Trees

Chapter 11

Chapter Summary

- Introduction to Trees
- Applications of Trees
- Tree Traversal
- Spanning Trees
- Minimum Spanning Trees

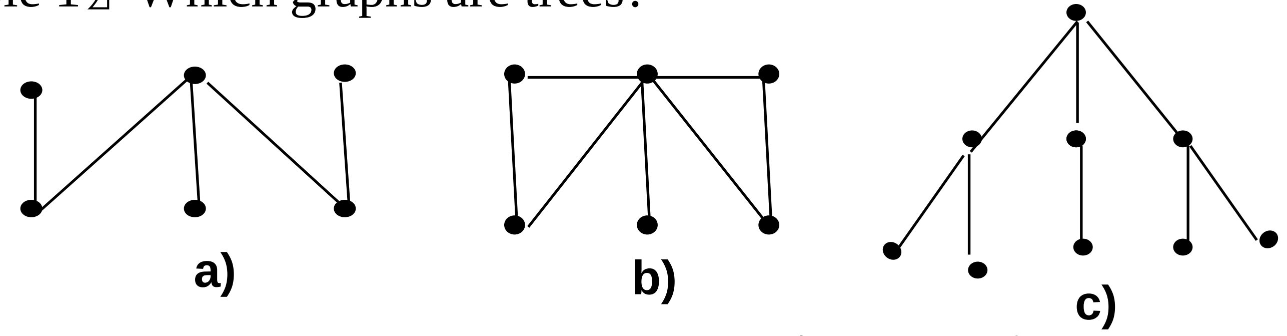
Introduction to Trees

Section 11.1

Tree

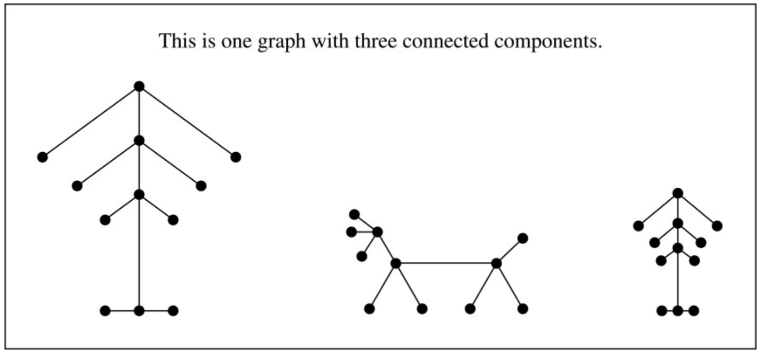
Definition: A *tree* is a connected undirected graph with no simple circuits.

〖Example 1〗 Which graphs are trees?



Note: Any tree must be a simple graph.

Definition: A *forest* is a graph that has no simple circuit, but is not connected. Each of the connected components in a forest is a tree.


This is one graph with three connected components.

【 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) $\Rightarrow$

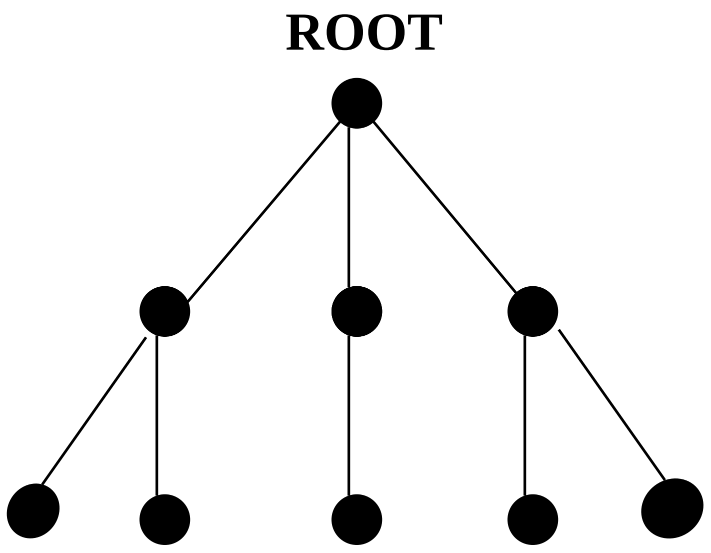
- ✓ there is a simple path between any two of its vertices
- ✓ unique

(2) $\Leftarrow$

- ✓ connected
- ✓ no simple circuits

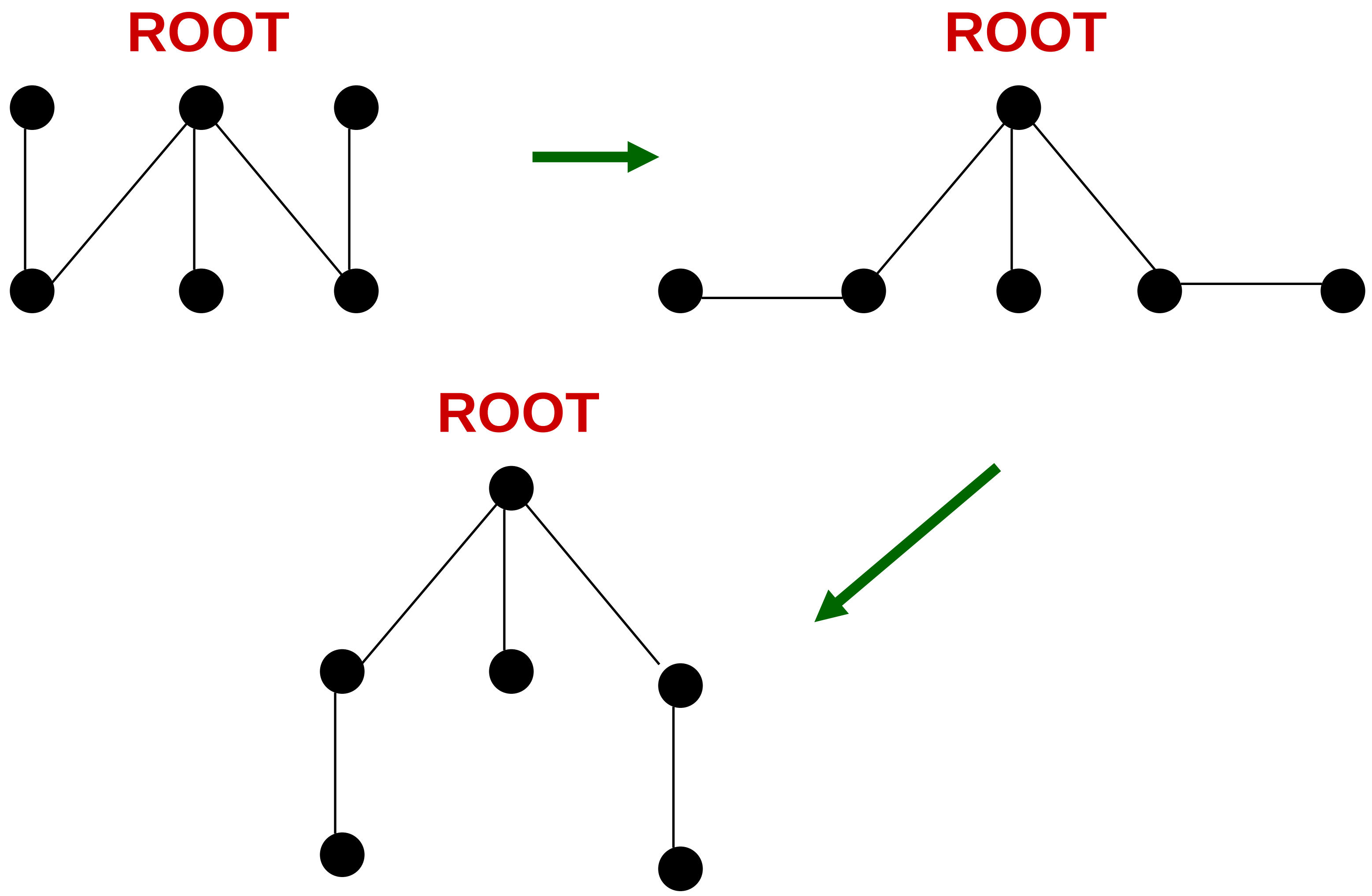
Root tree

Definition: A *rooted tree* is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

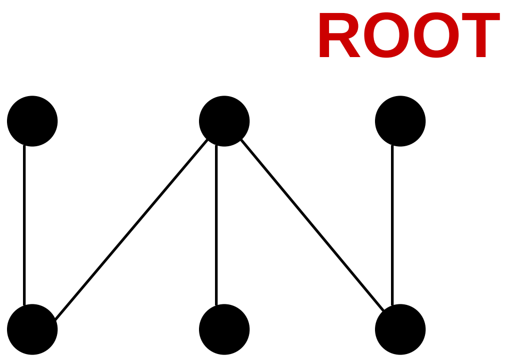


An unrooted tree is converted into different rooted trees when different vertices are chosen as the root.

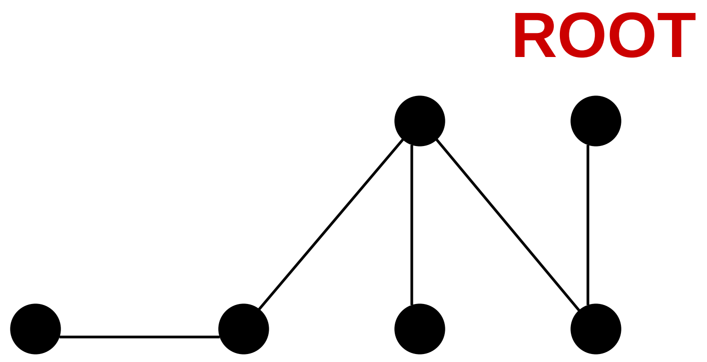
〖Example 2〗Change an unrooted tree into a rooted tree.



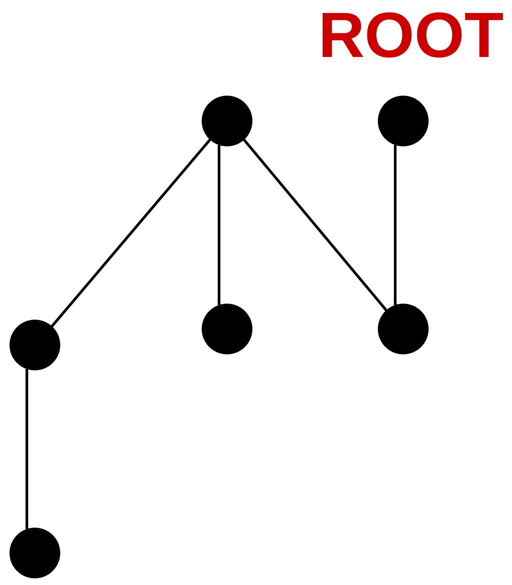
Problem: What if a different root is chosen?



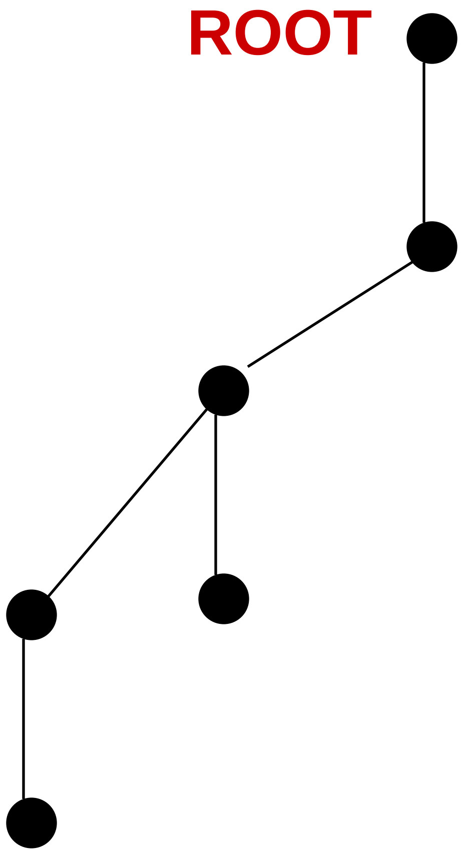
Problem: What if a different root is chosen?



Problem: What if a different root is chosen?

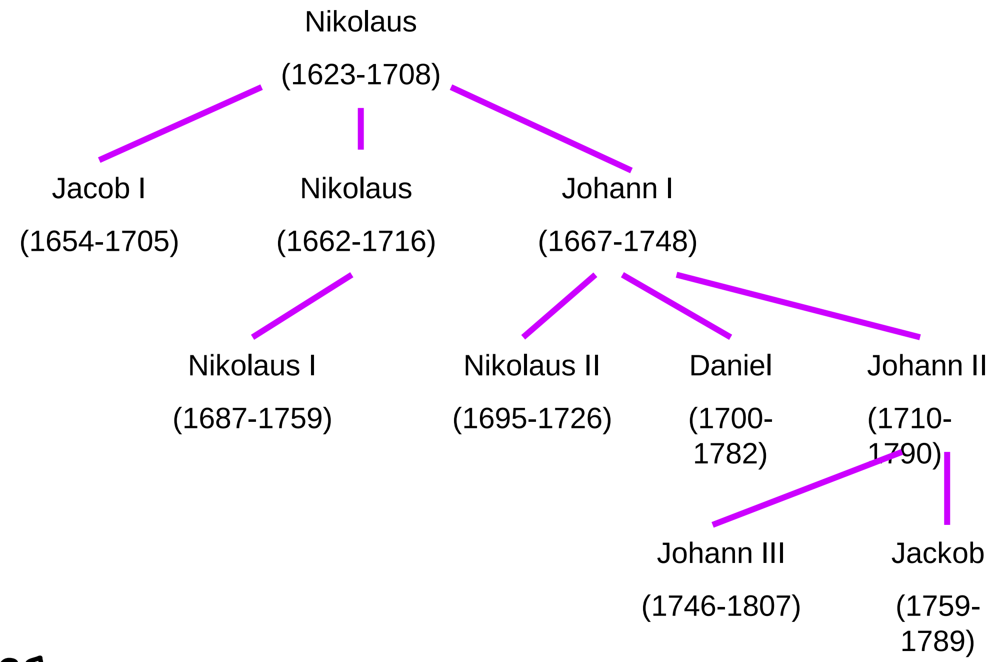


Problem: What if a different root is chosen?



Rooted Tree Terminology

Terminology for rooted trees is a mix from botany and genealogy (such as this family tree of the Bernoulli family of mathematicians).

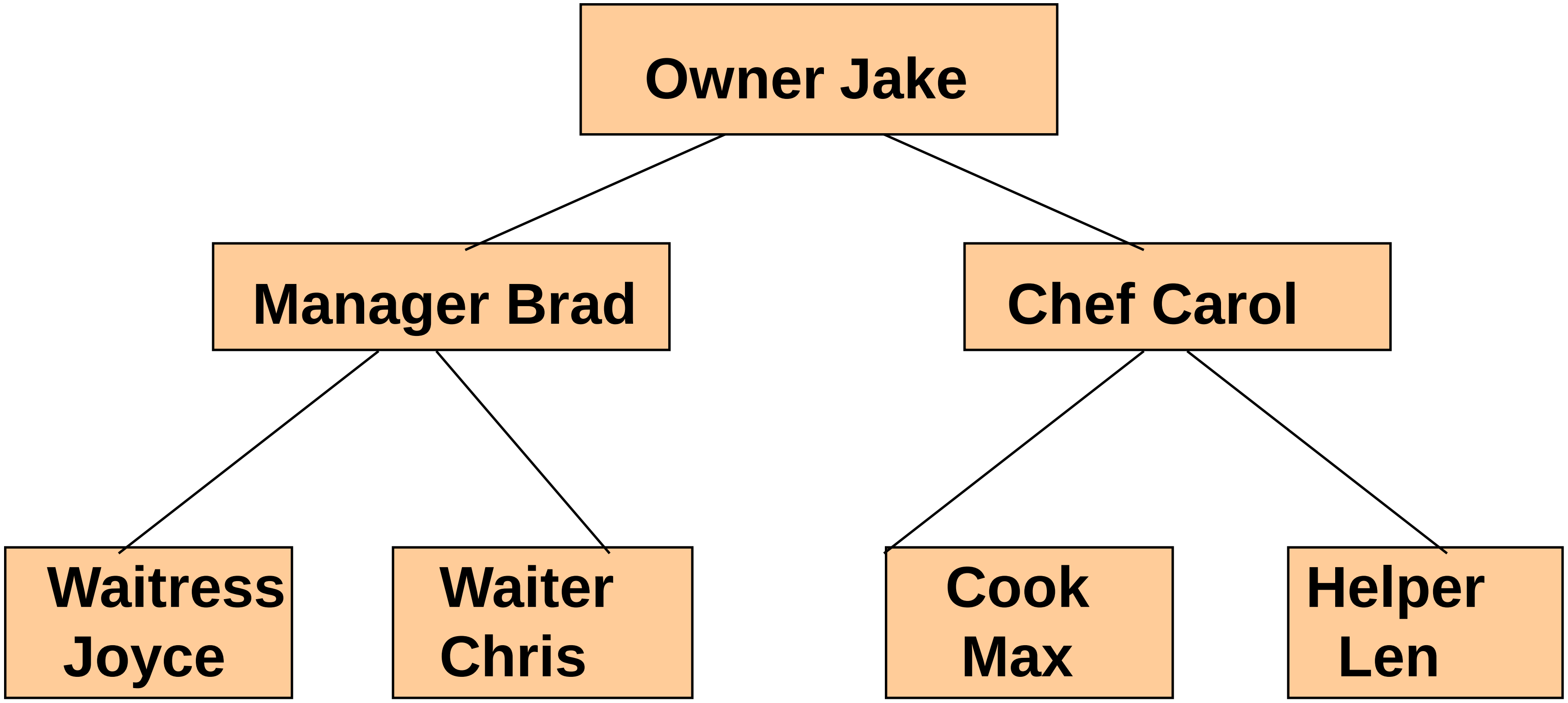


The Bernoulli family of mathematicians

Rooted Tree Terminologies

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

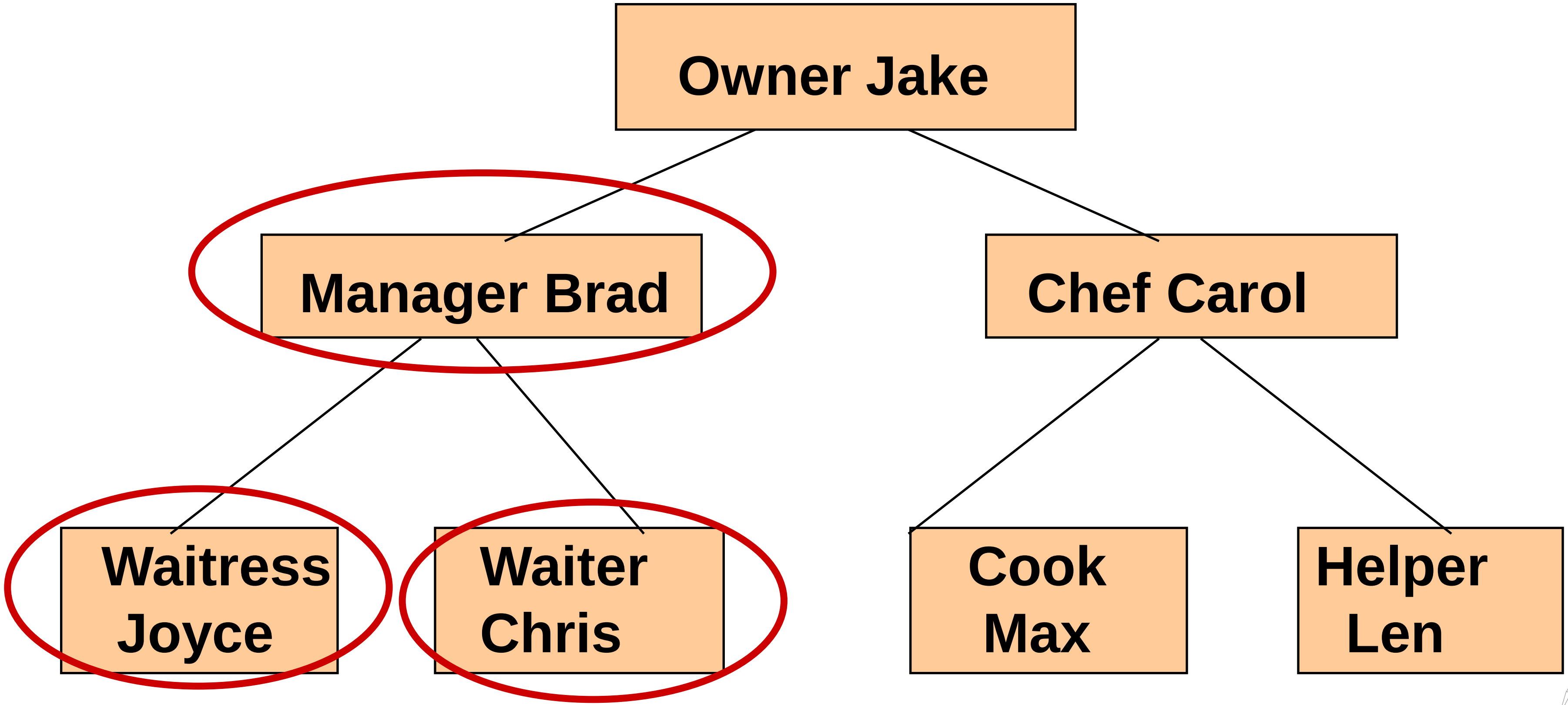
An Example: Jake’s Pizza Shop Tree



Parent VS. Child

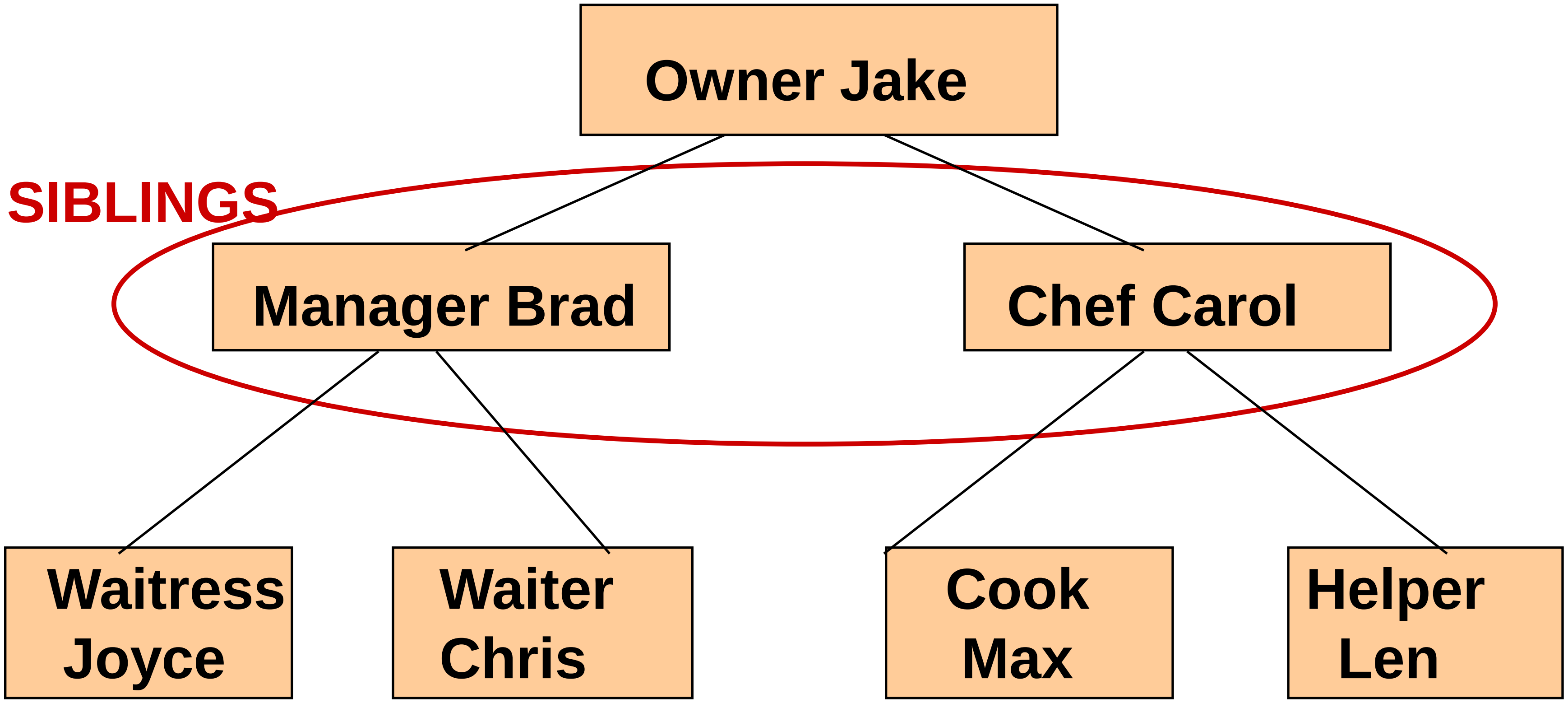
The parent of a non-root vertex v is the unique vertex u with a directed edge from u to v.

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



Sibling

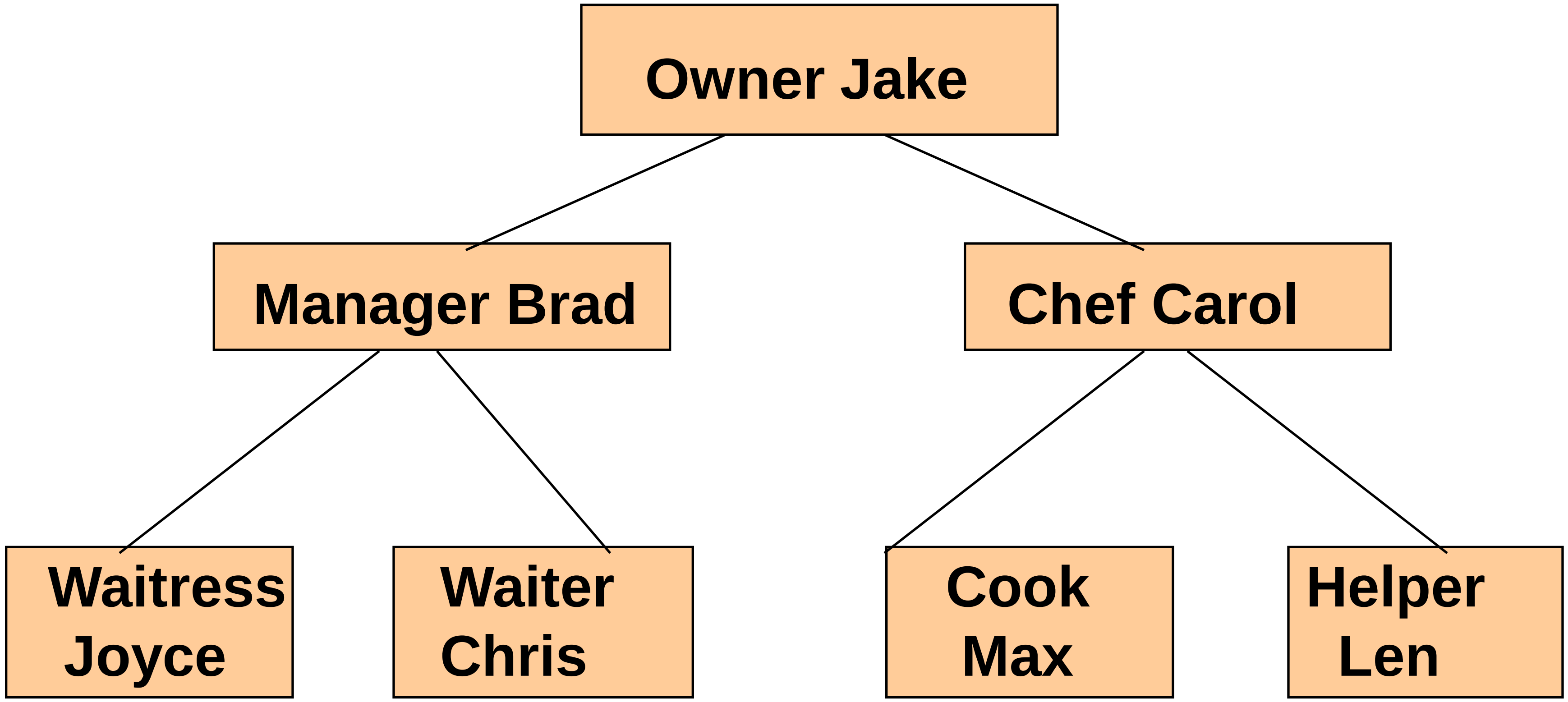
Vertices with the same parent are called siblings .

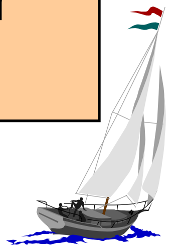


Ancestors VS. Descendants

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

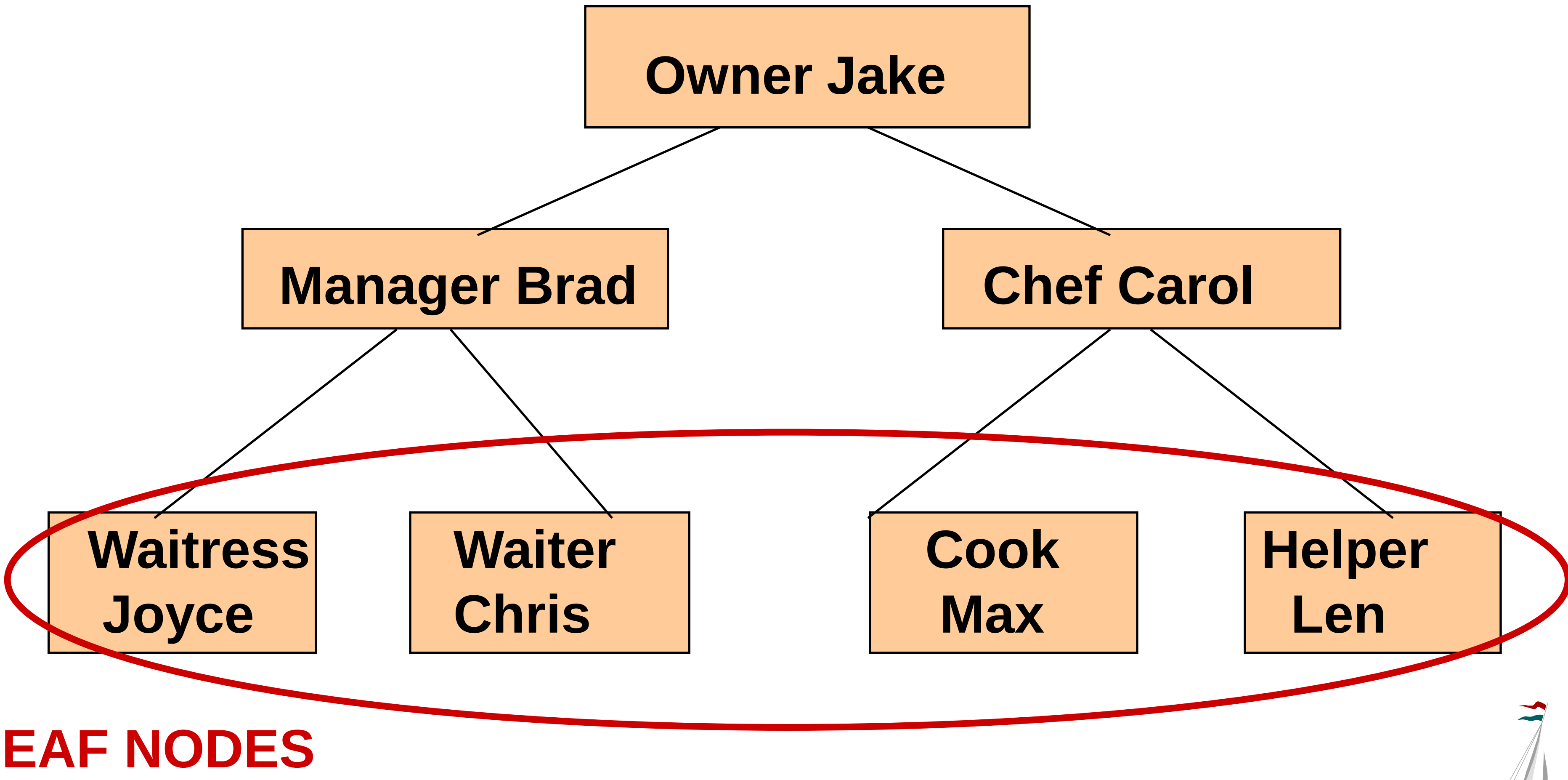
The **descendants of vertex v** are all the vertices that have v as an ancestor.





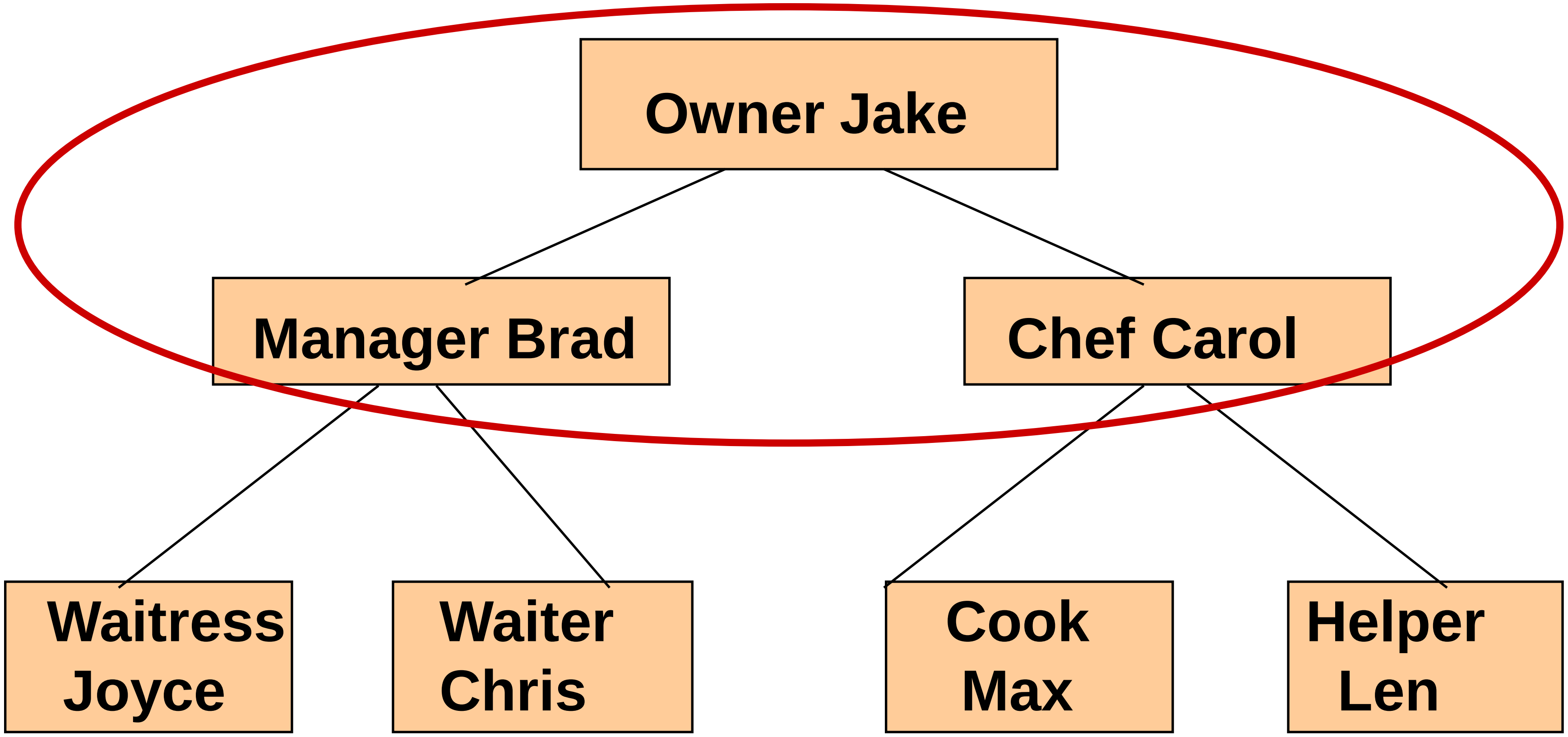
Leaf

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



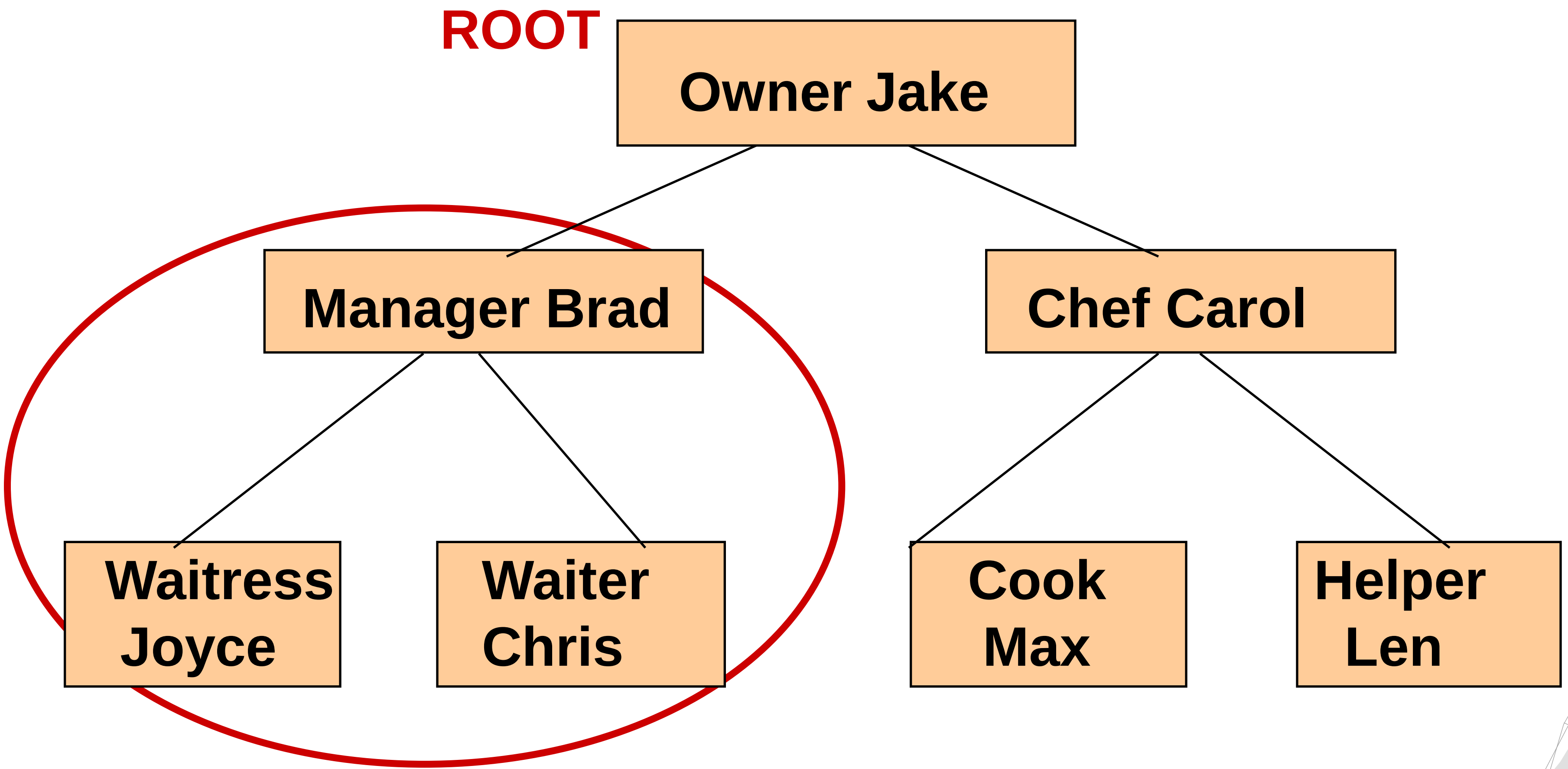
Internal Vertex

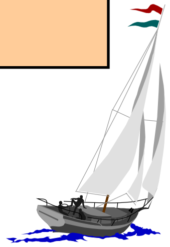
A vertex that have children is called an internal vertex.



Subtree

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

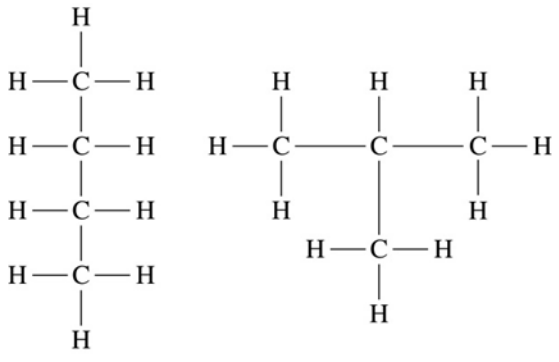




Trees as Models

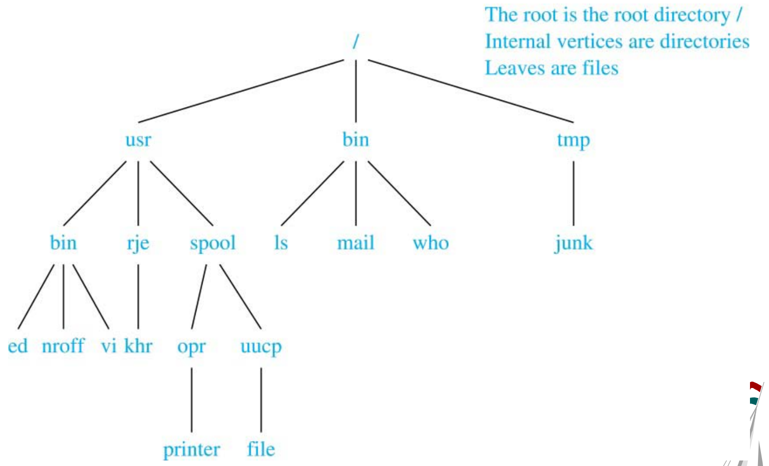
Trees are used as models in computer science, chemistry, geology, botany, psychology, and many other areas.

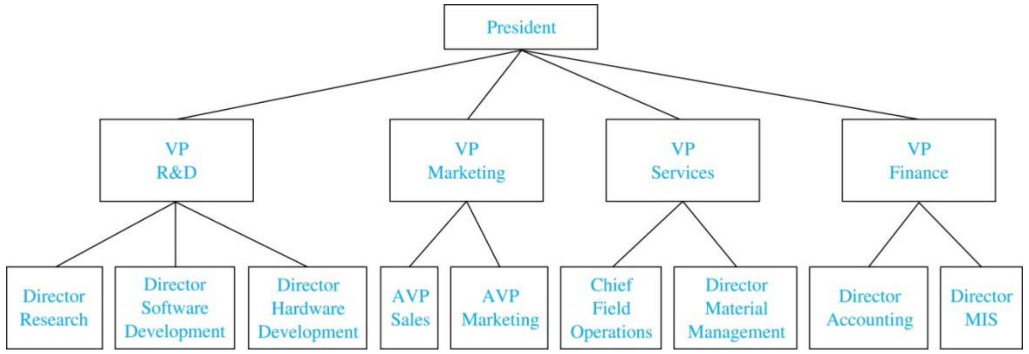
- **Trees were introduced by the mathematician Cayley in 1857 in his work counting the number of isomers of saturated hydrocarbons. The two isomers of butane are shown at the right.**
- **The organization of a computer file system into directories, subdirectories, and files is naturally represented as a tree.**
- **Trees are used to represent the structure of organizations.**



Butane

Isobutane



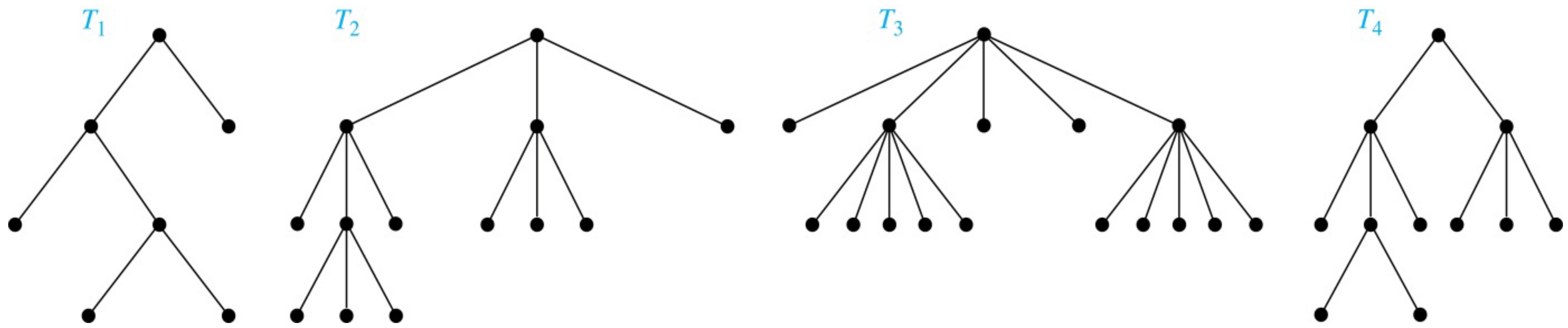


m-ary Rooted Trees

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

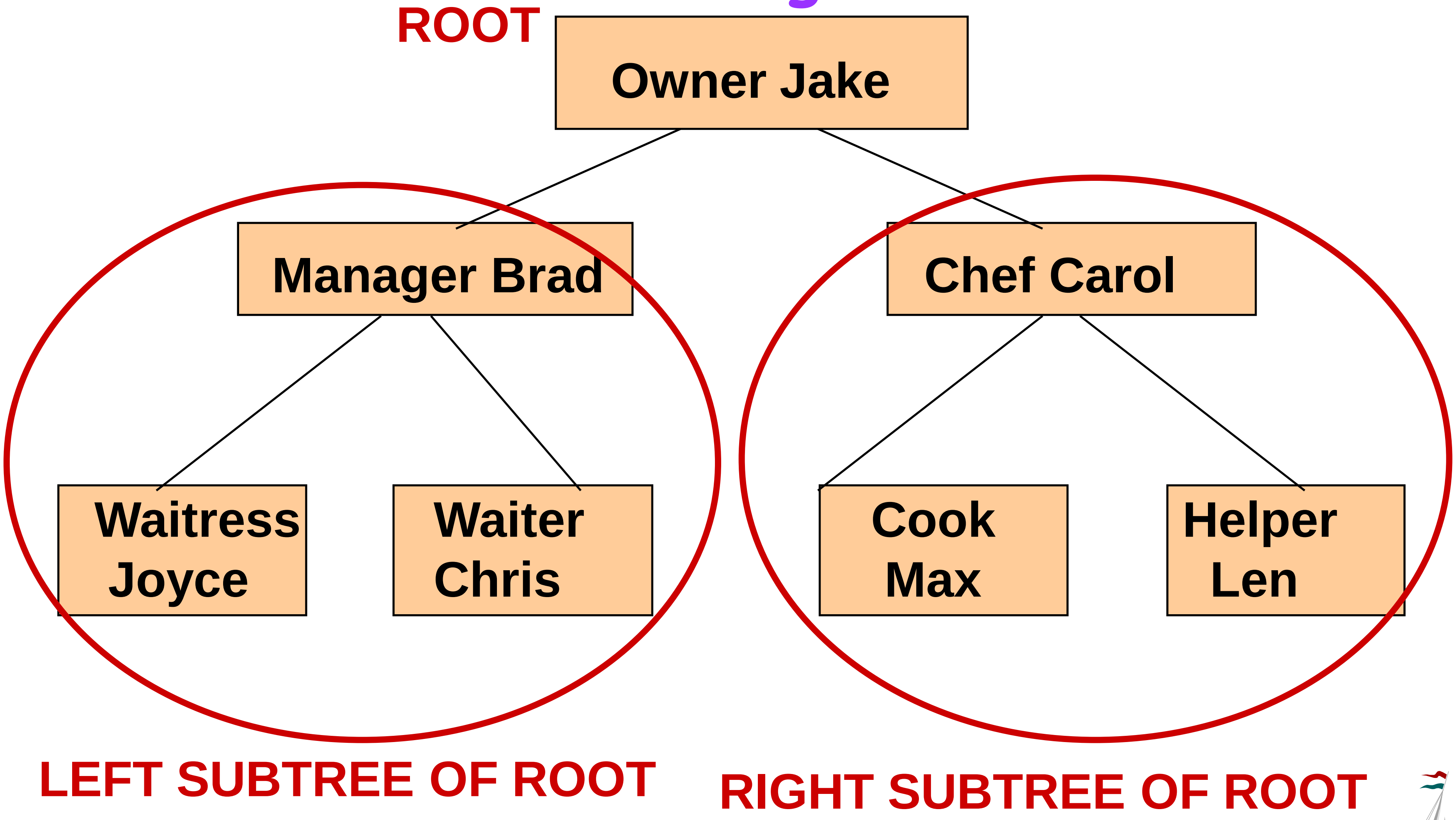
Definition: **An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.**

- We draw ordered rooted trees so that the children of each internal vertex are shown in order from left to right.

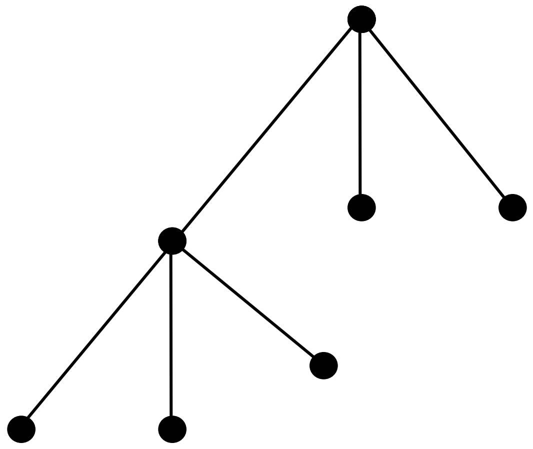
In an ordered binary tree, the two possible children of a vertex are called the left child and the right child, if they exist.

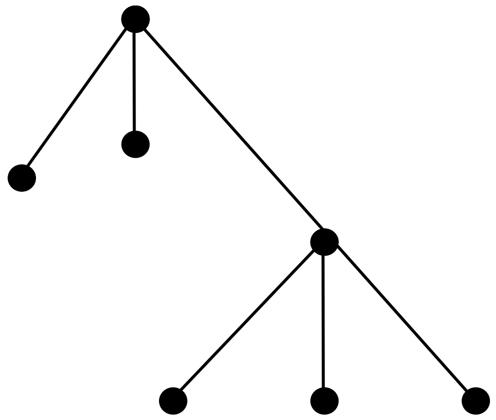
The tree rooted at the left child is called the left subtree, and that rooted at the right child is called the right subtree.

Left Subtree and right subtree



IS these two trees isomorphic?





Properties of Trees

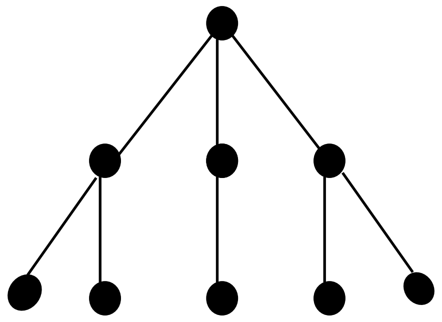
【 Theorem 2】 A tree with *n* vertices has *n*-1 edges.

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.

Properties of Trees

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

Proof (2):

$$T=(V,E),|V|=n,|E|=e \qquad \Rightarrow \qquad e=n-1$$

Any tree must be planar and connected. Then

$$r=e-n+2$$

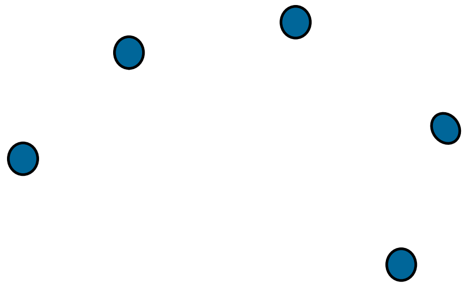
Any tree have no circuits. Then

$$r=1$$

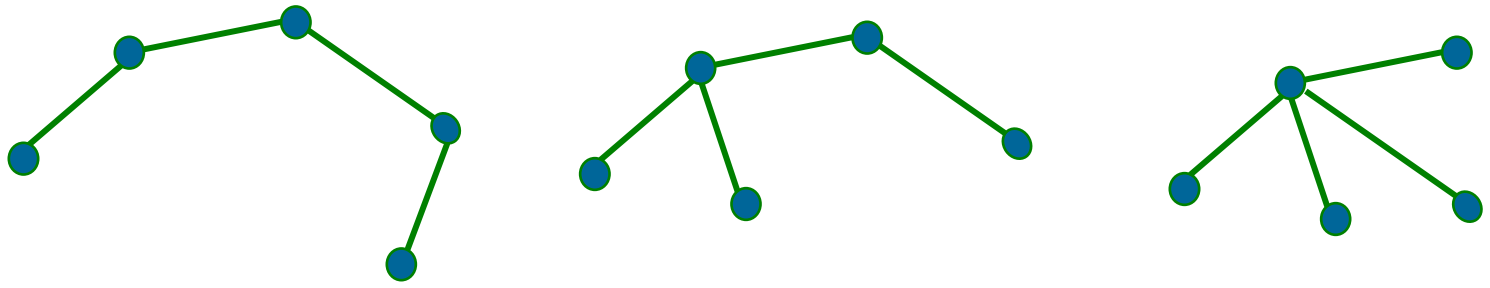
It follows that, $e=n-1$

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

Solution:

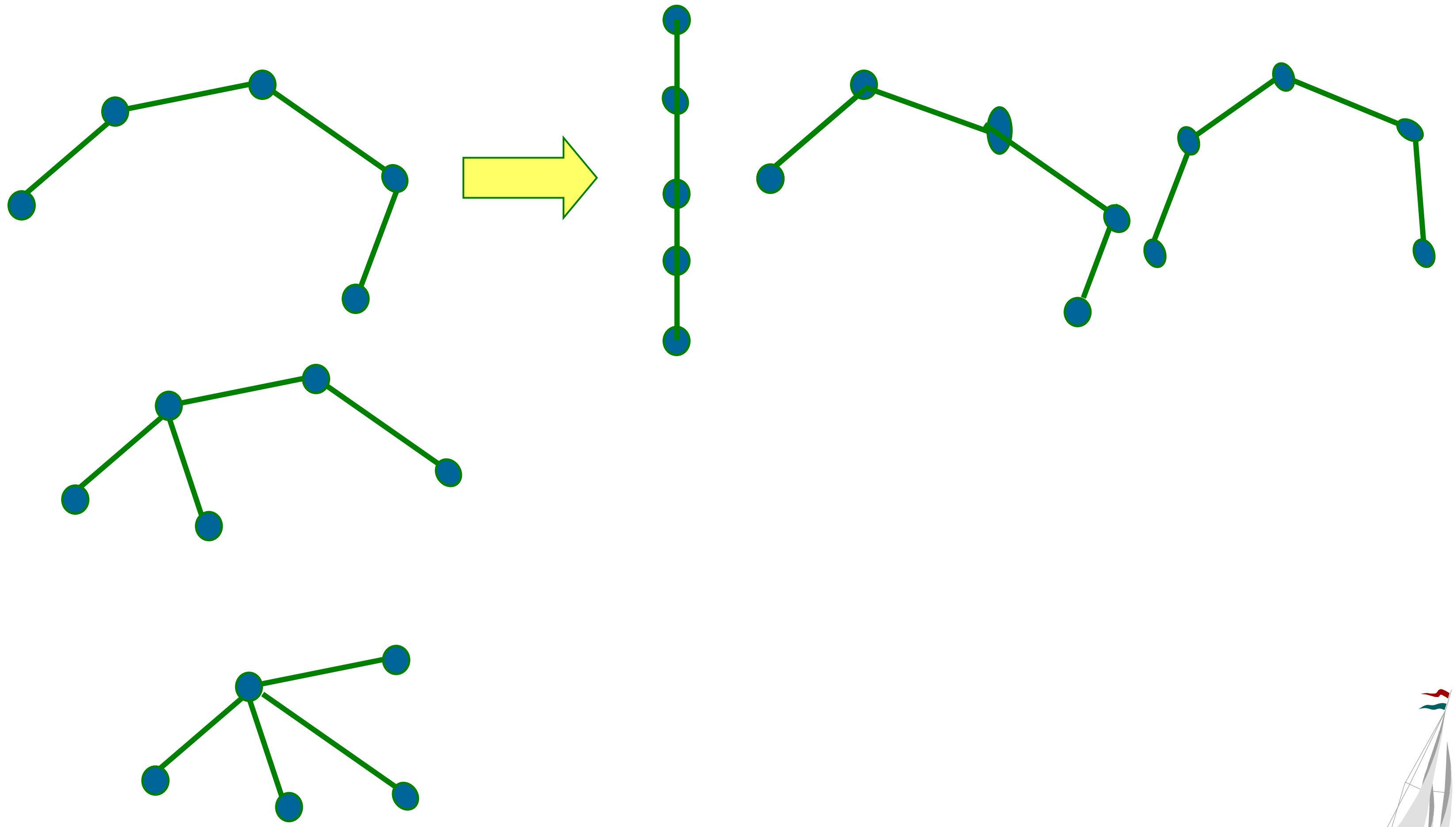


A tree must be connected and have no simple circuits, and have 4 edges.



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

Solution:



〖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.

$v = 2+1+3+x$

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

$x = 9$

Question:

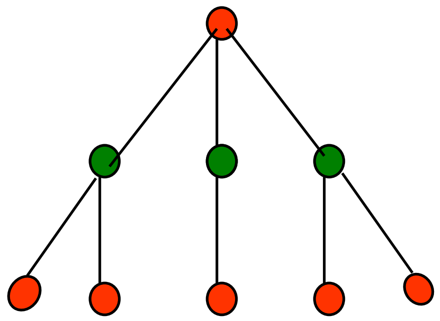
Every tree is a bipartite?

Yes.

Every tree can be colored using two colors.

Method:

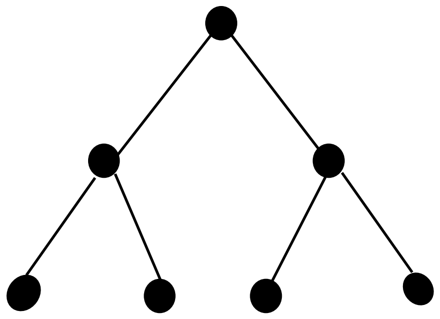
We choose a root and color it red. Then we color all the vertices at odd levels blue and all the vertices at even levels red.



Counting Vertices in Full m-Ary Trees

【 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.

Counting Vertices in Full m-Ary Trees (continued)

【 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 $\underline{l=(m-1)i+1 \text{ leaves}}$
- *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.

$i = 10000$

$n = 5i + 1$

$n = i + l$

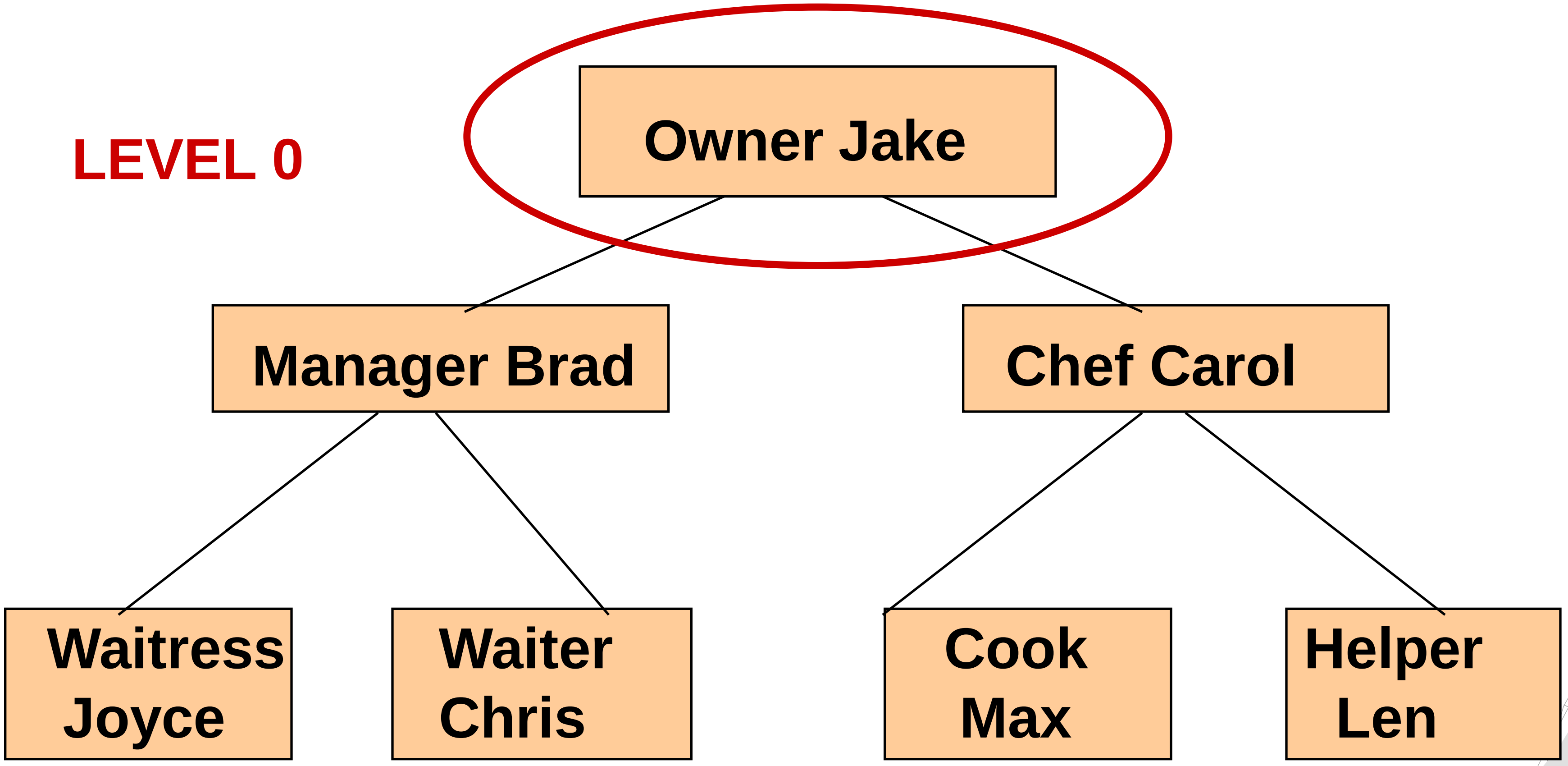
$l = 40001$

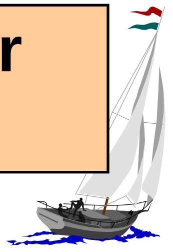
$n\text{-}1 = 50000$

Level of vertices and height of trees

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

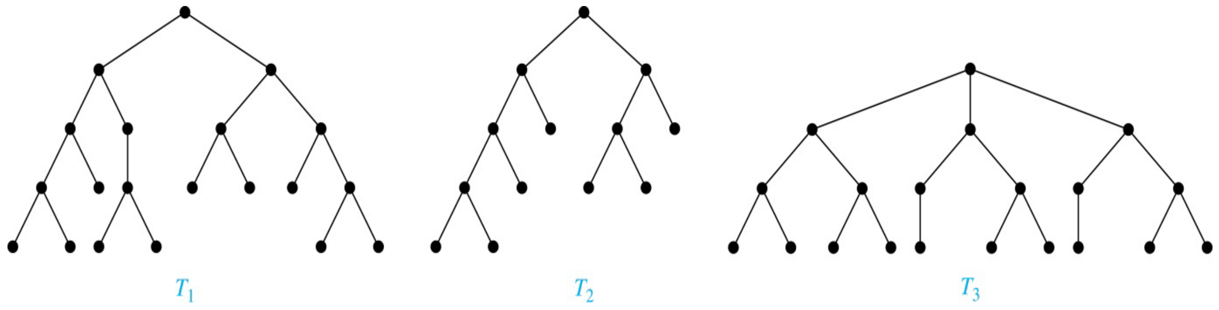
The **height of a rooted tree** is the maximum of the levels of its vertices.





Balanced m-Ary Trees

A rooted m-ary tree of height h is called balanced if all its leaves are at levels h or h-1.



T_1 and T_3 are balanced.
T_2 is not because it has leaves at levels 2, 3, and 4.

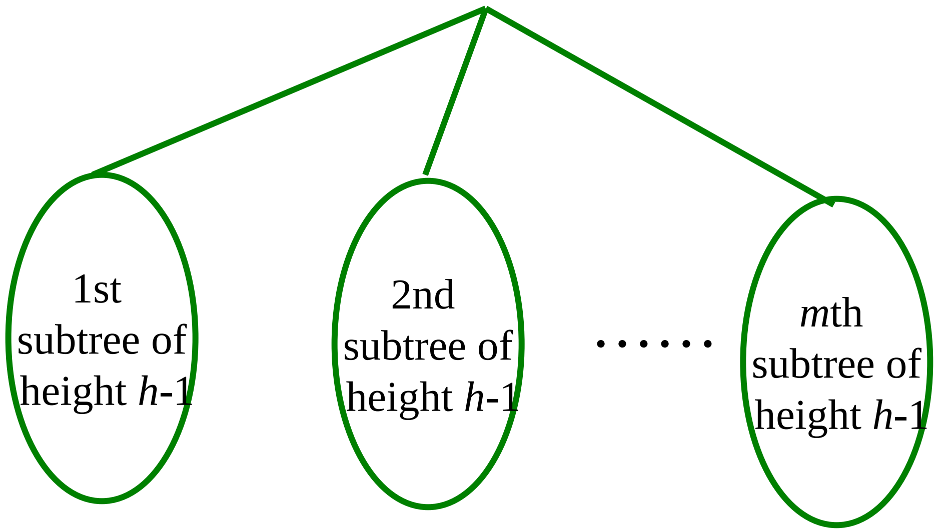
The Bound for the Number of Leaves in an m-Ary Tree

【 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.


1st subtree of height h-1
2nd subtree of height h-1
……
mth subtree of height h-1

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

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

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

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

Proof:

(1) $l \leq m^h$

(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,

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

Homework:

Seventh Edition:

P. 755 12, 20, 21, 28

Applications of Trees

Section 11.2

Problem:

- 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

The Concept of Binary Search Trees

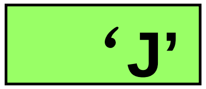
- A binary search tree can be used to store items in its vertices. It enables efficient searches.
- Binary search tree
  - An ordered rooted binary tree
  - Each vertex contains a distinct key value
  - The key values in the tree can be compared using "greater than" and "less than", and
  - The key value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

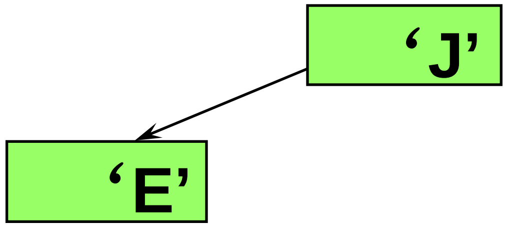
-- The first value to be inserted is put into the root.



-- Inserting ‘E’ into the BST

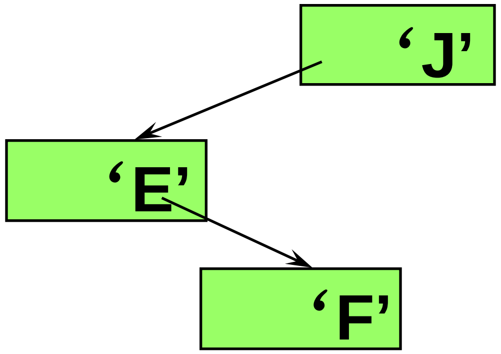
Thereafter, each value to be inserted begins by comparing itself to the value in the root, moving left it is less, or moving right if it is greater.

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



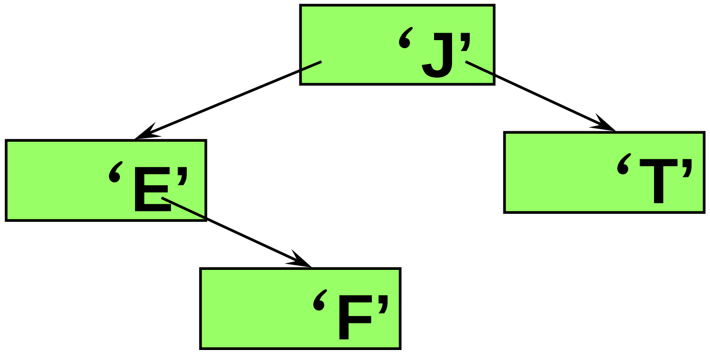
-- Inserting ‘F’ into the BST

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.



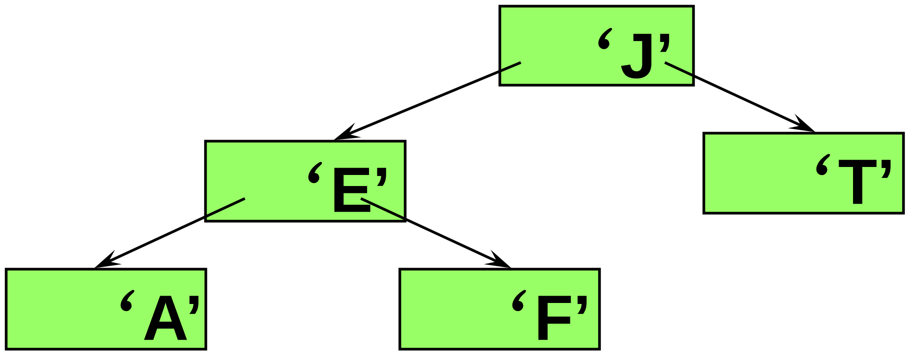
-- Inserting ‘T’ into the BST

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.


$$\begin{tikzcd} & \boxed{\text{‘J’}} \arrow[dl] \arrow[dr] & \\ \boxed{\text{‘E’}} \arrow[dr] & & \boxed{\text{‘T’}} \\ & \boxed{\text{‘F’}} & \end{tikzcd}$$

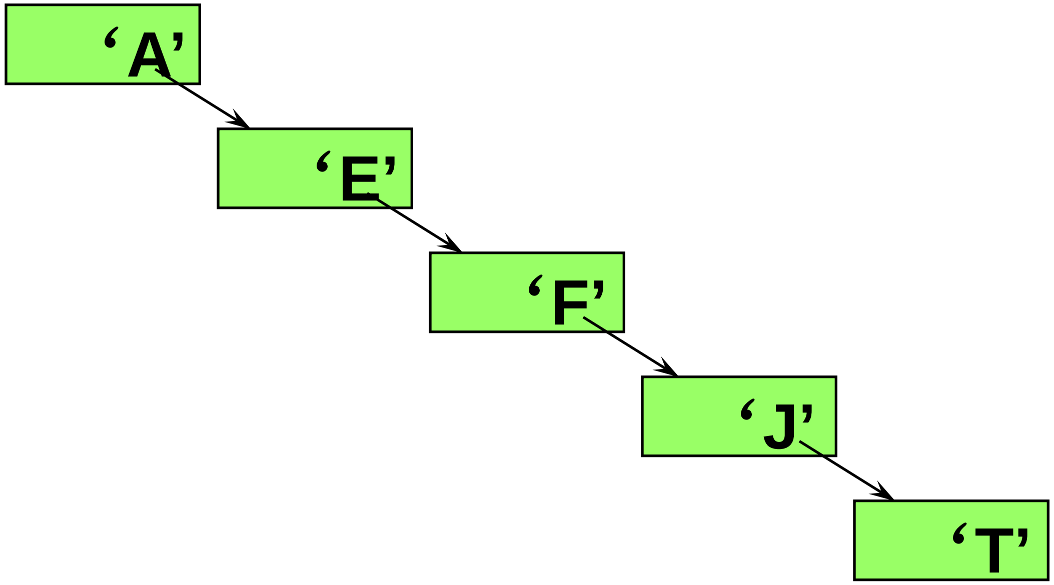
--Inserting ‘A’ into the BST

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.

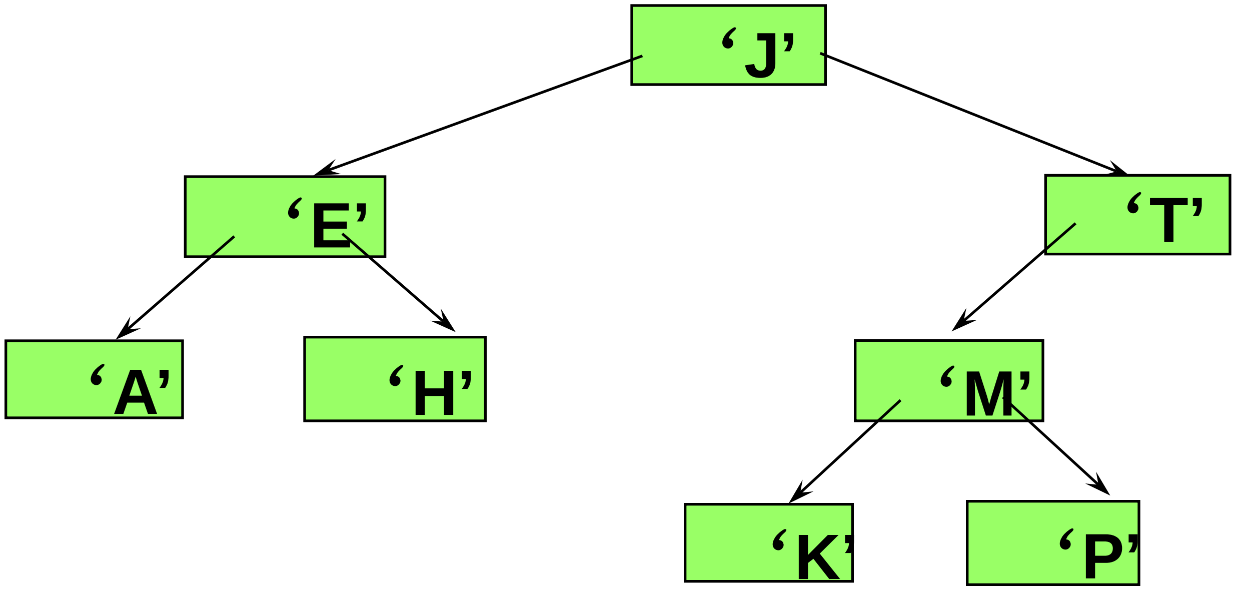

$$\begin{tikzcd}
 & & \text{‘J’} \arrow[dl] \arrow[dr] & \\
 & \text{‘E’} \arrow[dl] \arrow[dr] & & \text{‘T’} \\
\text{‘A’} & & \text{‘F’} &
\end{tikzcd}$$

What binary search tree . . .

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



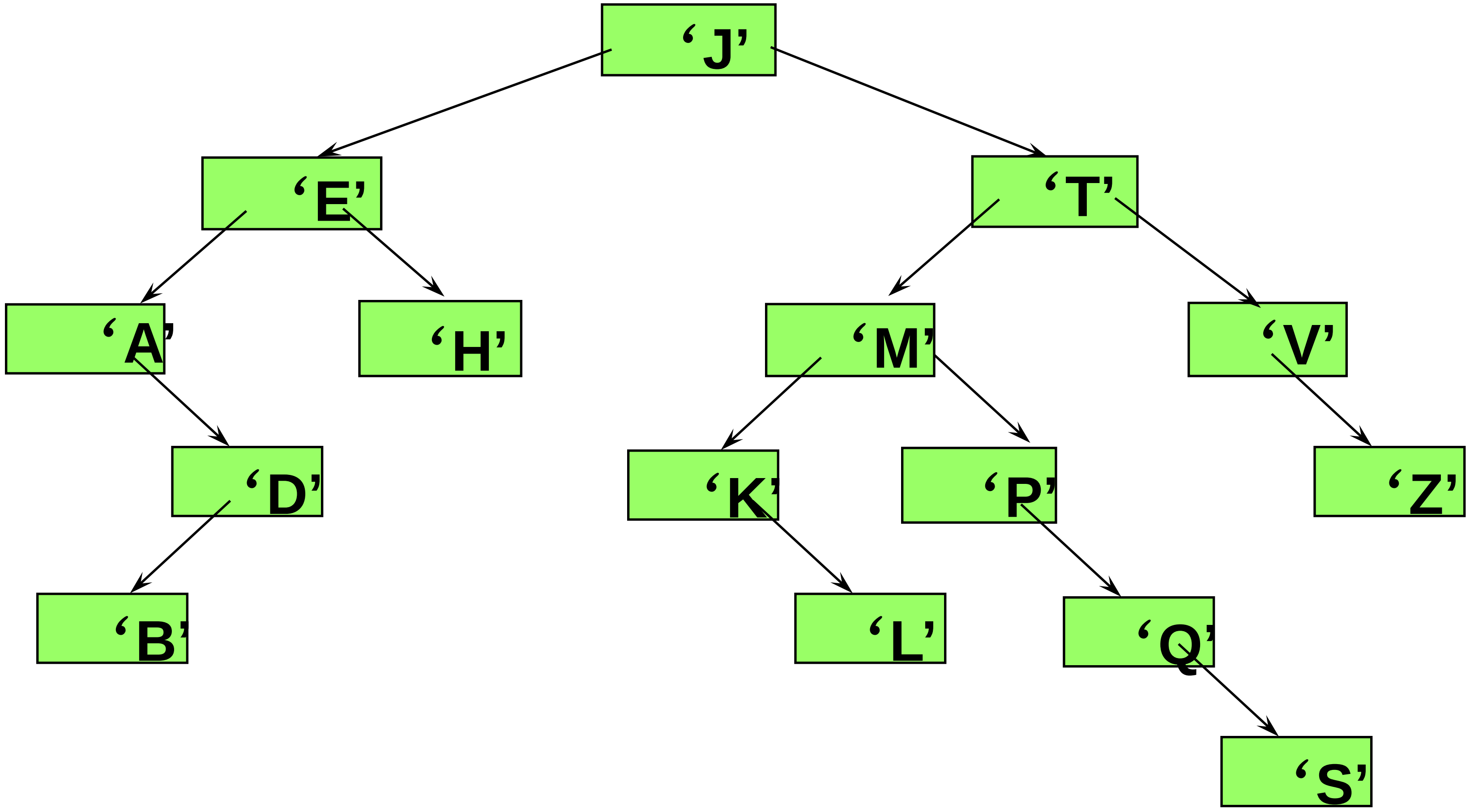
Another binary search tree



Add nodes containing these values in this order:

‘D’ ‘B’ ‘L’ ‘Q’ ‘S’ ‘V’ ‘Z’

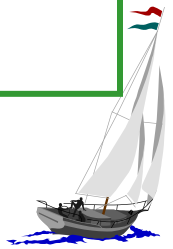
Is ‘F’ in the binary search tree?



Binary Search Tree Algorithm

Algorithm 1 Binary Search Tree Algorithm

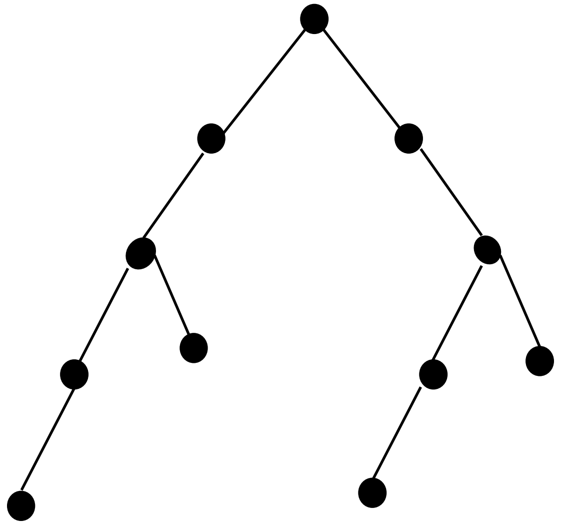
```
Procedure insertion ($T$: binary search tree, $x$: item)
$v$:=root of $T$
While  $v \neq$null and  label($v$) $\neq x$
 Begin
    if $x<$label($v$) then
        if left child of $v \neq$null then $v$:=left child of $v$
       else  add new vertex as a left child of $v$ and set $v$:=null
    else
       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 $x$}
```

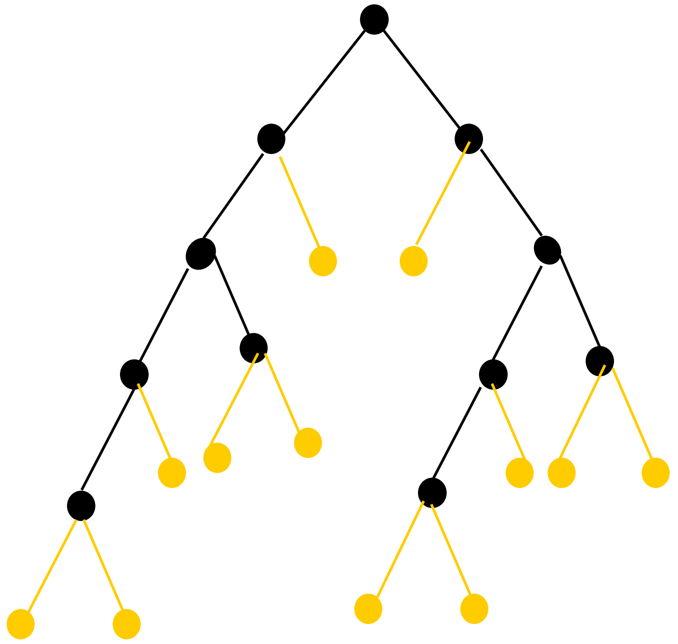


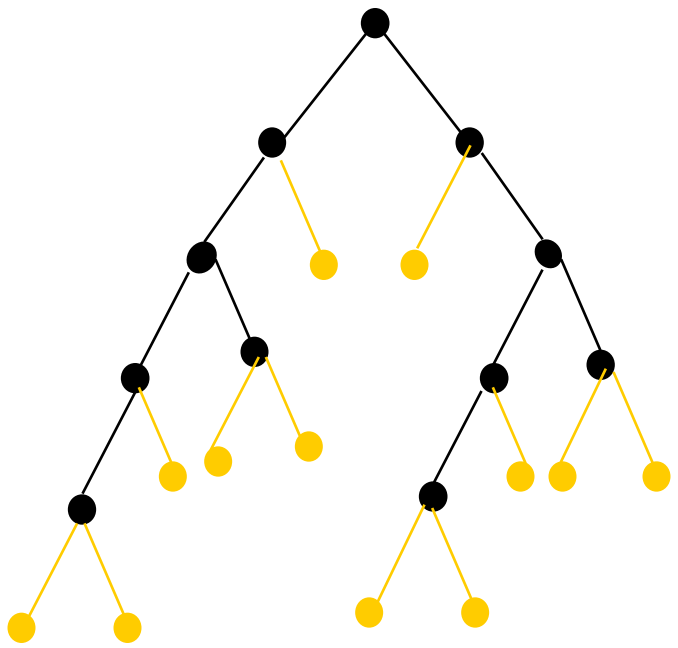
The computational complexity

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.







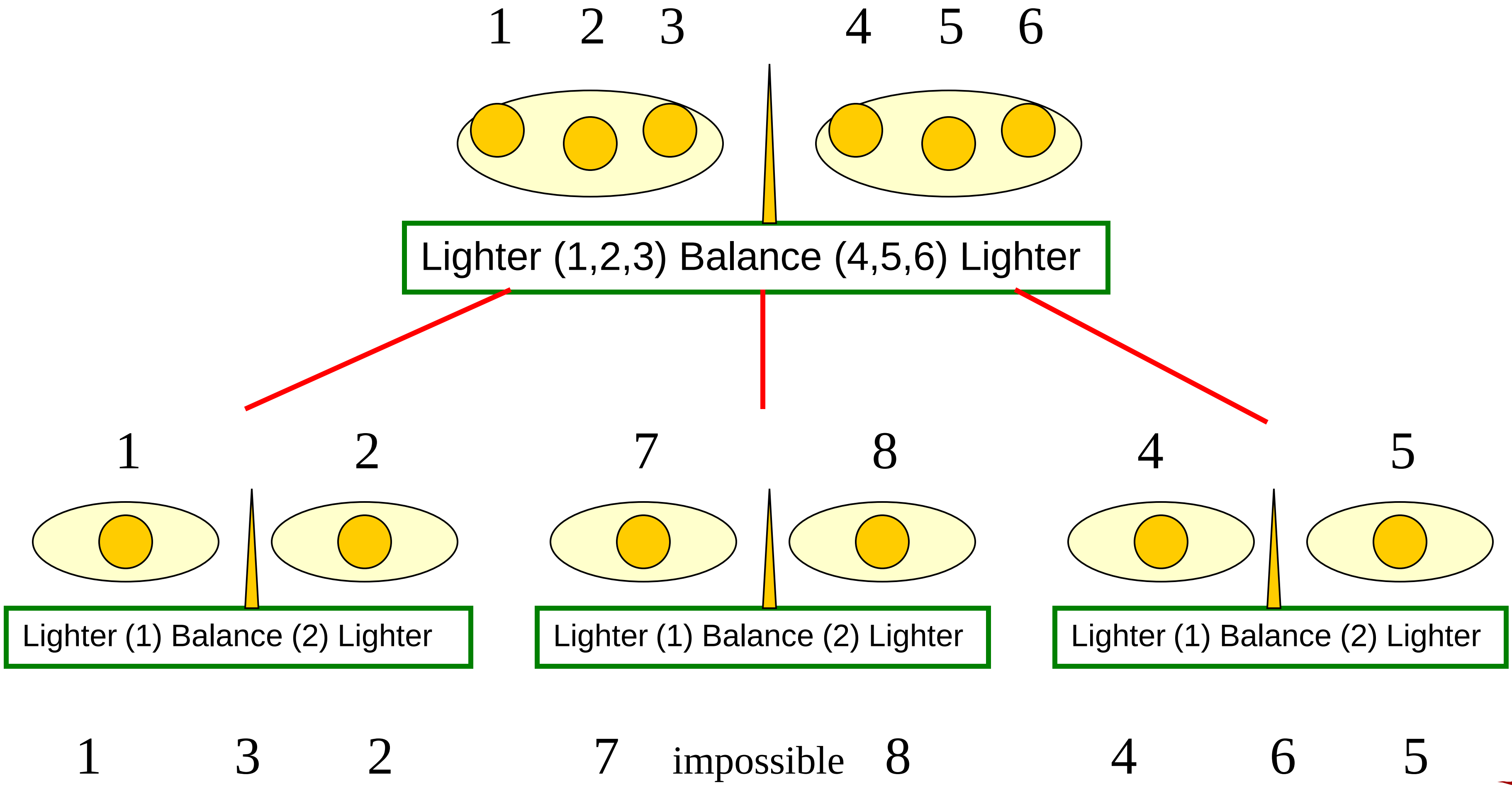
- The most comparisons needed to add a new item is the length of the longest path in *U* from the root to a leaf.
- If a binary search tree is balanced, locating or adding an item requires no more than $\lceil \log(n+1) \rceil$ comparisons.

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


1 2 3
4 5 6
Lighter (1,2,3) Balance (4,5,6) Lighter
1
2
7
8
4
5
Lighter (1) Balance (2) Lighter
Lighter (1) Balance (2) Lighter
Lighter (1) Balance (2) Lighter
1 3 2
7 impossible 8
4 6 5

Prefix Codes

- The problem of using bit strings to encode the letters of the English alphabet.

  How to improve coding efficiency?

- Using bit strings of different lengths to encode letters can improve coding efficiency.

  How to ensure the code having the definite meaning?

  For example, e: 0 a: 1 t: 01

  0101: eat, tea, eaea, tt?

- When letters are encoded using varying numbers of bits, some method must be used to determine where the bits for each character start and end.

the Concept of Prefix Codes

- To ensure that no bit string corresponds to more than one sequence of letters, the bit string for a letter must never occur as the first part of the bit string for another letter. Codes with this property are called prefix codes.

For example,

e: 0　　a: 10　　t: 110

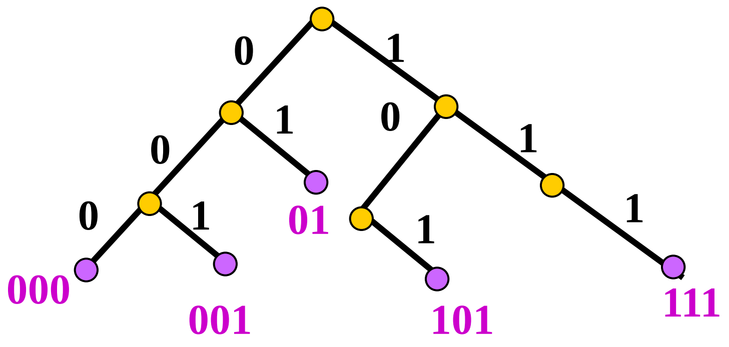
How to Construct Prefix Codes

- **Using a binary tree.**

  -- the left edge at each internal vertex is labeled by 0.

  -- the right edge at each internal vertex is labeled by 1.

  -- the leaves are labeled by characters which are encoded with the bit string constructed using the labels of the edges in the unique path from the root to the leaves.



Problem: **How to produce efficient codes based on the frequencies of occurrences of characters?**

For example,

Character	***a***	***b***	***c***	***d***	***e***	**……**
Frequences (w_i)	**82**	**14**	**28**	**38**	**131**	**……**

$$obj.\ \ \min\left(\sum_{i=1}^{26} l_i w_i\right)$$

where l_i is the length of prefix codes for characters i.

General problem: **Tree T has t leaves, w_1, w_2,…, w_t are weights, $l_i = l(w_i)$. Let the weight of tree T be**

$$w(T) = \sum_{i=1}^{t} l_i w_i$$

$$obj.\ \ \min(w(T))$$

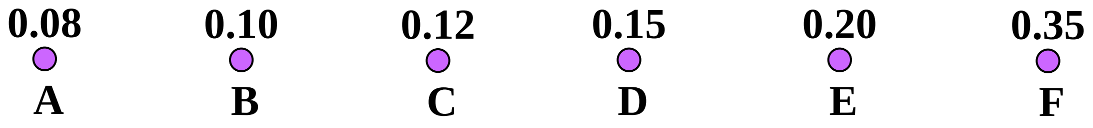
Huffman Coding

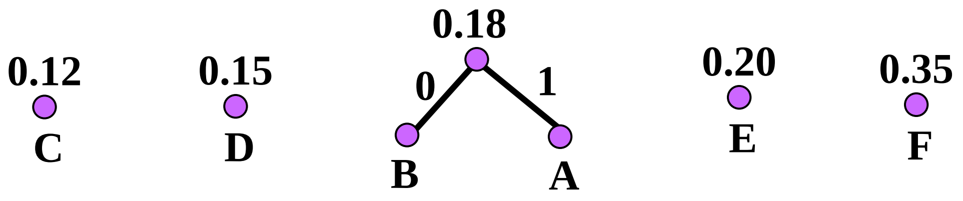
Algorithm 2 Huffman Coding.

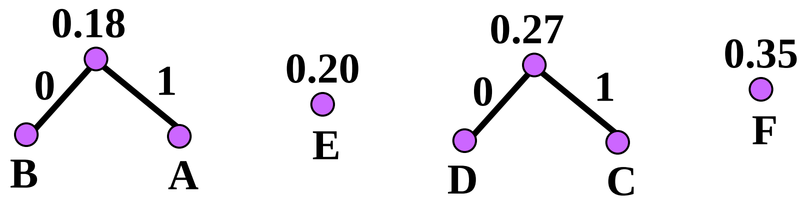
```
Procedure Huffman ($C$: symbols $a_i$ with frequencies $w_i$, $i=1, \ldots, n$)
$F$:=forest of $n$ rooted trees, each consisting of the single vertex $a_i$
    and assigned weight $w_i$
While $F$ is not a tree
 begin
    Replace the rooted trees $T$ and $T'$ of least weights from $F$ with
    $w(T) \geq w(T')$ with a tree having a new root that has $T$ as its left
    subtree and $T'$ as its right subtree. Label the new edge to $T$
    with 0 and the new edge to $T'$ with 1.
     Assign $w(T) + w(T')$ as the weight of the new tree.
end
 {The Huffman coding for the symbol $a_i$ is the concatenation of
    the labels of the edges in the unique path from the root to the
    vertex $a_i$}
```

〖Example 1〗 Use Huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?

Solution:

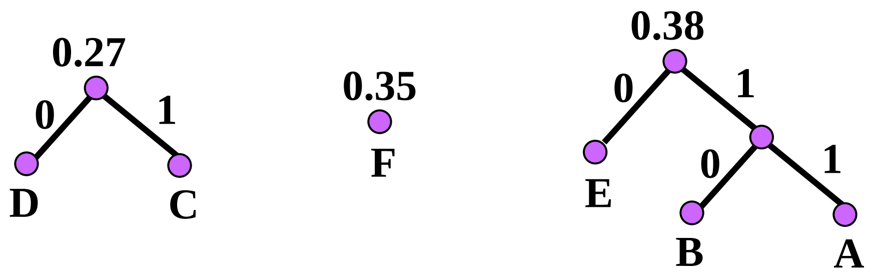




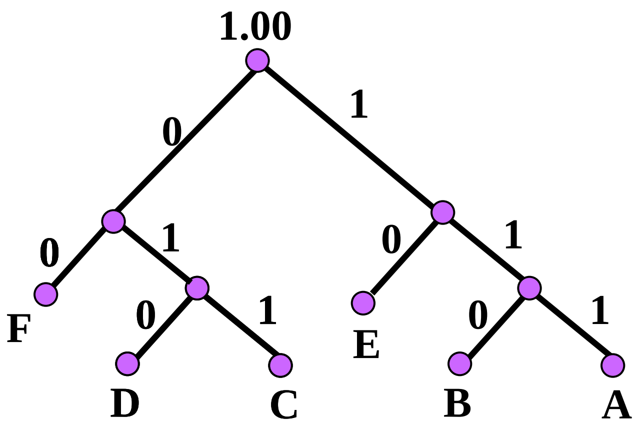


〖Example 1〗 Use Huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?

Solution:



……



Homework:

Seventh edition:

P. 769 1, 20, 23