vertices whenever necessary so that every vertex with a key has two children.





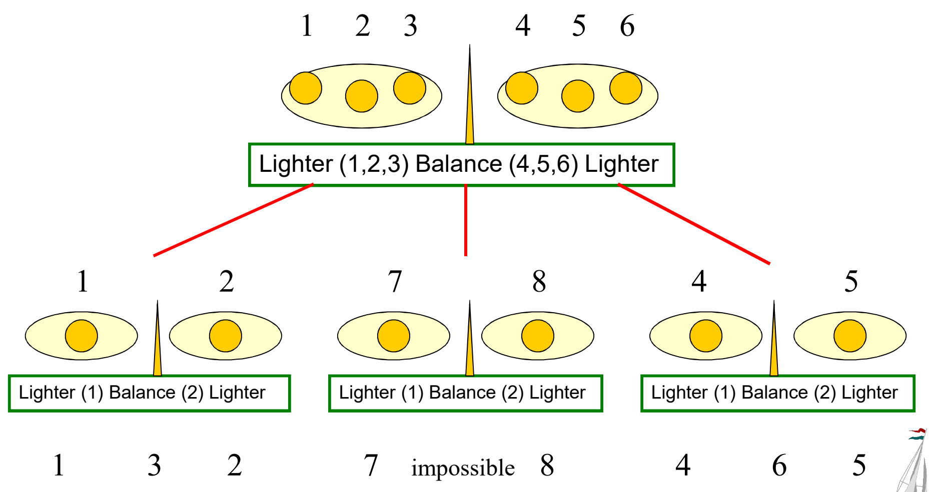


- **The most comparisons needed to add a new item is the length of the longest path in *U* from the root to a leaf.**
- **If a binary search tree is balanced, locating or adding an item requires no more than $\lceil \log(n+1) \rceil$ comparisons.**

2. Decision Trees

- **Rooted trees can be used to model problems in which a series of decisions leads to a solution.**
- **A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a *decision tree*.**

An example: Counterfeit coin detection



Homework:

Sec.11.2 8