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

Nikolaus
(1623-1708)

Jacob I
(1654-1705)

Nikolaus
(1662-1716)

Johann I
(1667-1748)

Nikolaus I
(1687-1759)

Nikolaus II
(1695-1726)

Daniel
(1700-1782)

Johann II
(1710-1790)

Johann III
(1746-1807)

Jackob
(1759-1789)

tree

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

Forest is an undirected graph with no simple circuits.

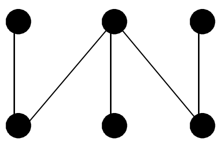
Note:

① Any tree must be a simple graph.

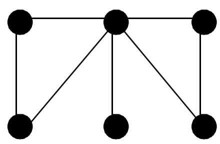
② Each connected components of forest is a tree.

〖Example 1〗 Which graphs are trees?

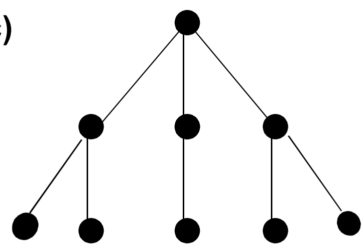
a)



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**

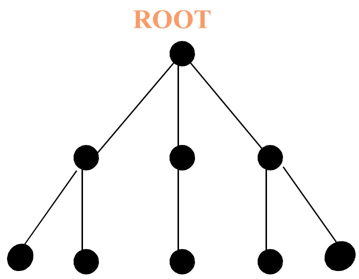
(2) $\Leftarrow$

- ✓ **connected**
- ✓ **no simple circuits**

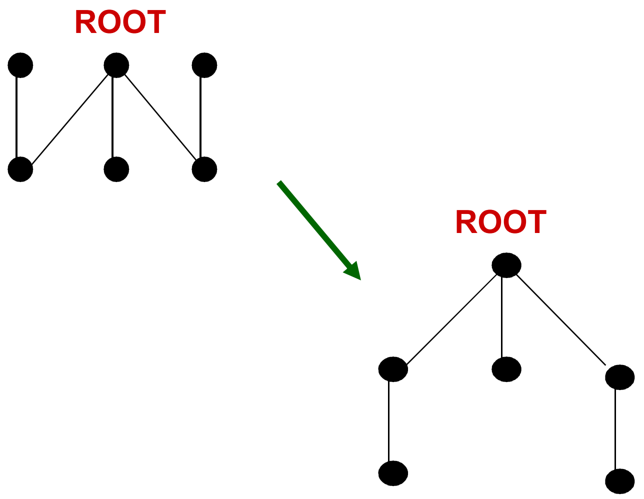
Rooted tree

- **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*.**

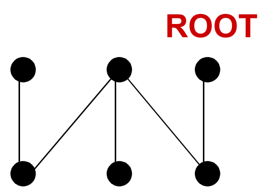
For example,



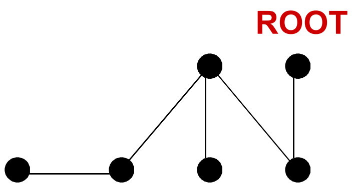
〖Example 2〗 Change an unrooted tree into a rooted tree.



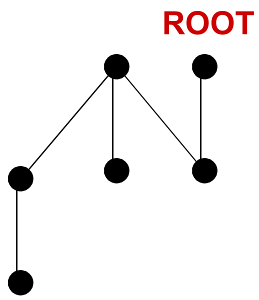
Problem: **What if a different root is chosen?**



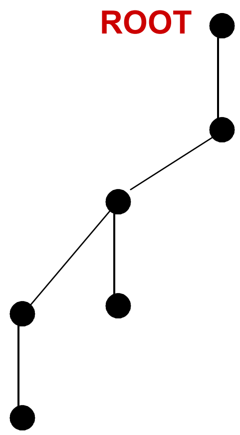
Problem: What if a different root is chosen?



Problem: What if a different root is chosen?



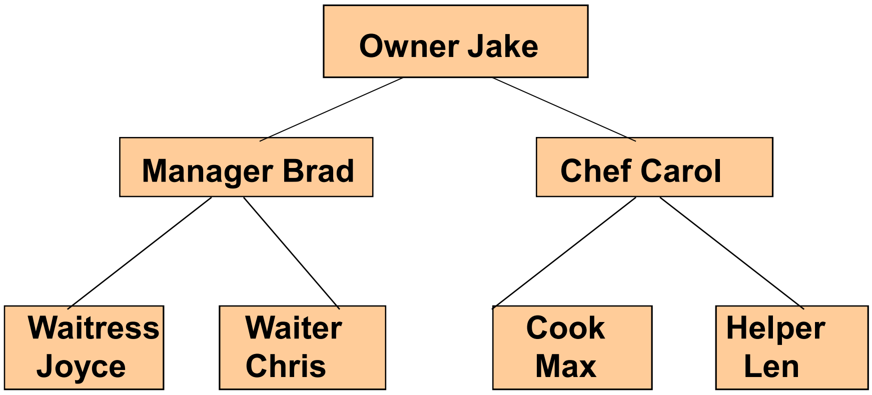
Problem: **What if a different root is chosen?**



Some concepts in tree

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

Jake's Pizza Shop Tree

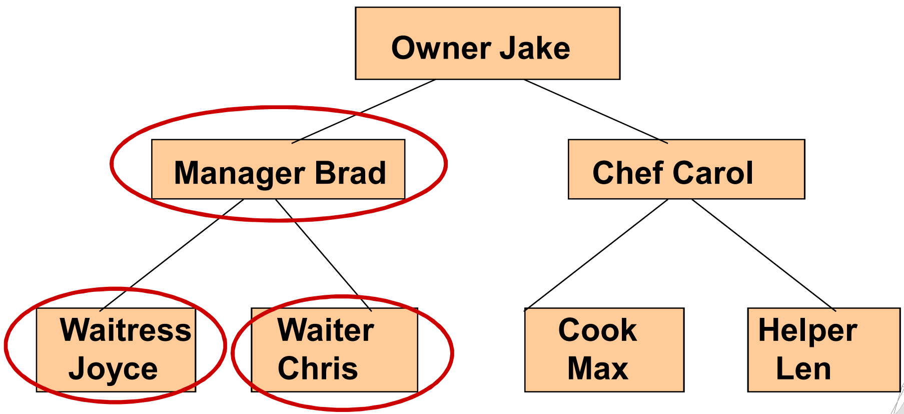


Parent

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

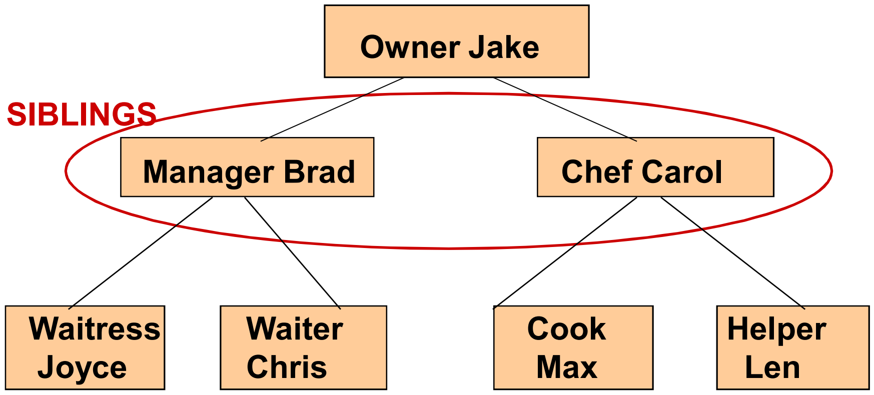
Child

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



Sibling

Vertices with the same parent are called *siblings* .

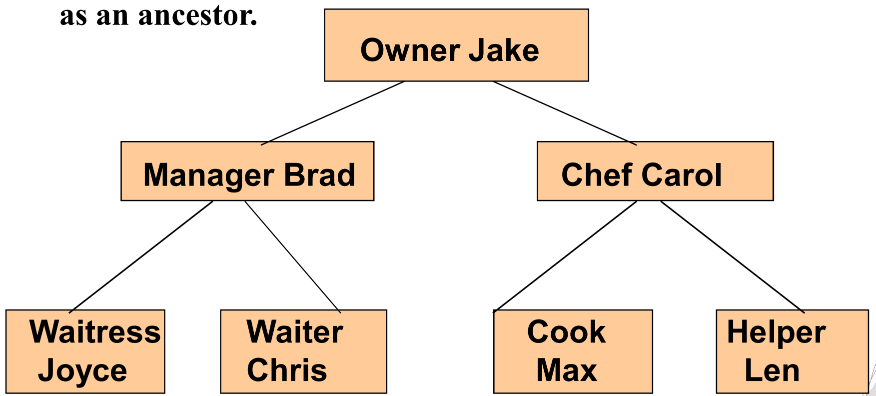


Ancestors

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

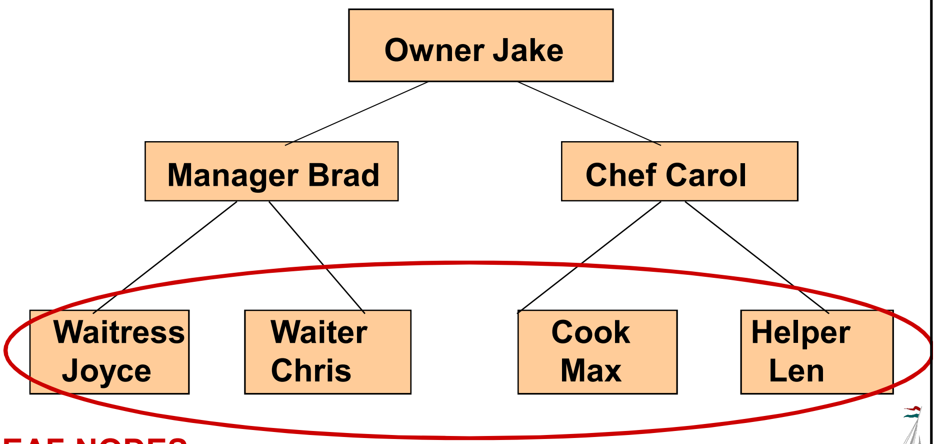
Descendants

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



Leaf

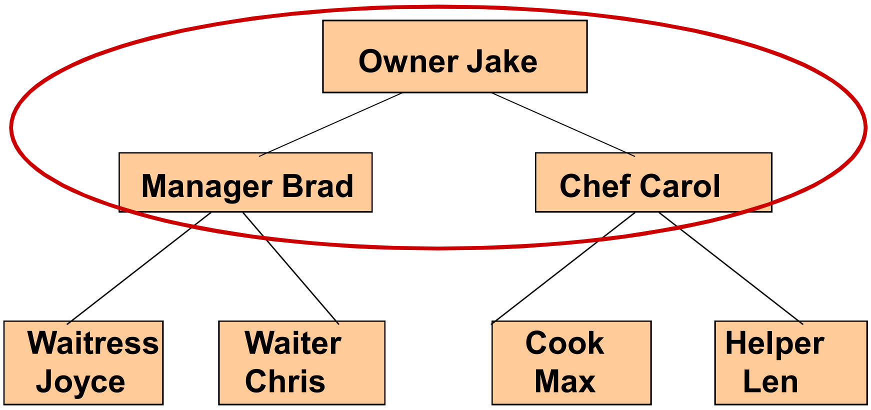
A vertex is called a *leaf* if it has no children.



LEAF NODES

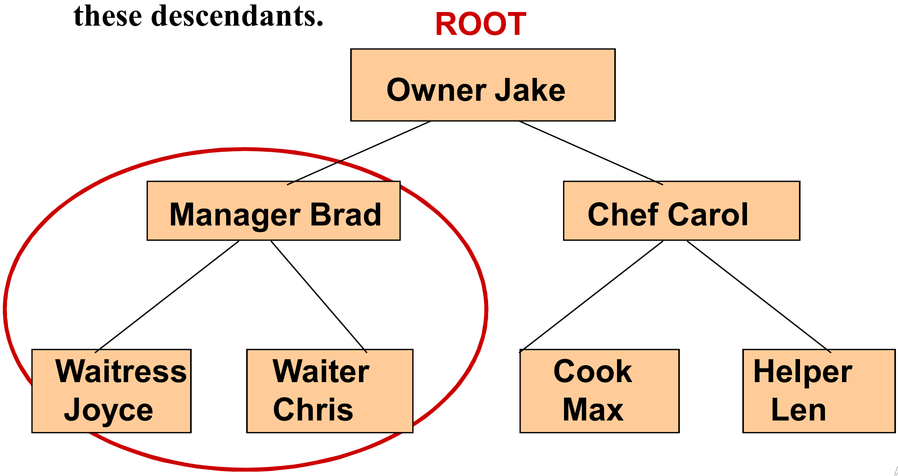
Internal Vertex

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.

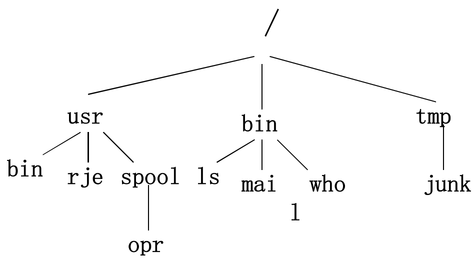


Trees as Models

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.

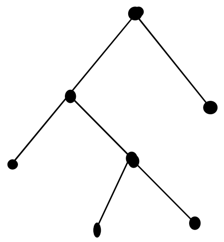


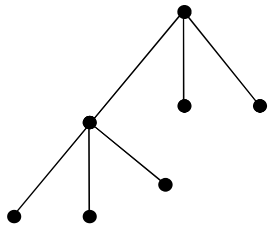
Binary Tree

【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.





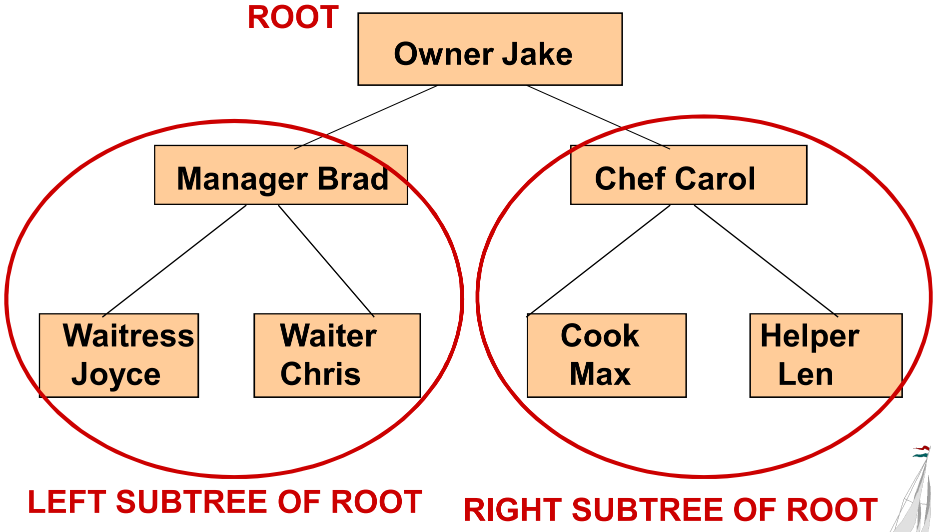
Ordered rooted tree

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

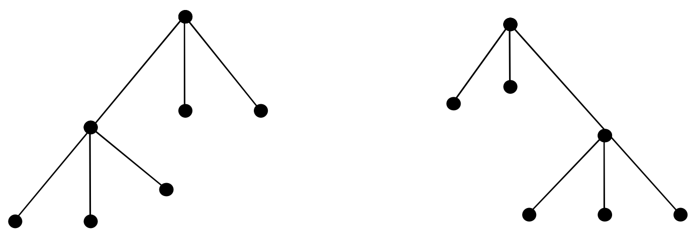
In an ordered binary tree, the two possible children of a vertex are called the *left child* and the *right child*, if they exist.

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

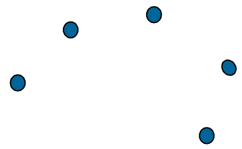


IS these two trees isomorphic?

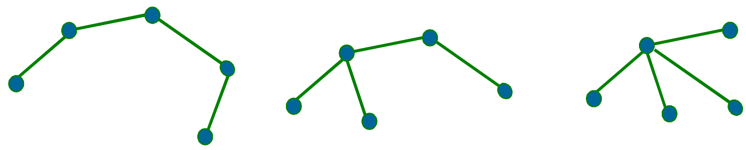


〖Example 3〗(1) How many nonisomorphic unrooted trees are there with n vertices if n=5 ?

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

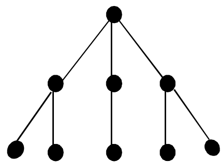
【 Theorem 2】 **A tree with *n* vertices has *n*-1 edges.**

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.

$v = 2+1+3+x$

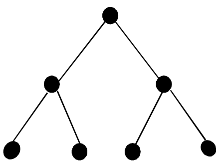
$e = \frac{1}{2} (2\times2 + 1\times3 + 3\times4 + x \times1) = v -1$

$x = 9$

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.

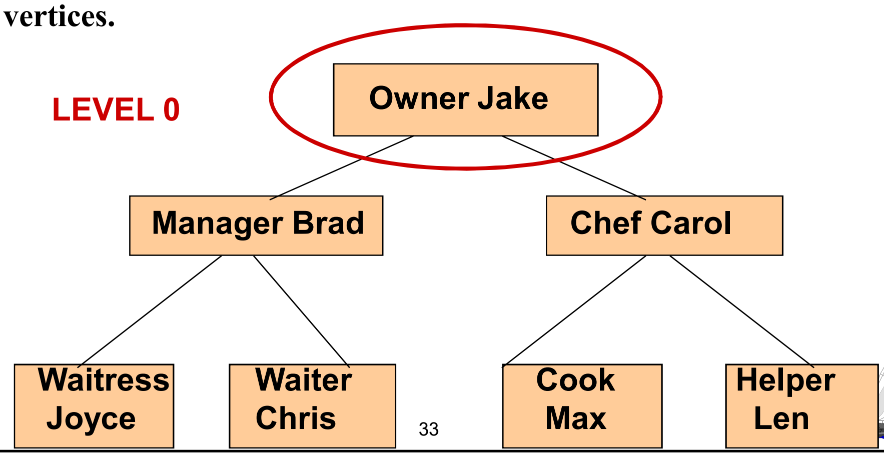
$$i = 10000$$
$$n = 5i + 1$$
$$n = i + l$$
$$l = 40001$$
$$n\text{-}1 = 50000$$

Level

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

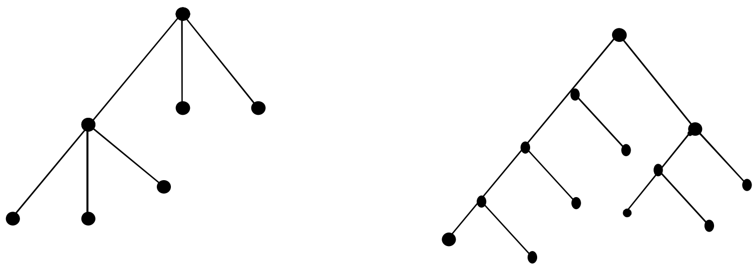
Height

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，我认为错的）



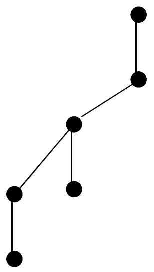
Balanced

A tree is balanced if:

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
```



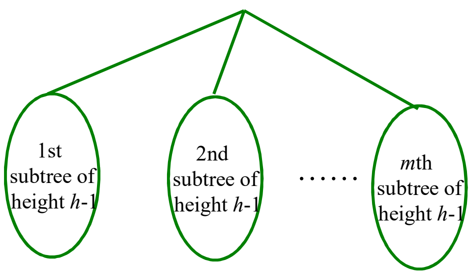
Tree Properties

【 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**

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

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,

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

Question

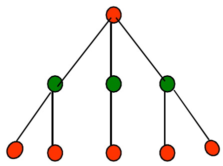
Every tree is a bipartite?

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

11.1 Introduction to Trees

11.2 Applications of Trees

11.3 Tree Traversal

11.4 Spanning Trees

11.5 Minimum Spanning 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.

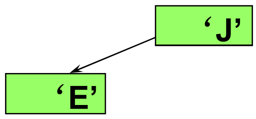
-- The first value to be inserted is put into the root.

'J'

-- Inserting ‘E’ into the BST

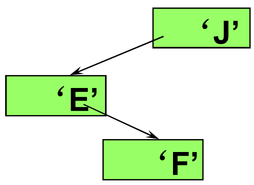
Thereafter, each value to be inserted begins by comparing itself to the value in the root, moving left it is less, or moving right if it is greater.

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



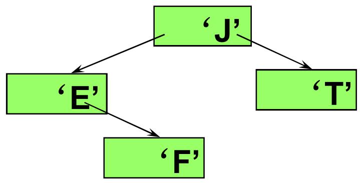
-- Inserting ‘F’ into the BST

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.



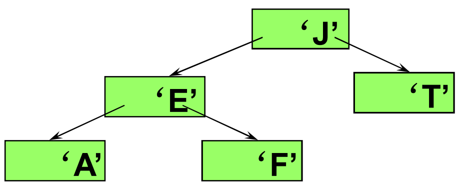
-- Inserting 'T' into the BST

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.



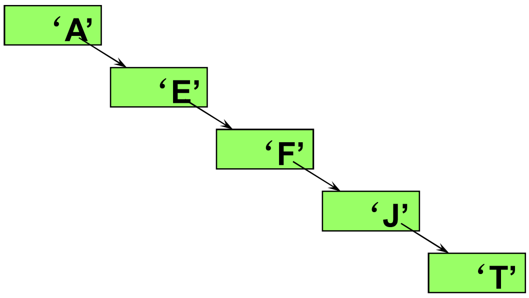
--Inserting 'A' into the BST

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.

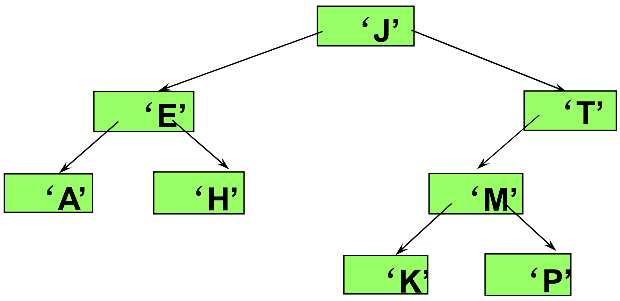


What binary search tree . . .

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



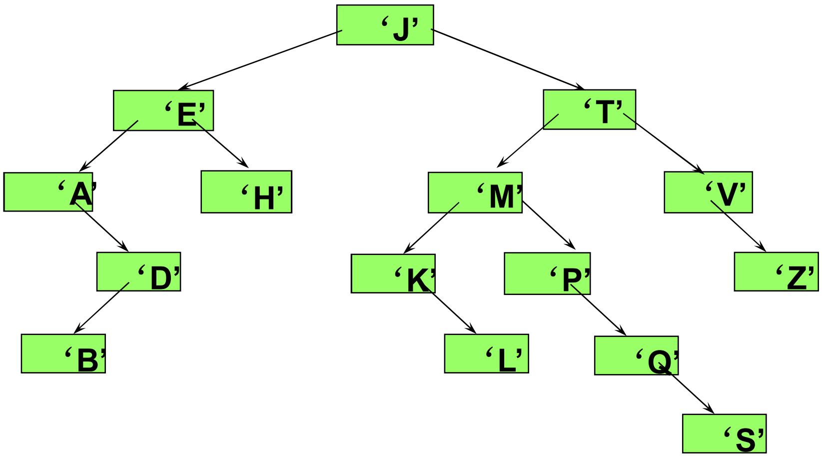
Another binary search tree



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

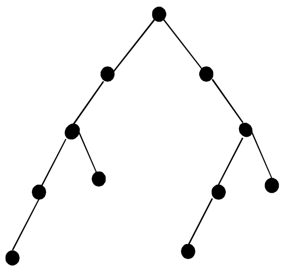
Algorithm 1 Locating and adding items to a binary search tree

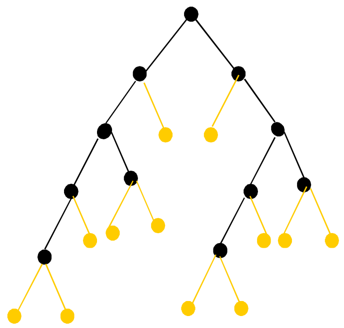
Procedure insertion (T: binary search tree, x: item)
v:=root of T
While $v \neq$null ***and*** label(v) $\neq x$
Begin
  if x<label(v) ***then***
    if left child of $v \neq$null ***then*** v:=left child of v
    else add new vertex as a left child of v and set v:=null
  else
    if right child of $v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

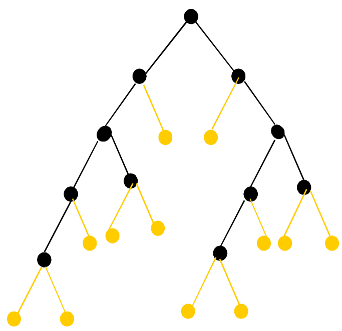
☸ The computational complexity

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.





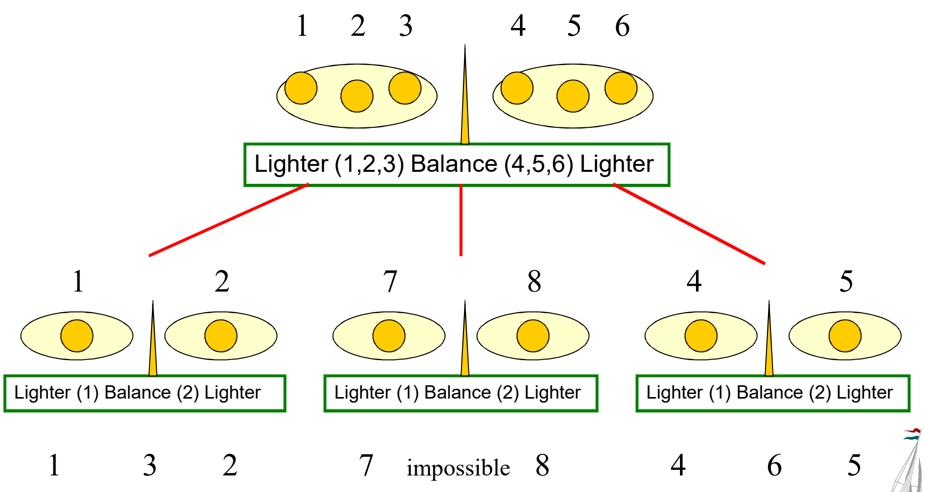


- **The most comparisons needed to add a new item is the length of the longest path in *U* from the root to a leaf.**
- **If a binary search tree is balanced, locating or adding an item requires no more than $\lceil \log(n+1) \rceil$ comparisons.**

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



3. 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

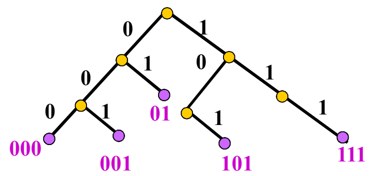
How to Construct Prefix Codes

- **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.



Huffman Coding

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)	**0.30**	**0.14**	**0.28**	**0.38**	**0.13**	**……**

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

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

General problem: Tree T has t leaves, $w_1, w_2, \ldots, w_t$ are weights, $l_i = l(w_i)$. Let the weight of tree T be $w(T) = \sum_{i=1}^{t} l_i w_i$

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

❁ Huffman Coding

Algorithm 2 Huffman Coding.

Procedure *Huffman* (C: symbols a_i with frequencies w_i, i=1, …, 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.

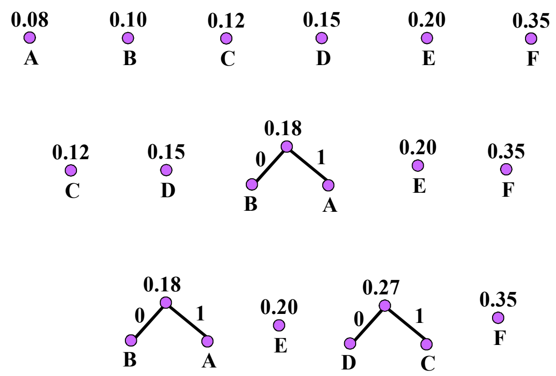
Assign $w(T) + w(T')$ as the weight of the new tree.

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}

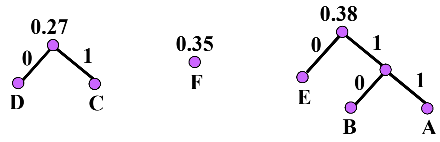
〖Example 1〗 Use Huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?

Solution:

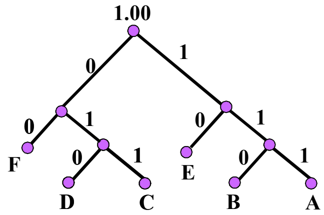


〖Example 1〗 Use Huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?

Solution:



……



Homework:

Sec.11.2 1, 20, 23

CHAPTER 11 Trees

1. Traversal Algorithms

- ***A traversal algorithm*** **is a procedure for systematically visiting every vertex of an ordered rooted tree.**
- **Tree traversals are defined recursively.**
- **Three traversals are named:**
  - ✓ **preorder,**
  - ✓ **inorder,**
  - ✓ **postorder.**

☸ PREORDER Traversal Algorithm

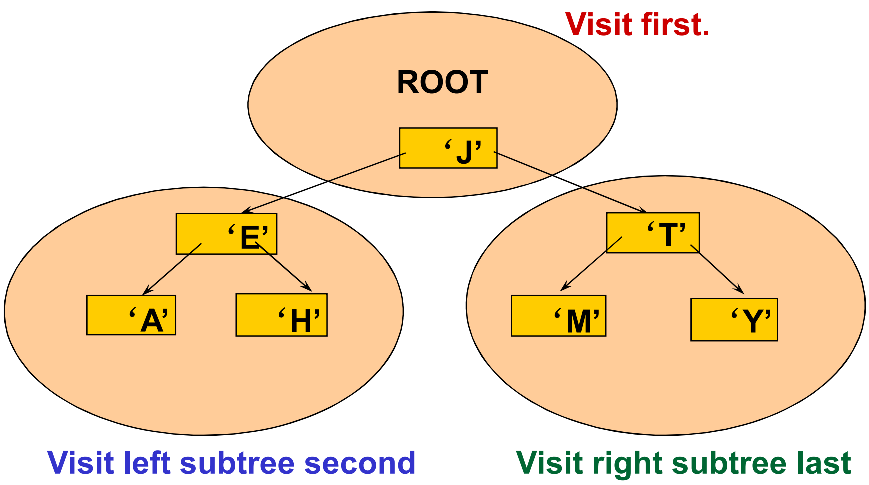
【Definition】 Let T be an ordered tree with root r. If T has only r, then r is the *preorder traversal* of T. Otherwise, suppose $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right in T. The *preorder traversal* begins by visiting r. Then traverses T_1 in preorder, then traverses T_2 in preorder, and so on, until T_n is traversed in preorder.

Note:

Preorder traversal of an binary ordered tree

- Visit the root.
- Visit the left subtree, using preorder.
- Visit the right subtree, using preorder.

Preorder Traversal: J E A H T M Y



❁ INORDER Traversal Algorithm

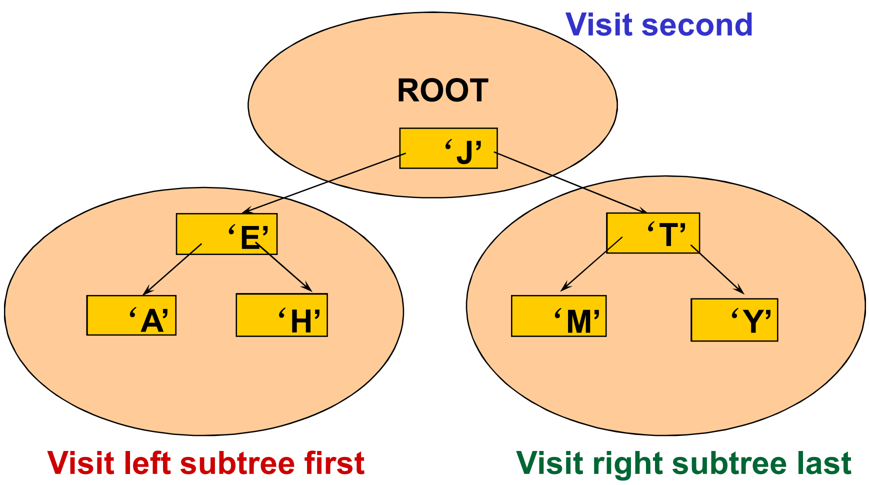
【**Definition**】 **Let T be an ordered tree with root r. If T has only r, then r is the *inorder traversal* of T. Otherwise, suppose $T_1, T_2, \ldots, T_n$ are the left to right subtrees at r. The *inorder traversal* begins by traversing T_1 in inorder. Then visits r, then traverses T_2 in inorder, and so on, until T_n is traversed in inorder.**

Note:

Inorder traversal of an binary ordered tree

- **Visit the left subtree, using inorder.**
- **Visit the root.**
- **Visit the right subtree, using inorder.**

Inorder Traversal: **A E H J M T Y**



☸ POSTORDER Traversal Algorithm

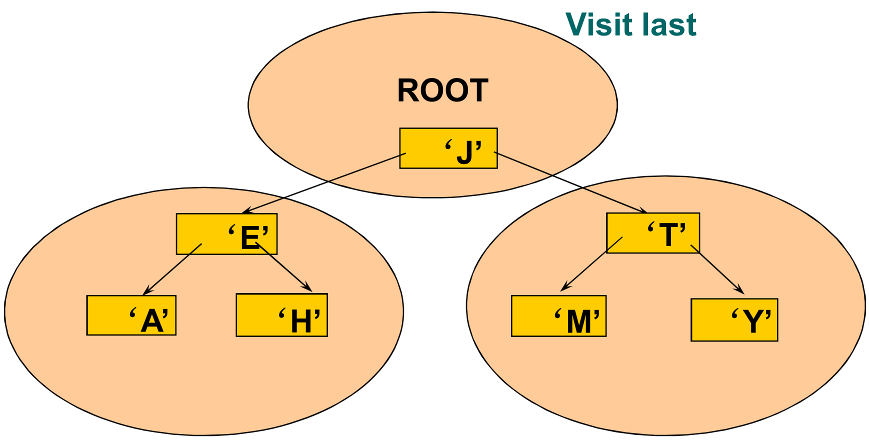
【Definition】 Let T be an ordered tree with root r. If T has only r, then r is the postorder traversal of T. Otherwise, suppose T_1, T_2,…, T_n are the left to right subtrees at r. The postorder traversal begins by traversing T_1 in postorder. Then traverses T_2 in postorder, until T_n is traversed in postorder, finally ends by visiting r.

Note:

Postorder traversal of an binary ordered tree

- Visit the left subtree, using postorder.
- Visit the right subtree, using postorder.
- Visit the root.

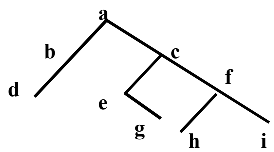
Postorder Traversal: A H E M Y T J



Visit left subtree first

Visit right subtree second

〖Example 1〗 In which order does a preorder, inorder or postorder traversal visit the vertices in the ordered rooted tree shown in the following figure?



- **Preorder traversal : a, b, d, c, e, g, f, h, i**
- **Inorder traversal : d, b, a, e, g, c, h, f, i**
- **Postorder traversal : d, b, g, e, h, i, f, c, a**

2. Infix, prefix, and postfix notation

Complicated expressions can be represented using ordered rooted trees, such as

- ✓ **Compound propositions**
- ✓ **Combinations of sets**
- ✓ **Arithmetic expressions**

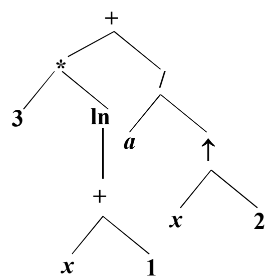
A Binary Expression Tree is . . .

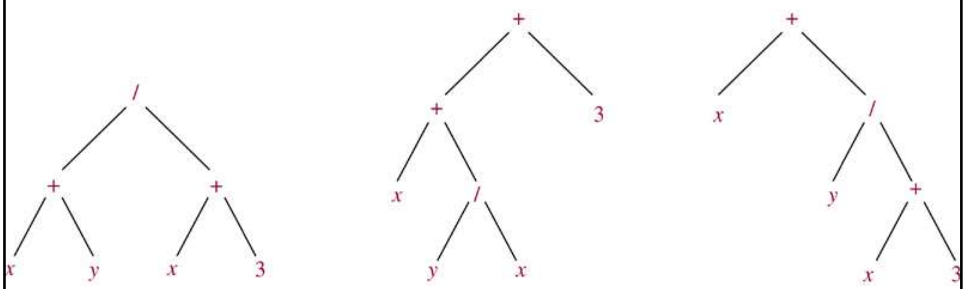
A special kind of binary tree in which:

1. **Each leaf node contains a single operand,**
2. **Each nonleaf node contains a single operator, and**
3. **The left and right subtrees of an operator node represent subexpressions that must be evaluated before applying the operator at the root of the subtree.**

〖Example 2〗 What is the ordered tree that represents the expression $3*\ln(x+1)+a/x^2$?

Solution:

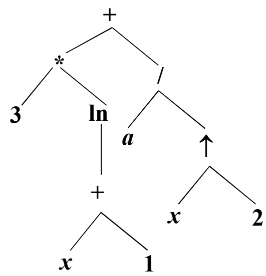




☸ Infix Form

The fully parenthesized expression obtained by an inorder traversal of the binary tree is said to be in *infix form*.

For example,

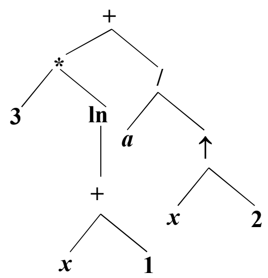


Infix form: $(3 * \ln(x+1)) + (a/(x \uparrow 2))$

Prefix Form

The expression obtained by an preorder traversal of the binary tree is said to be in ***prefix form (Polish notation).***

For example,

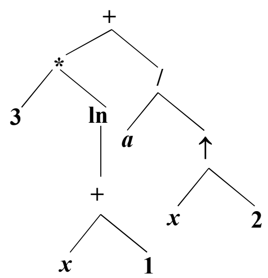


Prefix form: $+*3\ln+x1/a\uparrow x2$

Postfix Form

The expression obtained by an postorder traversal of the binary tree is said to be in ***postfix form (reverse Polish notation).***

For example,

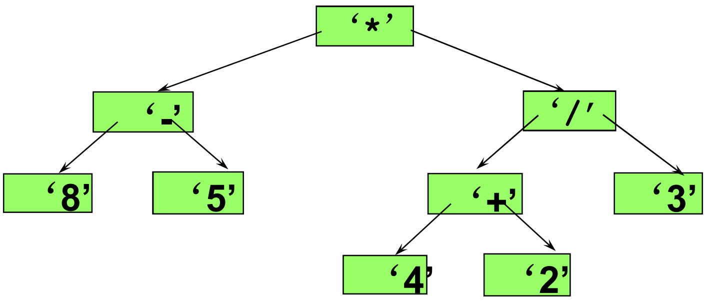


Postfix form: $3x1+\ln*ax2\uparrow/+$

❁ Evaluate the binary expression tree

- When a binary expression tree is used to represent an expression, the levels of the nodes in the tree indicate their relative precedence of evaluation.
- Operations at higher levels of the tree are evaluated later than those below them. The operation at the root is always the last operation performed.

Evaluate this binary expression tree

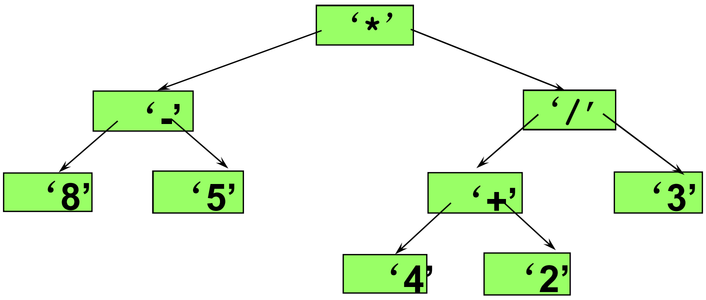


Prefix form: * - 8 5 / + 4 2 3

Note:

We can evaluate this expression from right to left.

Evaluate this binary expression tree



Postfix form: **8 5 - 4 2 + 3 / ***

Note:

We can evaluate this expression from left to right.

Homework:

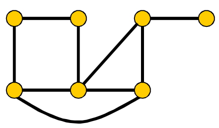
Sec. 11.3 8, 16

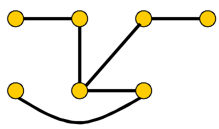
CHAPTER 11 Trees

❁ The definition of spanning tree

【Definition 1】 Let *G* be a simple graph. A spanning tree of *G* is a subgraph of *G* that is a tree containing every vertex of *G*.

For example,

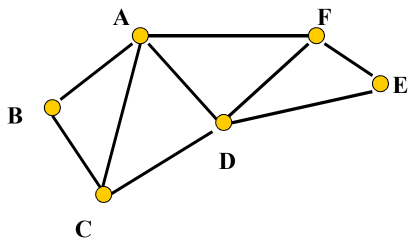


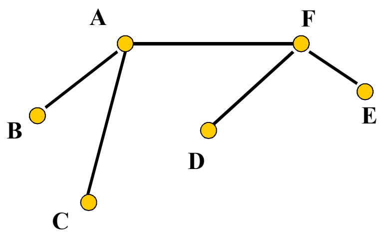


Problem:

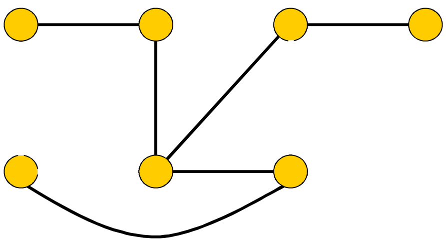
Why should we study the problem of spanning tree?

-- Consider the system of roads





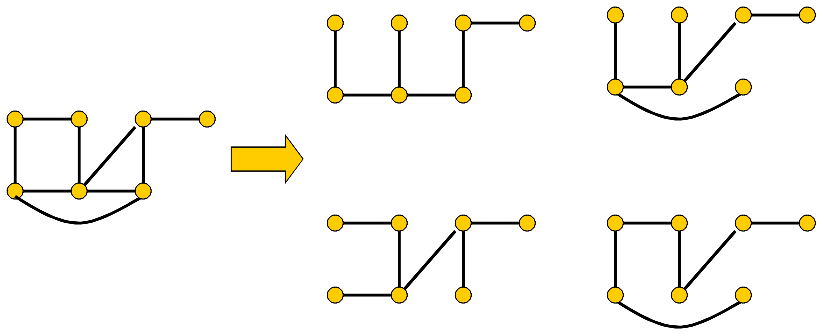
☸ Find A Spanning Tree of The Simple Graph



Method:

Find spanning trees by removing edges from simple circuits.

More than one spanning tree for a simple graph



【 Theorem 1 】 A simple graph is connected if and only if it has a spanning tree.

Proof:

First, suppose that a simple graph G has a spanning tree tree T.

T contains every vertex of G.

There is a path in T between any two of its vertices.

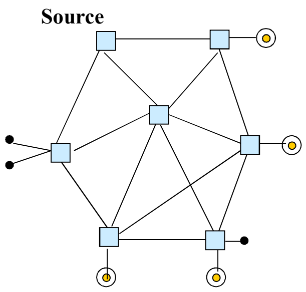
Since T is a subgraph of G, there is a path in G between any two of its vertices. Hence G is connected.

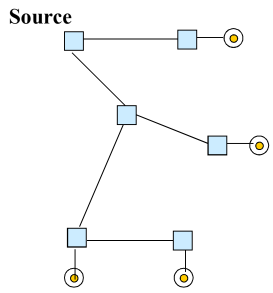
Second, suppose that G is connected.

We can find a spanning trees by removing edges from simple circuits of G.

❁ The Applications of Spanning Trees

〖Example 1〗 **IP Multicasting**.





☸ Algorithms for constructing spanning trees

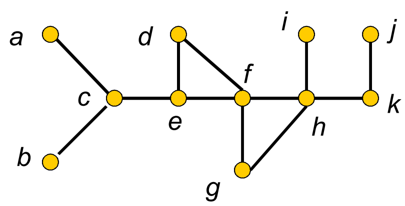
- **Theorem 1 gives an algorithm for finding spanning trees by removing edges from simple circuits.**
- **Instead of constructing spanning trees by removing edges, spanning trees can be built up by successively adding edges.**
- **Two algorithm:**
  - ✓ **Depth-first search**
  - ✓ **Breadth-first search**

❁ Depth-first search

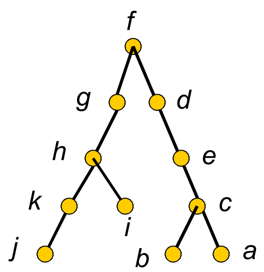
Depth-first search (also called *backtracking*) -- this procedure forms a rooted tree, and the underlying undirected graph is a spanning tree.

1. Arbitrarily choose a vertex of the graph as root.
2. Form a path starting at this vertex by successively adding edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path.
3. Continue adding edges to this path as long as possible.
4. If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree.
5. If the path does not go through all vertices, more edges must be added. Move back to the next to last vertex in the path, if possible, form a new path starting at this vertex passing through vertices that were not already visited. If this cannot be done, move back another vertex in the path. Repeat this process.

〖Example 2〗 Use a depth-first search to find a spanning tree for the following graph.



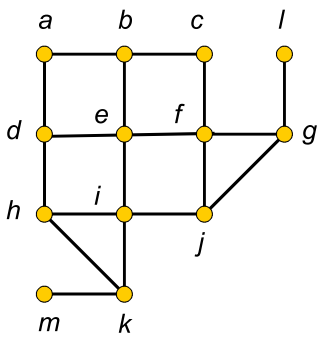
Solution:

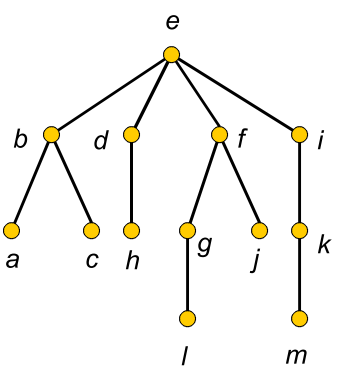


☸ Breadth-first search

1. **Arbitrarily choose a vertex of the graph as a root, and add all edges incident to this vertex.**
2. **The new vertices added at this stage become the vertices at level 1 in the spanning tree. Arbitrarily order them.**
3. **For each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit. Arbitrarily order the children of each vertex at level 1. This produces the vertices at level 2 in the tree.**
4. **Follow the same procedure until all the vertices in the tree have been added.**

〖Example 3〗 Use a breadth-first search to find a spanning tree for the following graph.

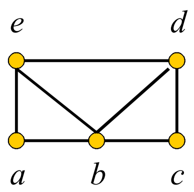


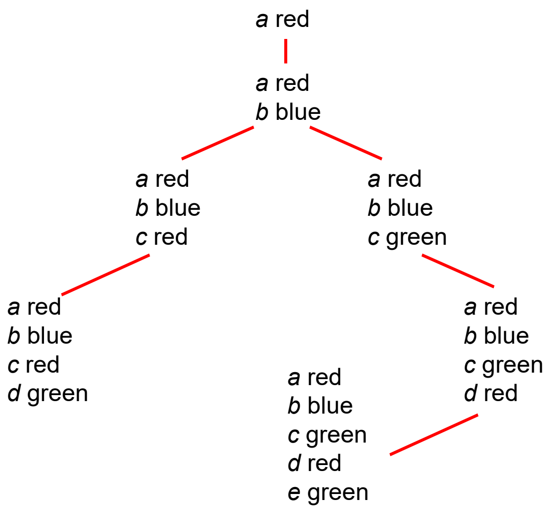


❁ Backtracking scheme

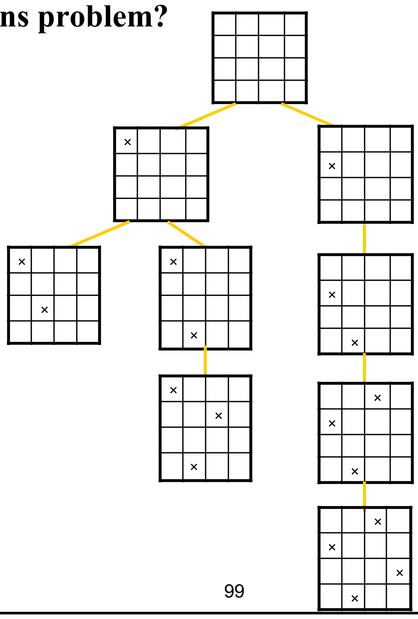
- There are problems that can be solved only by performing an exhaustive search of all possible solutions.
- One way to search systematically for a solution is to use a decision tree, where each internal vertex represents a decision and each leaf a possible solution.
- The method to find a solution via backtracking
- The applications of backtracking scheme
  - ✓ Graph Coloring
  - ✓ The n-Queens Problem
  - ✓ Sums of Subsets

〖Example 4〗 How can backtracking be used to decide whether the following graph can be colored using 3 colors?



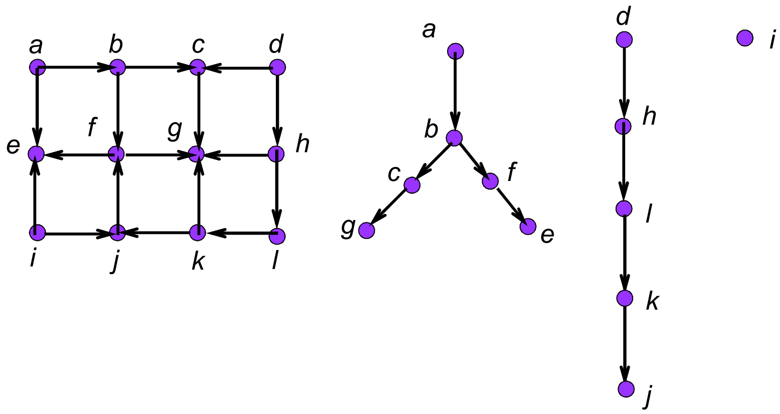


〖Example 5〗 **How can backtracking be used to solve the *4*-queens problem?**



❁ Depth-first Search in Directed Graphs

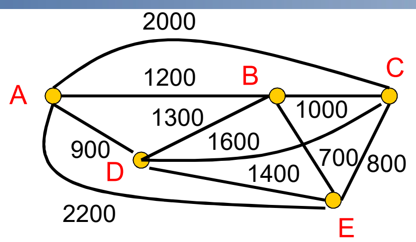
〖Example 6〗 **What is the output of depth-first search given the following graph as input?**



Homework:

Sec. 11.4 4, 14, 16(14), 29

CHAPTER 11 Trees



A weighted graph showing monthly lease costs for lines in a computer network.

Problem:

Which links should be made to ensure that there is a path between any two computer centers so that the total cost of the network is minimized?

We can solve this problem by finding a spanning tree so that the sum of the weights of the edges of the tree is minimized. Such a spanning tree is called a **minimum spanning tree**.

❁ the Concept of Minimum Spanning Trees

【Definition 1】 A *minimum spanning tree* in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

Algorithms for minimum spanning trees

Two algorithms for constructing minimum spanning trees.

- ✓ **Prim's algorithm**
- ✓ **Kruskal's algorithm**

Both proceed by successively adding edges of smallest weight from those edges with a specified property that have not already been used.

These two algorithms are examples of *greedy algorithms*.

Prim's algorithm

Procedure *Prim* (*G*: weighted connected undirected graph with *n* vertices)

***T*:= a minimum-weight edge**

for *i*:= 1 to *n*-2

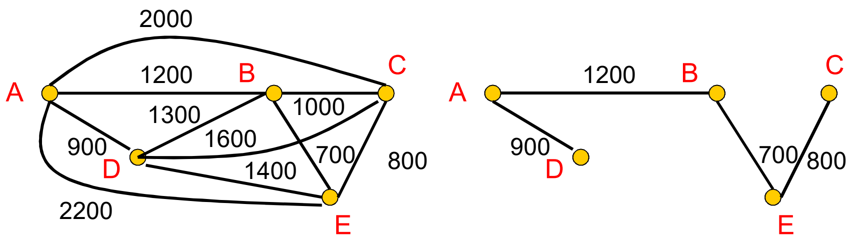
begin

  ***e*:= an edge of minimum weight incident to a vertex in *T* and not forming a simple circuit in *T* if added to *T*.**

  ***T*:= *T* with *e* added**

end {*T* is a minimum spanning tree of *G*}

〖Example 1〗 Find a minimum spanning tree in the weighted graph.



Choice	Edge	Cost
1	BE	700
2	EC	800
3	BA	1200
4	AD	900
	Total:	3600

☸ Kruskal's algorithm

procedure Kruskal (*G*: weighted connected undirected graph with n vertices)

T:= empty graph

for i:= 1 to *n*-1

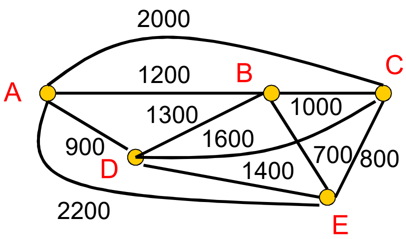
begin

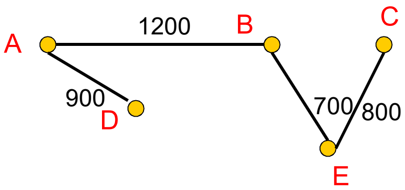
e:= any edge in *G* with smallest weight that does not form a simple circuit when added to *T*

T:= *T* with e added

end {*T* is a minimum spanning tree of *G*}

〖Example 2〗 **Find a minimum spanning tree in the weighted graph.**





Choice	Edge	Cost
1	BE	700
2	EC	800
3	AD	900
4	AB	1200
	Total:	3600

Homework:

Sec. 11.5 **3,7,12**