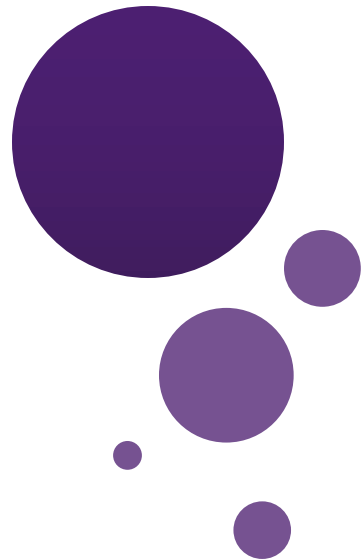




UNIVERSITY
AT ALBANY

State University of New York

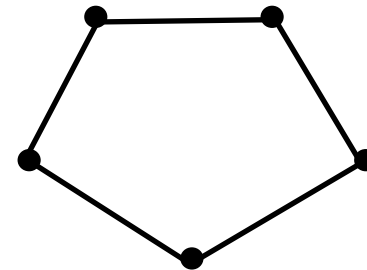
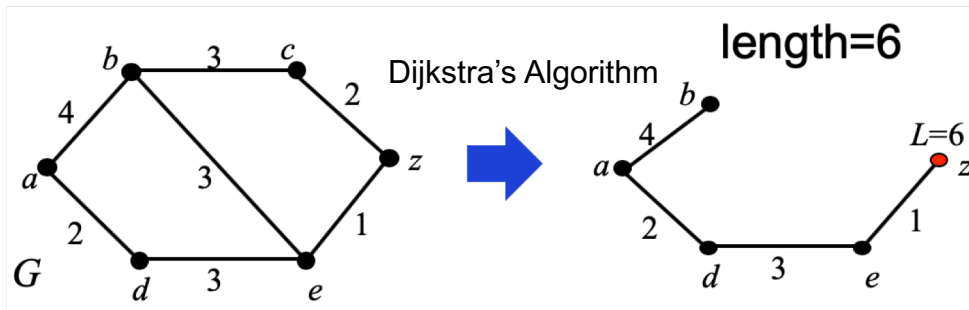
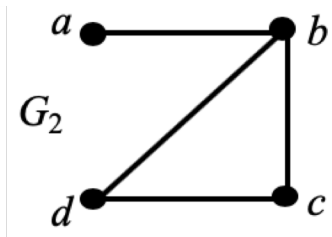
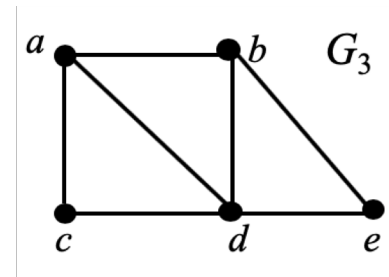
Lecture 37: Trees



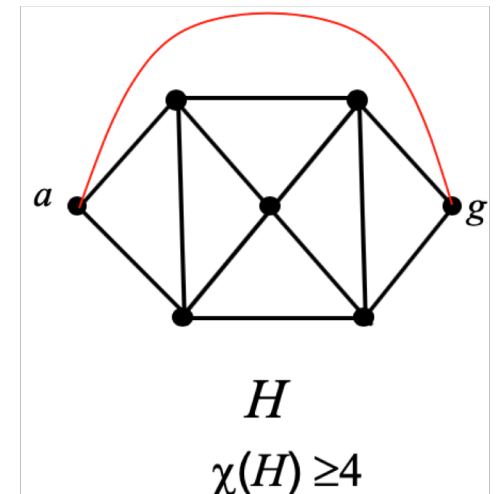
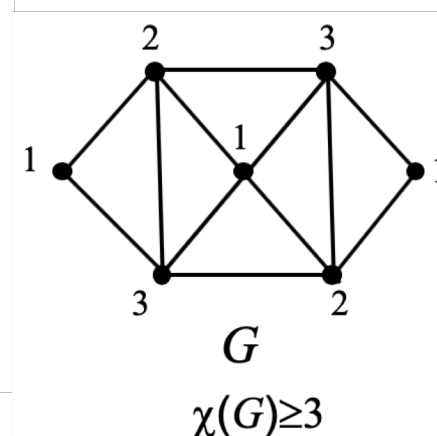
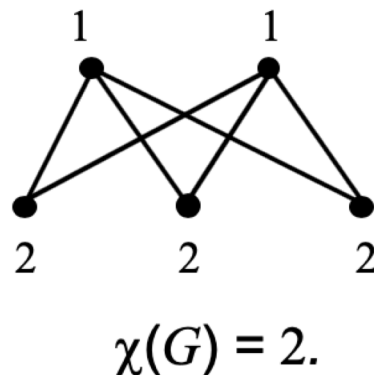
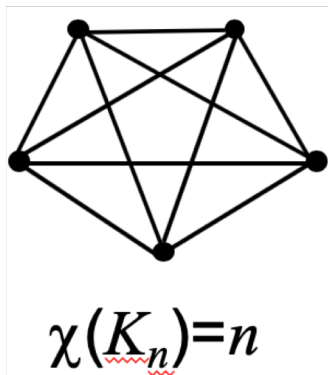
Dr. Chengjiang Long
Computer Vision Researcher at Kitware Inc.
Adjunct Professor at SUNY at Albany.
Email: clong2@albany.edu

Recap Previous Lecture

- Euler and Hamilton Path
- Shortest Path Problem
- Planar Graph and Graph Coloring



$\chi(C_n) = 2$ if n is even,
 $\chi(C_n) = 3$ if n is odd.



Outline

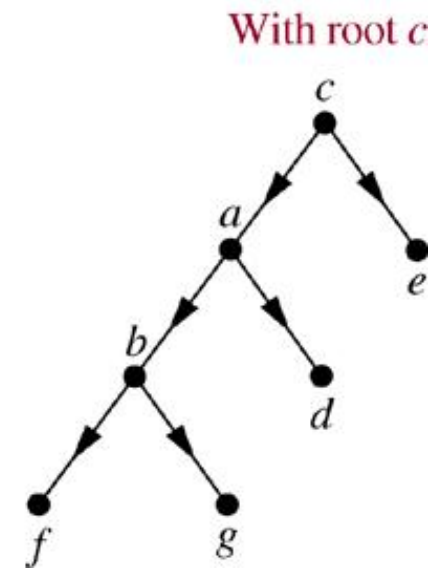
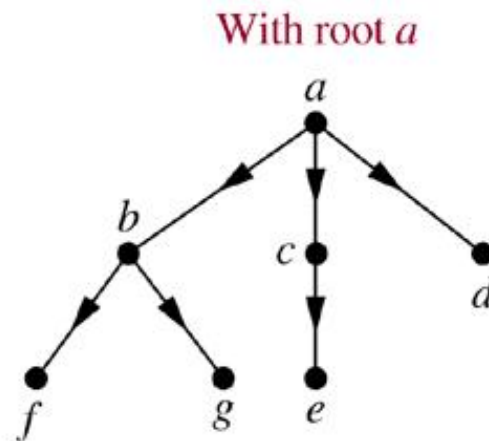
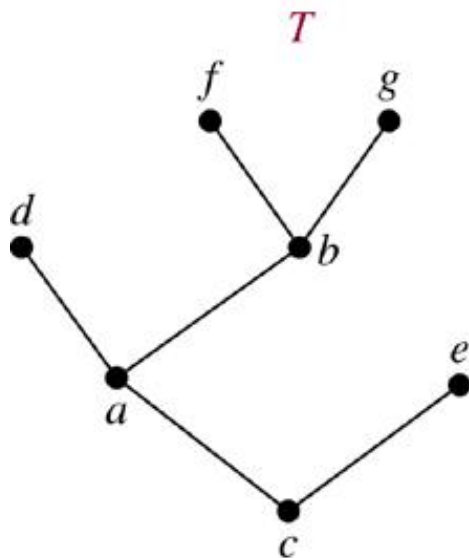
- Introduction to Trees
- Tree Traversal
- Spanning Trees

Outline

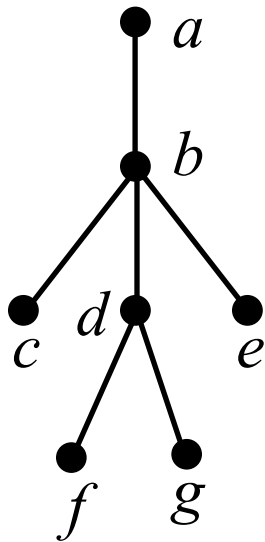
- **Introduction to Trees**
- Tree Traversal
- Spanning Trees

Introduction to Trees

- A **tree** is a connected undirected graph with no simple circuits.
- A **rooted tree** is a tree in which one vertex has been designed as the root and every edge is directed away from the root.



Introduction to Trees



a is the **parent** of b , b is the **child** of a ,

c, d, e are **siblings**,

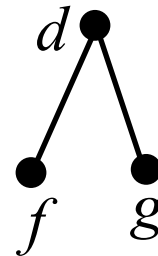
a, b, d are **ancestors** of f

c, d, e, f, g are **descendants** of b

c, e, f, g are **leaves** of the tree (deg=1)

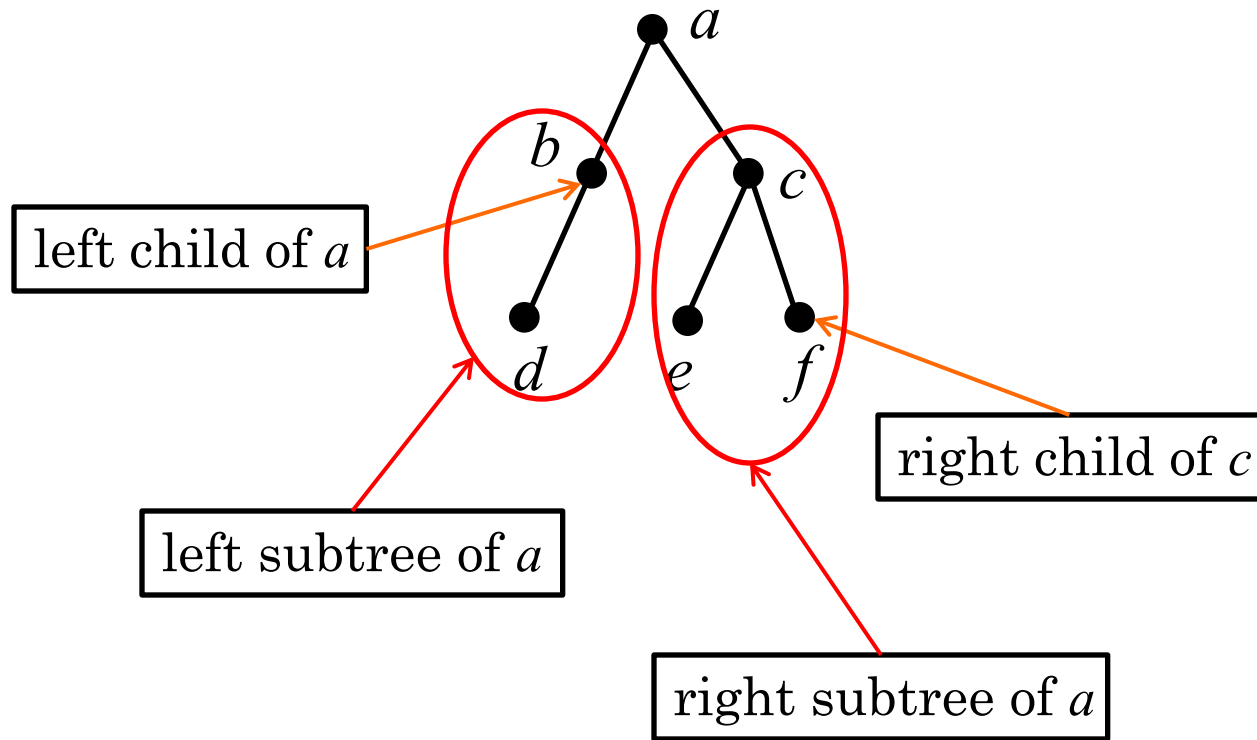
a, b, d are **internal vertices** of the tree
(at least one child)

subtree with d as its root:



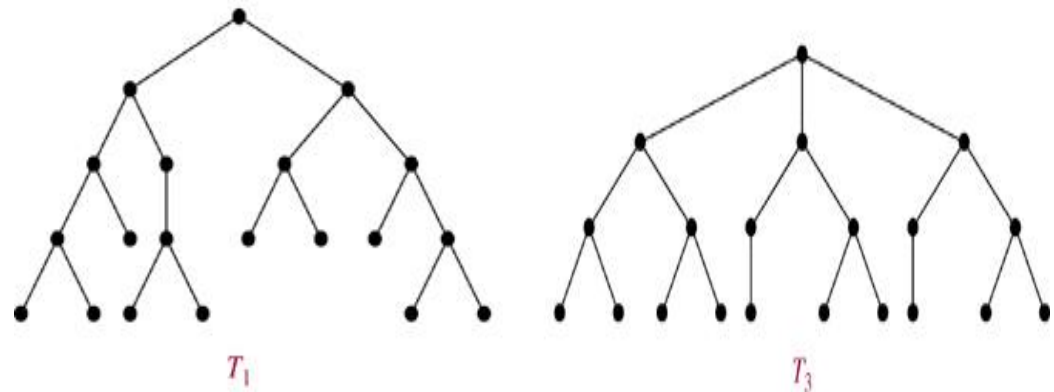
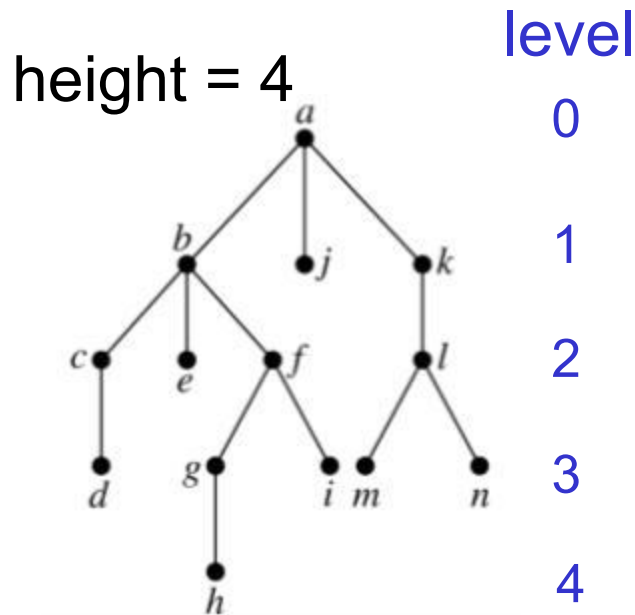
Vertices that have children are called **internal vertices**.

Binary Tree



Levels of Trees

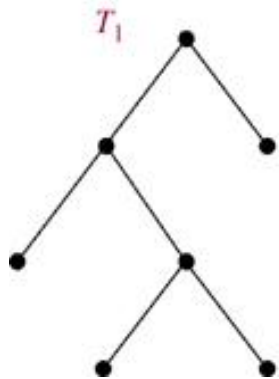
- The **level** of a vertex v in a rooted tree is the length of the unique path from the root to this vertex. The level of the root is defined to be zero. The **height** of a rooted tree is the maximum of the levels of vertices.



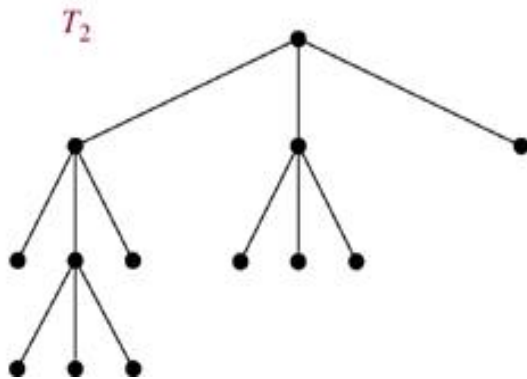
Remark: A rooted m -ary tree of height h is **balanced** if all leaves are at levels h or $h-1$.

m-ary Tree

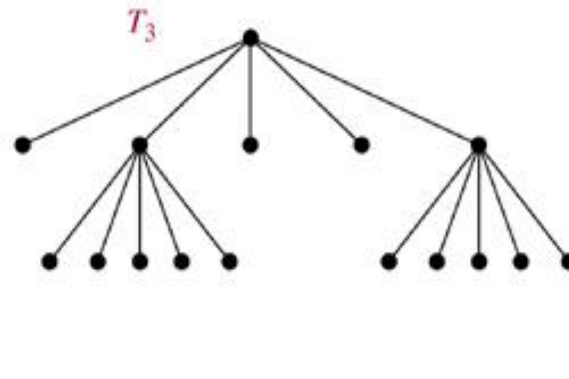
- A rooted tree is called an *m-ary tree* if every internal vertex has no more than m children.
- The tree is called a *full m-ary tree* if every internal vertex has exactly m children. An m -ary tree with $m=2$ is called a *binary tree*.



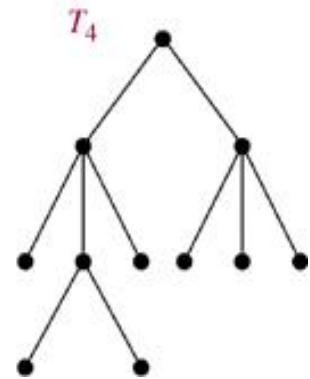
full binary tree



full 3-ary tree



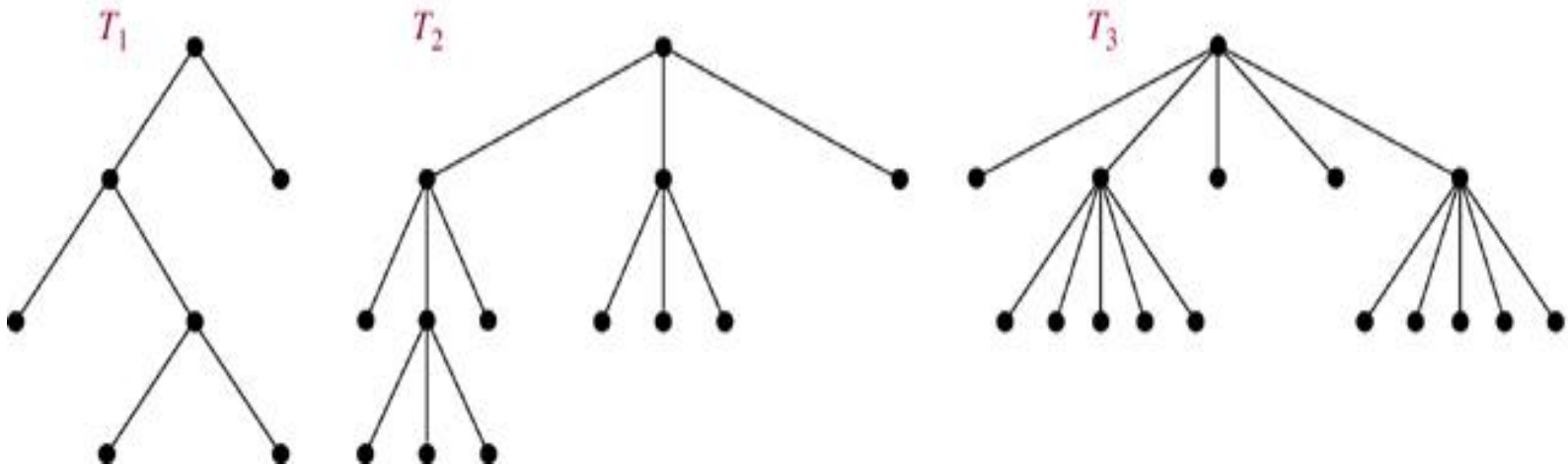
full 5-ary tree



not full 3-ary tree

Properties of Trees

- A tree with n vertices has $n-1$ edges.
- A full m -ary tree with i internal vertices contains $n = mi + 1$ vertices.
- A full m -ary tree with n vertices contains $(n-1)/m$ internal vertices, and hence $n - (n-1)/m = ((m-1)n+1)/m$ leaves.



Complete m-ary Tree

- A **complete m -ary tree** is a full m -ary tree, where every leaf is at the same level.

Example: How many vertices and how many leaves does a complete m -ary tree of height h have?

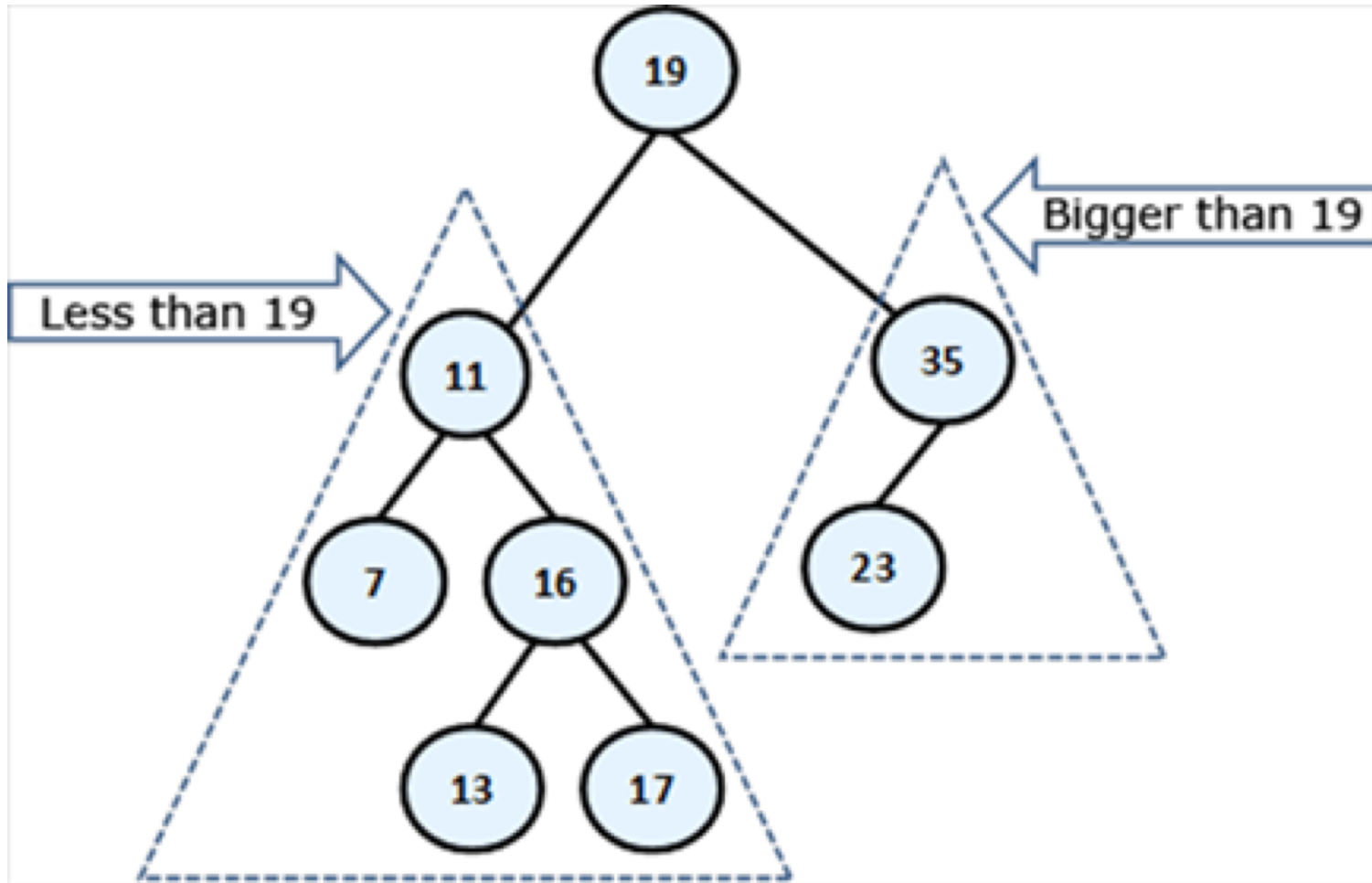
Solution:

$$\# \text{ of vertices} = 1 + m + m^2 + \dots + m^h = (m^{h+1} - 1) / (m - 1)$$

$$\# \text{ of leaves} = m^h$$

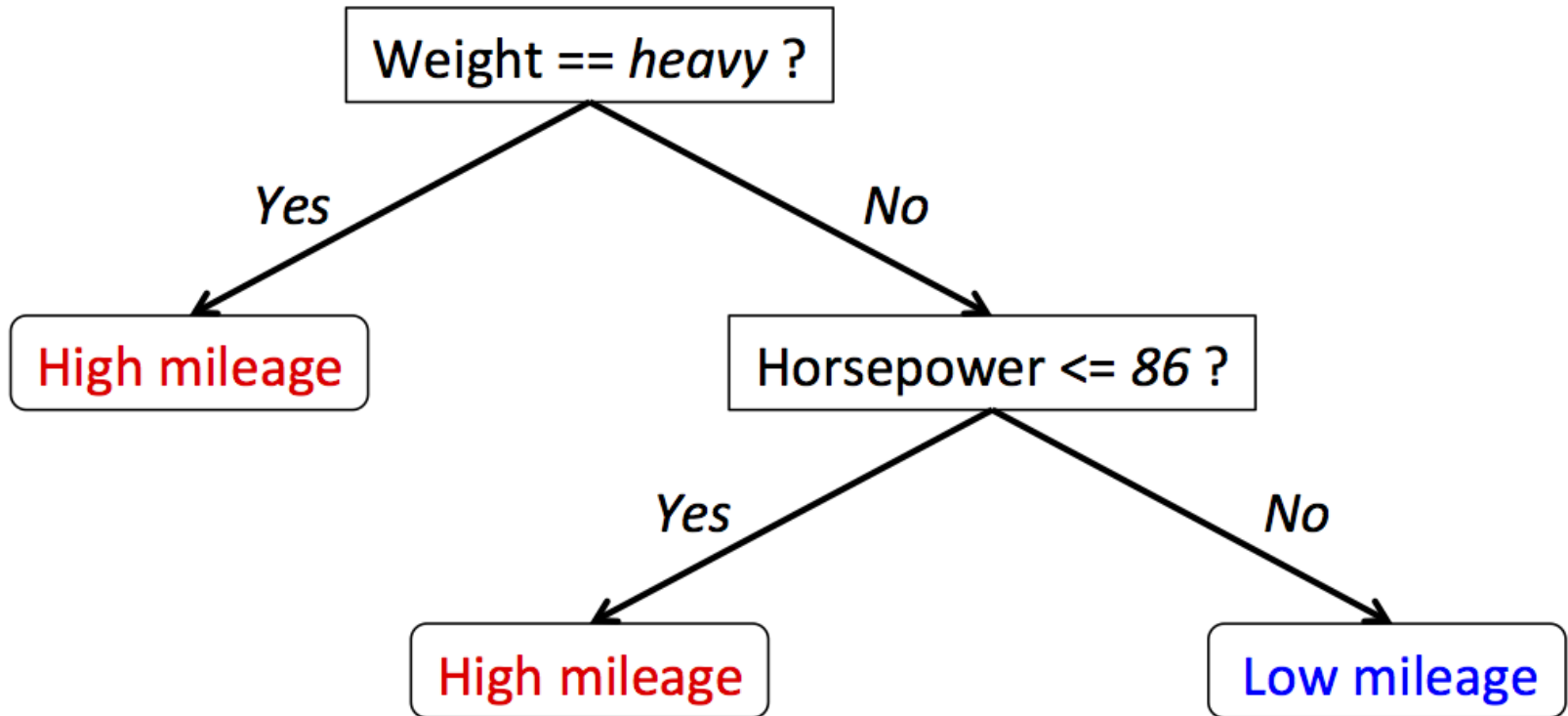
Remark: There are at most m^h leaves in an m -ary tree of height h .

Applications: Binary Search Trees



Applications: Decision Trees

Decision Tree Model for Car Mileage Prediction

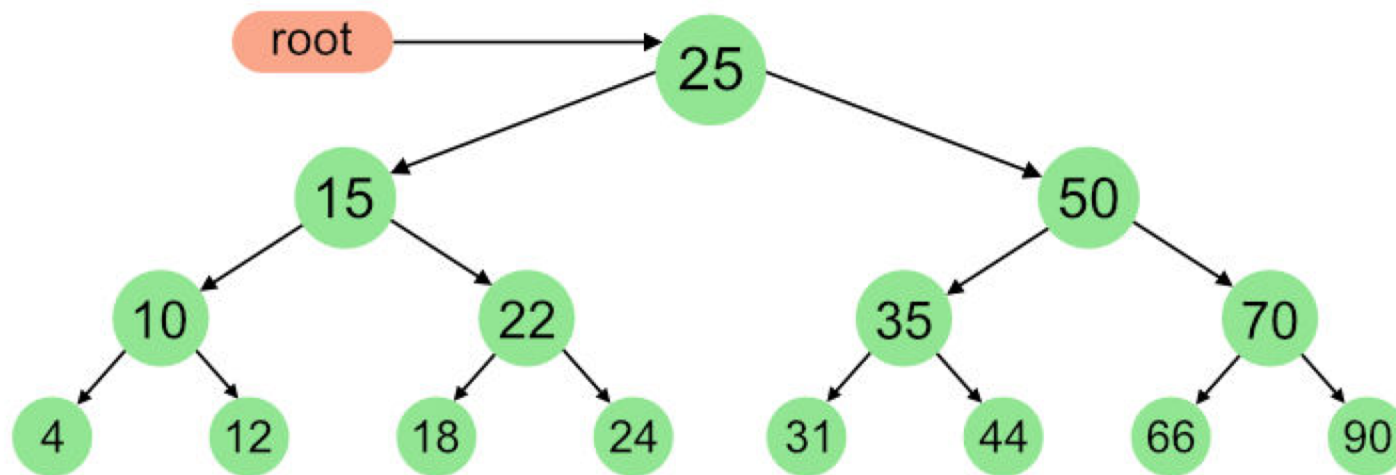


Outline

- Introduction to Trees
- **Tree Traversal**
- Spanning Trees

Tree Traversal

- We need procedures for visiting each vertex of an ordered rooted tree to access data.



A Pre-order traversal visits nodes in the following order:

25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

InOrder(root) visits nodes in the following order:

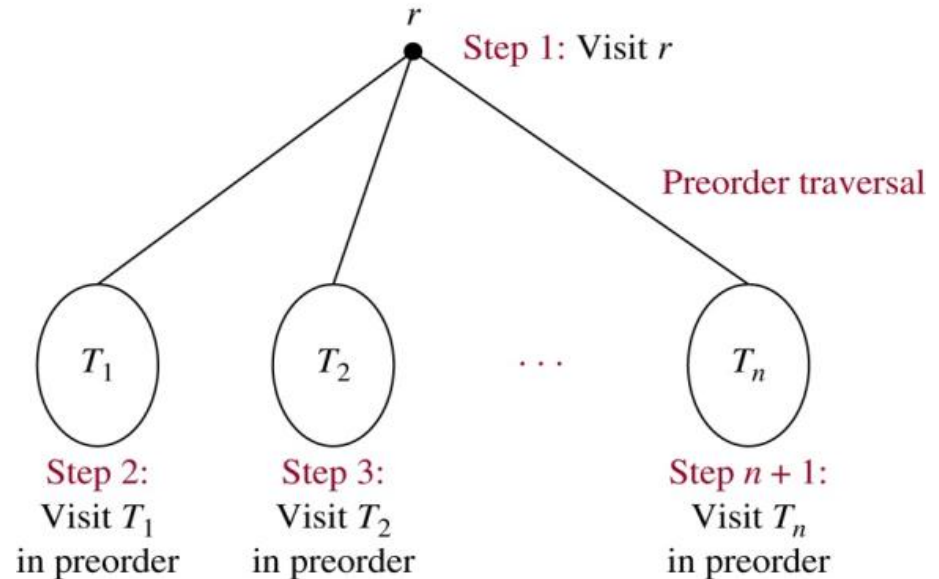
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Post-order traversal visits nodes in the following order:

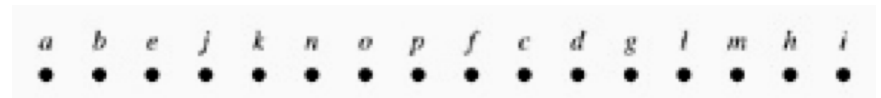
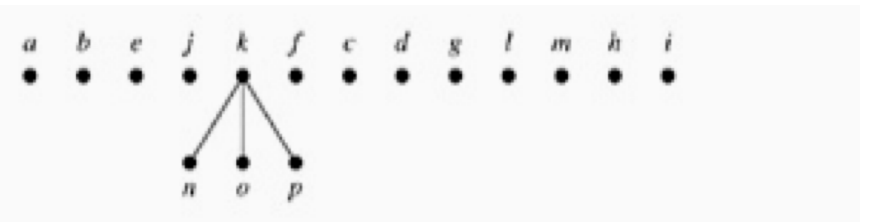
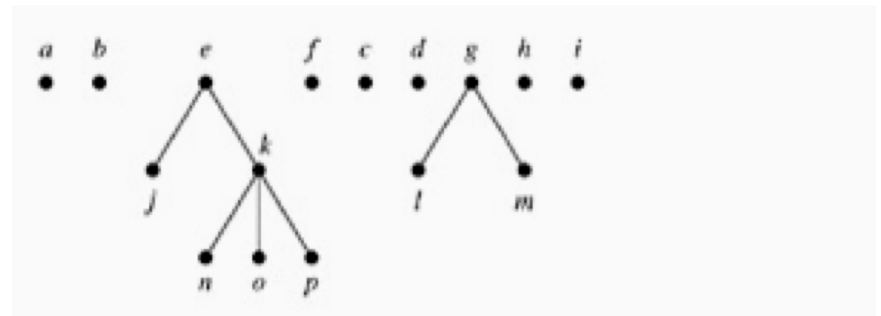
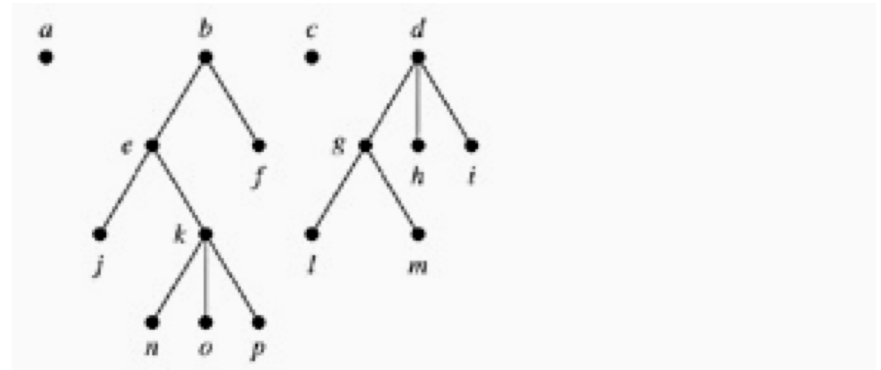
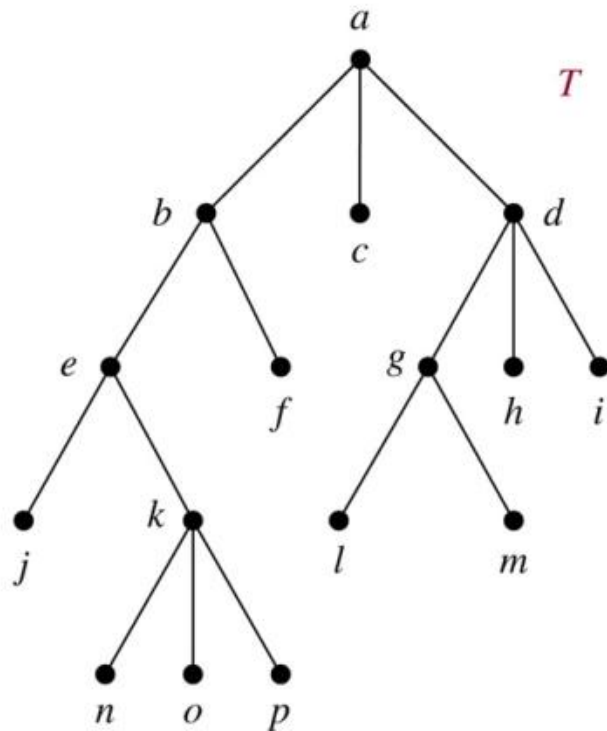
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25

Preorder traversal (Preorder)

```
Procedure preorder( $T$ : ordered  
    rooted tree)  
 $r :=$  root of  $T$   
list  $r$   
for each child  $c$  of  $r$  from left to right  
begin  
     $T(c) :=$  subtree with  $c$  as its root  
    preorder( $T(c)$ )  
end
```

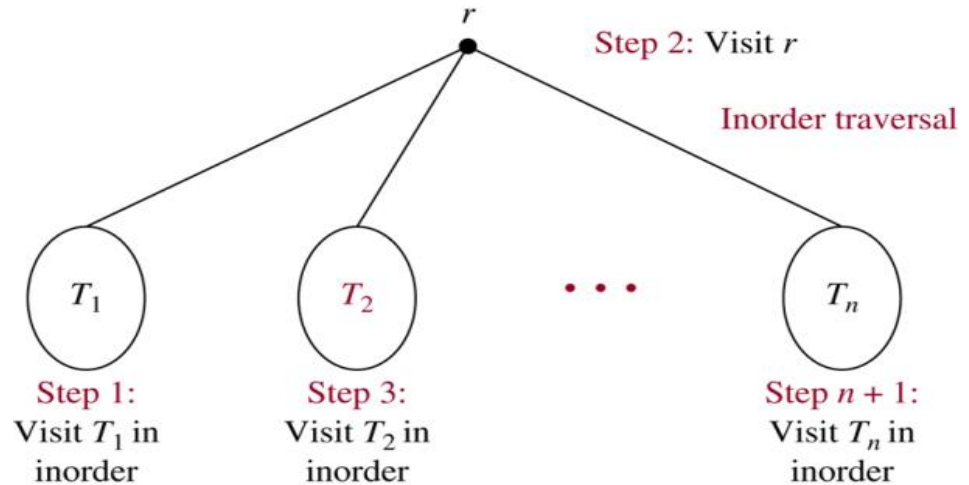


Example

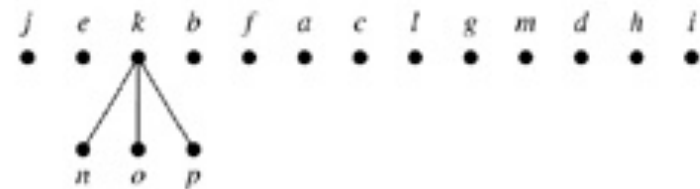
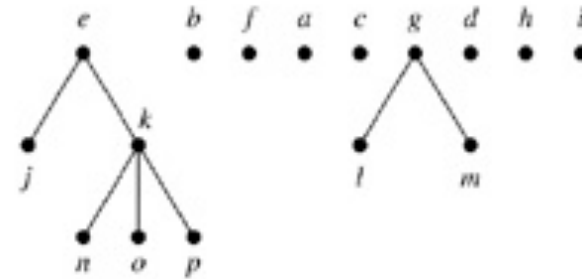
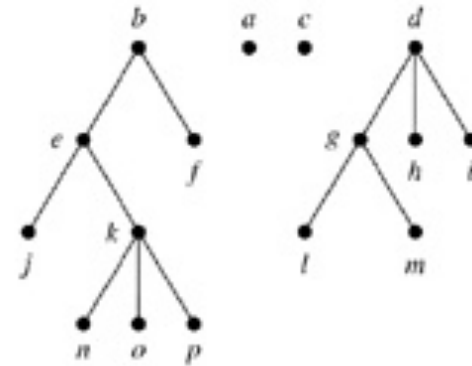
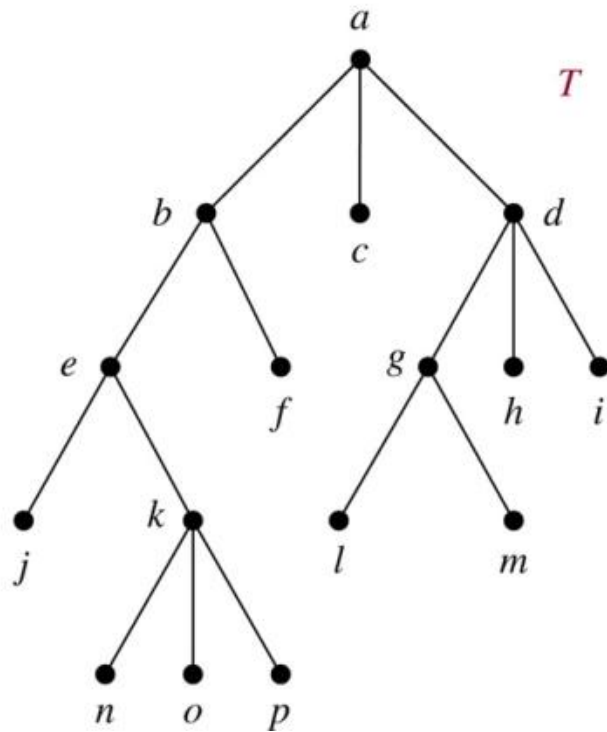


Inorder traversal (Ineorder)

```
Procedure inorder(T: ordered rooted  
    tree)  
  r := root of T  
  If r is a leaf then list r  
  else  
    begin  
      l := first child of r from left to right  
      T(l) := subtree with l as its root  
      inorder(T(l))  
      list r  
      for each child c of r except for l  
      from left to right  
        T(c) := subtree with c as its root  
        inorder(T(c))  
    end
```

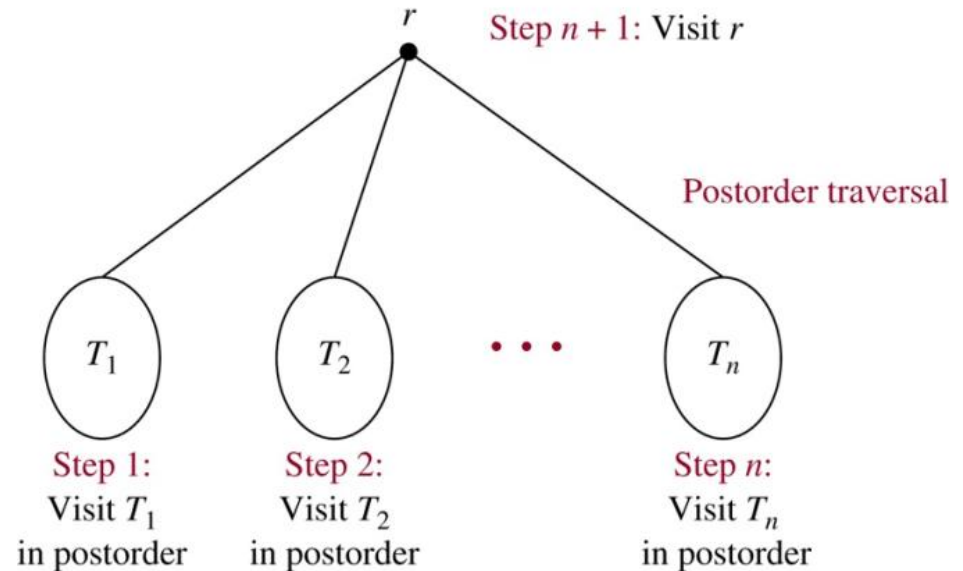


Example

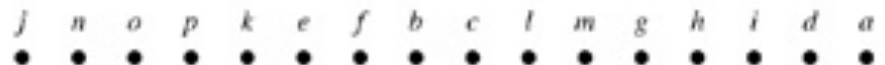
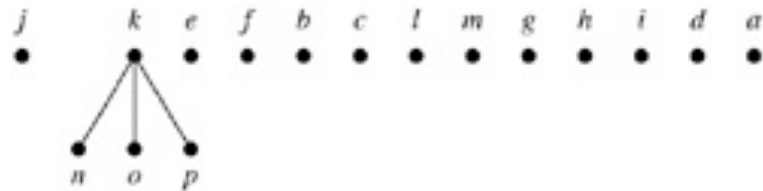
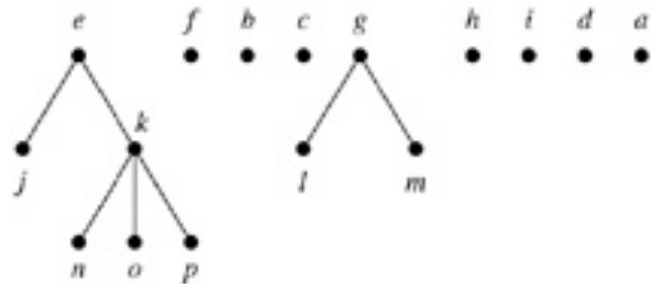
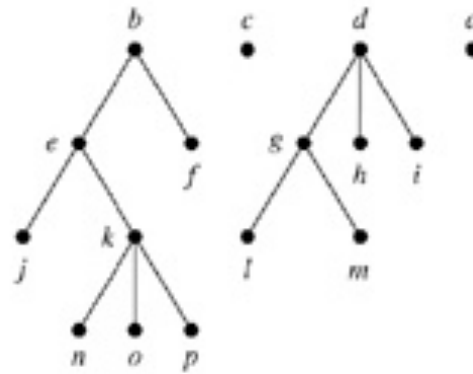
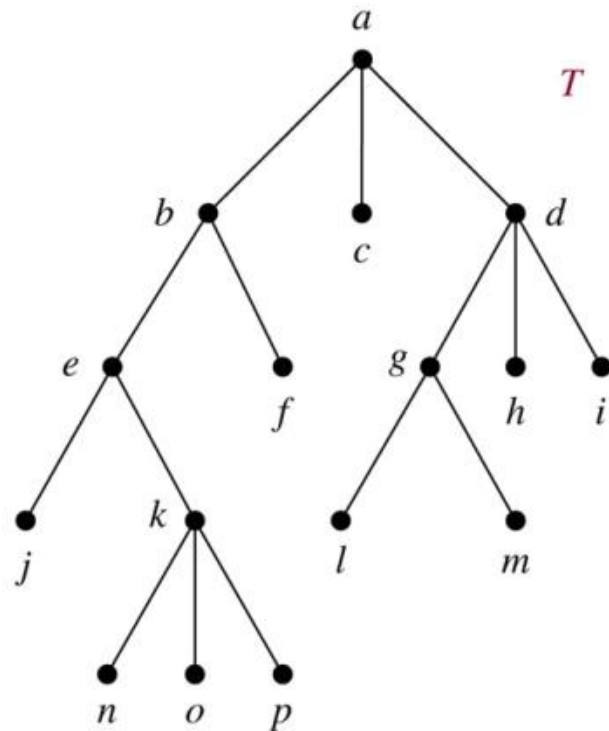


Postorder traversal (Postorder)

```
Procedure postorder( $T$ : ordered  
    rooted tree)  
 $r :=$  root of  $T$   
for each child  $c$  of  $r$  from left to right  
begin  
     $T(c) :=$  subtree with  $c$  as its root  
    postorder( $T(c)$ )  
end  
list  $r$ 
```

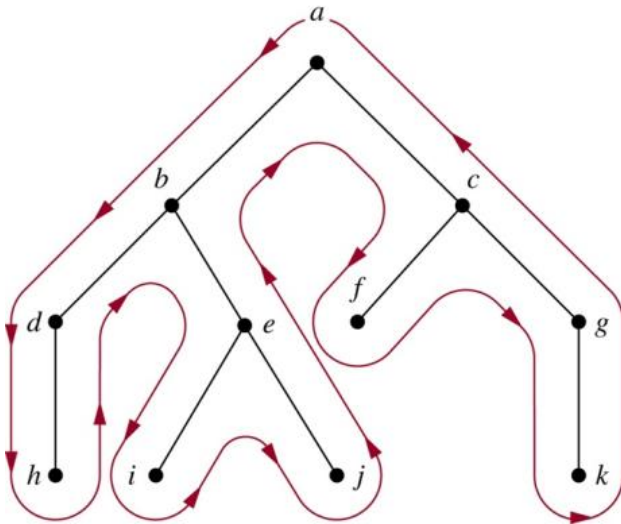


Example



Tree Traversal Representation

- Easy representation: draw a red line around the ordered rooted tree starting at the root, moving along the edges.



Preorder: $a, b, d, h, e, i, j, c, f, g, k$

Inorder: $h, d, b, i, e, j, a, f, c, k, g$

Postorder: $h, d, i, j, e, b, f, k, g, c, a$

- Preorder: listing each vertex the first time this line passes it.
- Inorder: listing a leaf the first time the line passes it and listing each internal vertex the second time the line passes it.
- Postorder: listing a vertex the last time it is passed on the way back up to its parent.

Example

- We can represent complicated expressions, such as compound propositions, combinations of sets, and arithmetic expressions using ordered rooted trees.

Example Find the ordered rooted tree for $((x+y) \times 2) + ((x-4)/3)$.

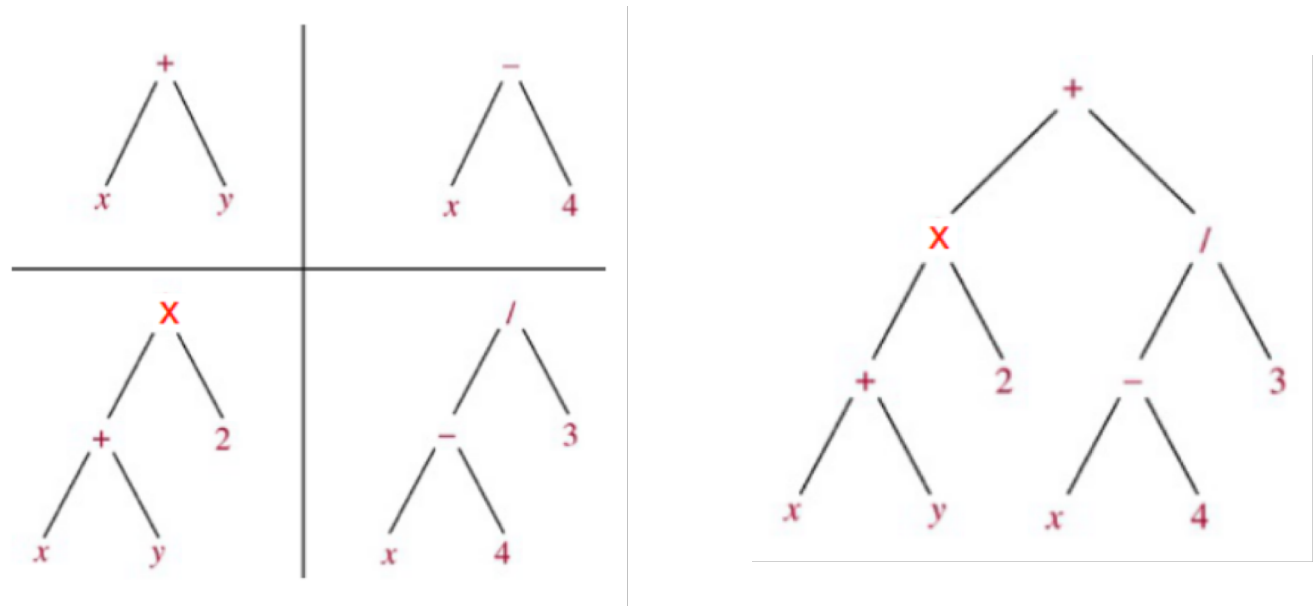
Solution:

leaf:

variable

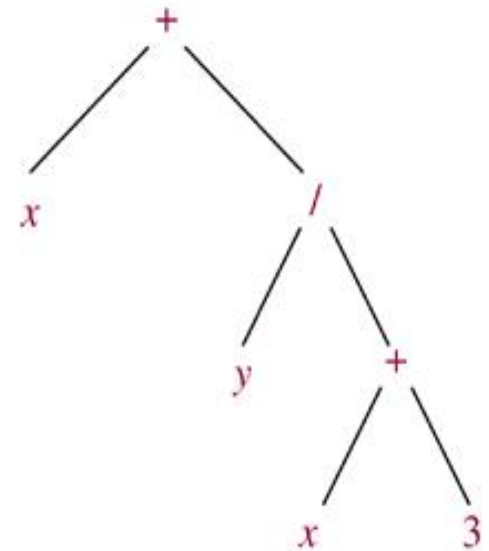
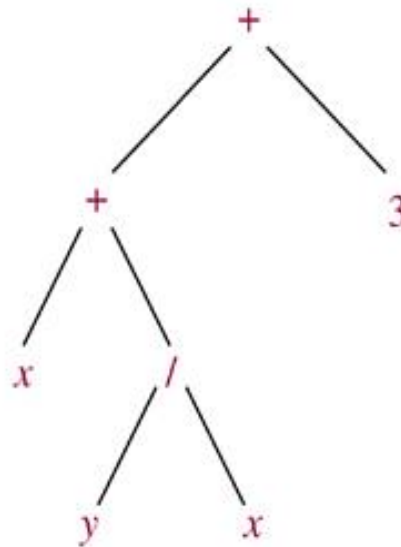
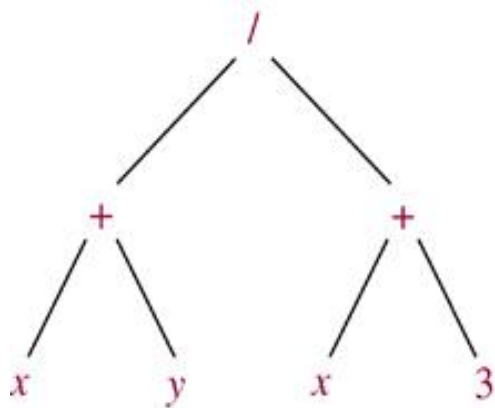
internal vertex:

operation on
its left and right
subtrees



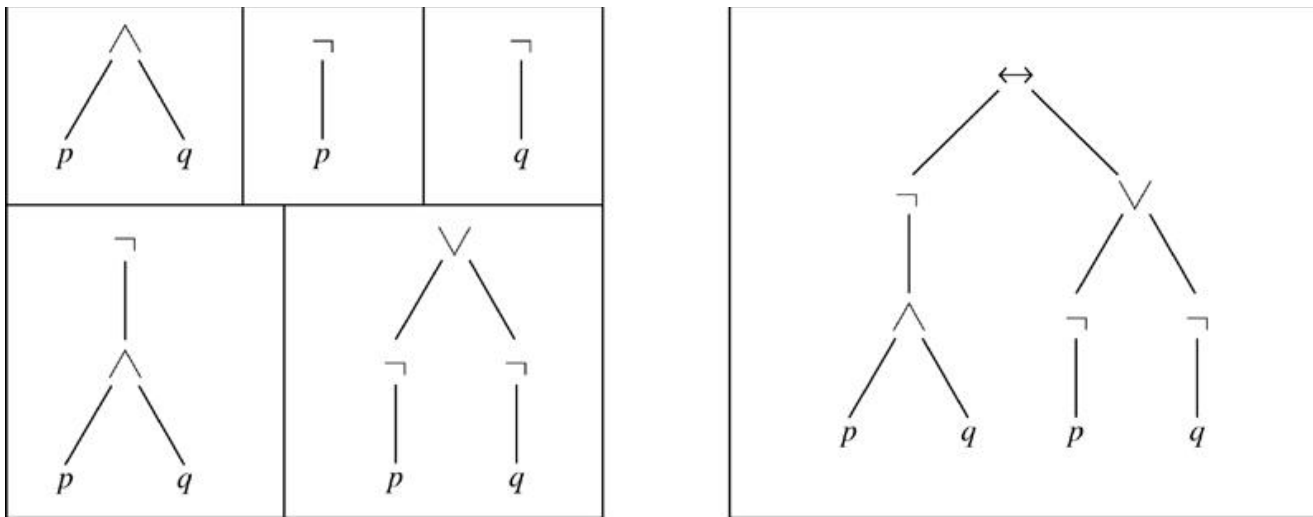
Example

- The following binary trees represent the expressions:
 $(x+y)/(x+3)$, $(x+(y/x))+3$, $x+(y/(x+3))$.
All their inorder traversals lead to $x+y/x+3 \Rightarrow$
ambiguous $\Rightarrow$ need parentheses



Example

- Find the ordered rooted tree representing the compound proposition $(\neg(p \wedge q)) \leftrightarrow (\neg p \vee \neg q)$. Then use this rooted tree to find the prefix, postfix, and infix forms of this expression.



prefix: $\leftrightarrow \neg \wedge p q \vee \neg p \neg q$

infix: $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$

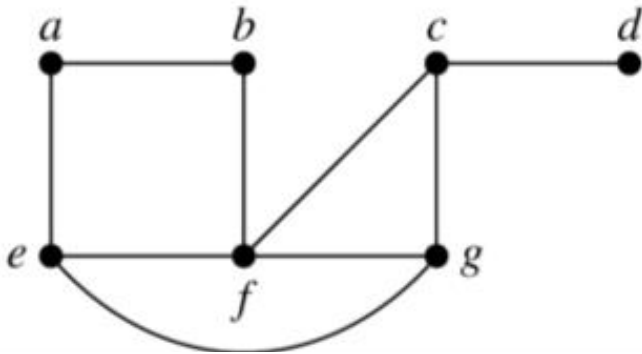
postfix: $p q \wedge \neg p \neg q \neg \vee \leftrightarrow$

Outline

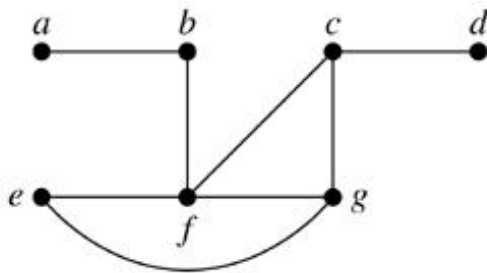
- Introduction to Trees
- Tree Traversal
- **Spanning Trees**

Spanning Trees

- 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 .

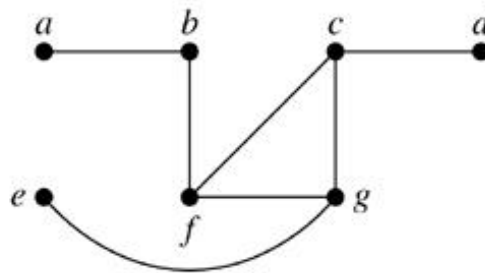


Remove an edge from any circuit.
(repeat until no circuit exists)



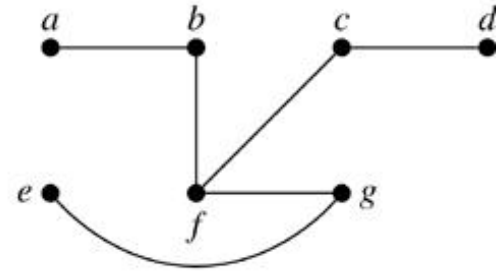
Edge removed: $\{a, e\}$

(a)



$\{e, f\}$

(b)

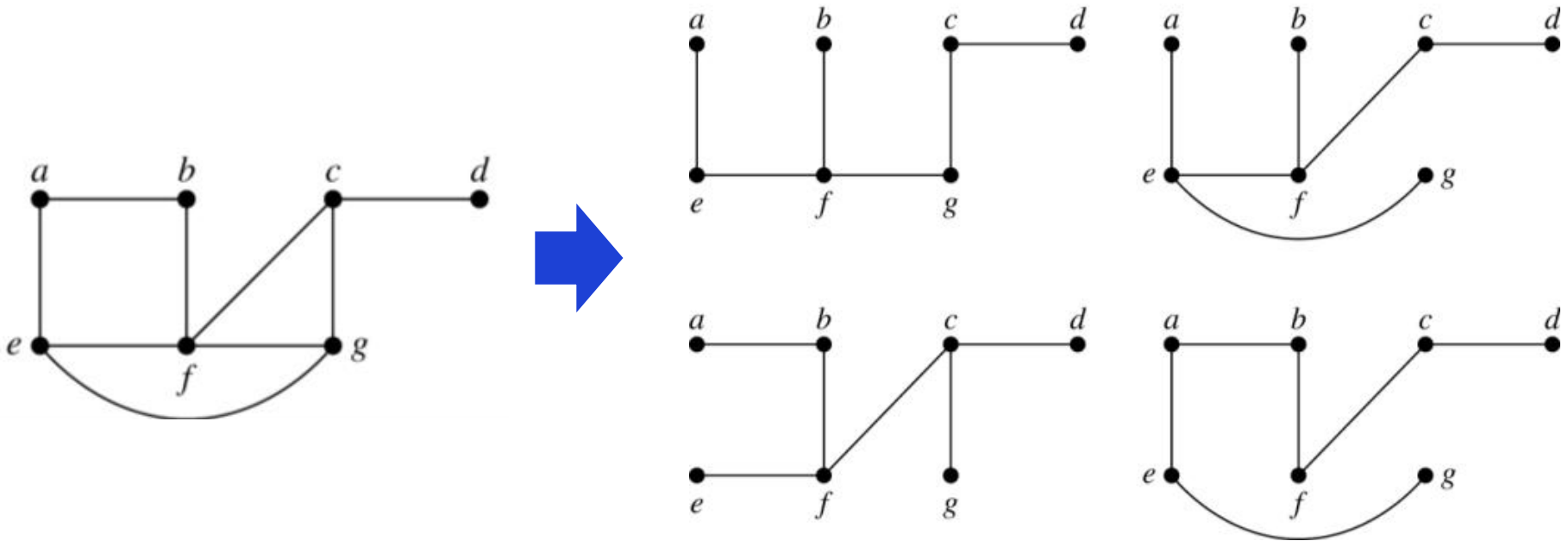


$\{c, g\}$

(c)

Spanning Trees

Four spanning trees of G :



Theorem: A simple graph is connected if and only if it has a spanning tree.

Depth-First Search (DFS)

Procedure *DFS*(G : connected graph with vertices $v_1, v_2, \dots, v_n$)

$T :=$ tree consisting only of the vertex v_1

visit(v_1)

procedure *visit*(v : vertex of G)

for each vertex w adjacent to v and not yet in T

begin

 add vertex w and edge $\{v, w\}$ to T

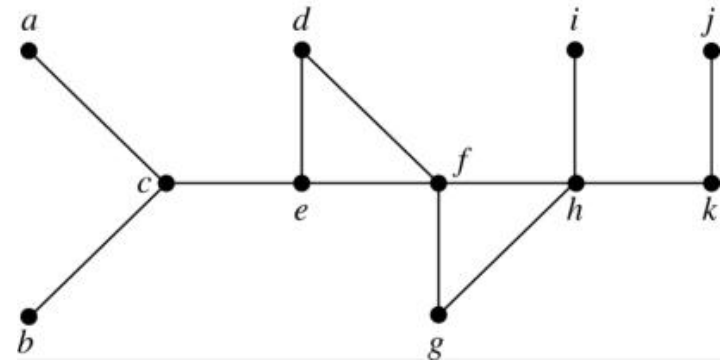
visit(w)

end

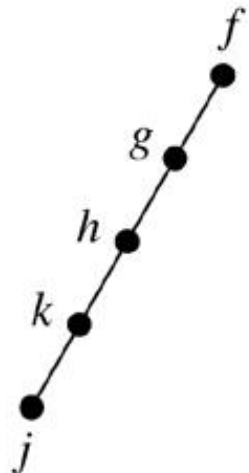
Example

- Use depth-first search to find a spanning tree for the graph.

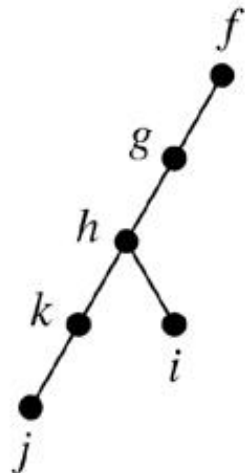
Solution. (arbitrarily start with the vertex f)



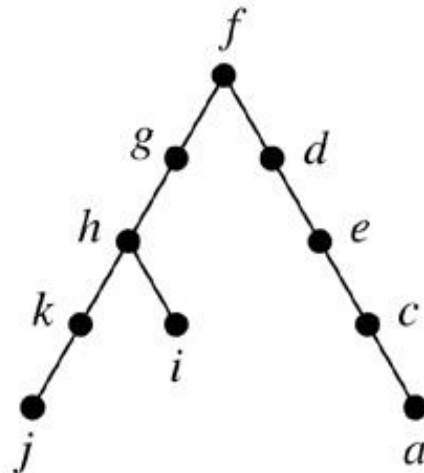
f



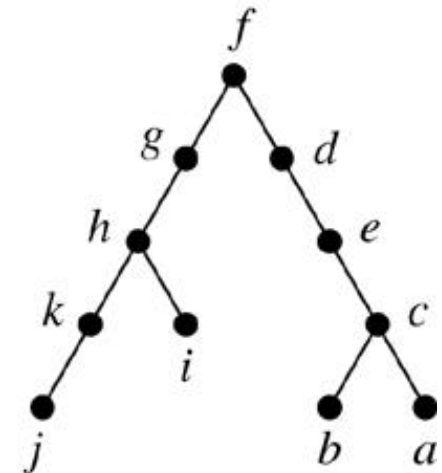
(a)



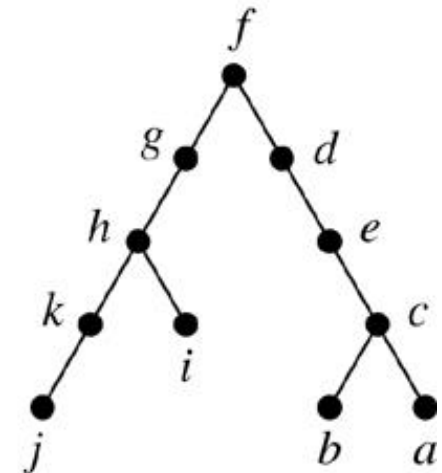
(b)



(c)



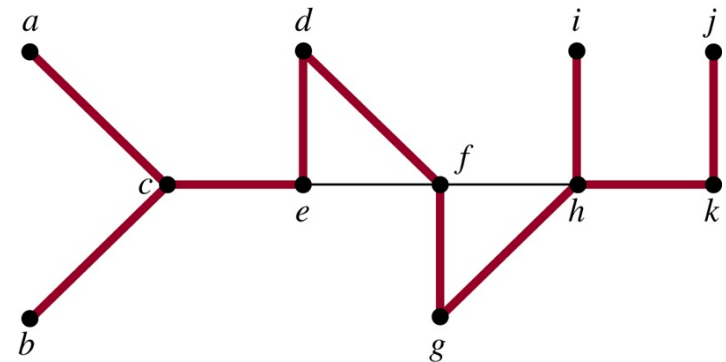
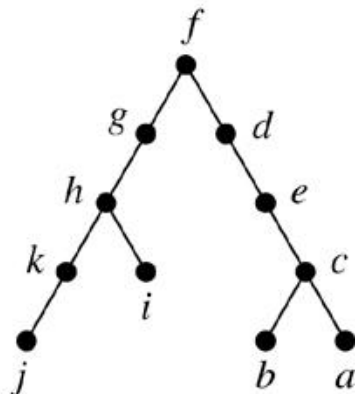
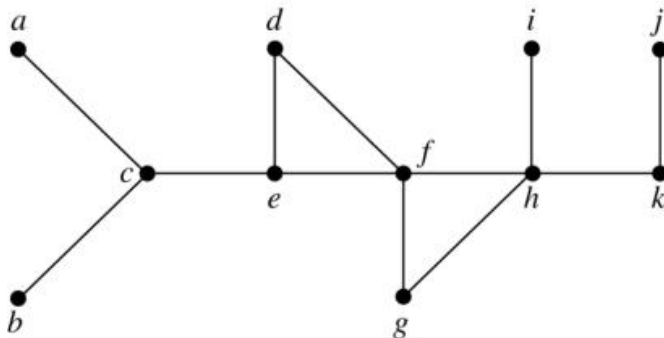
(d)



(e)

Example

- The edges selected by DFS of a graph are called **tree edges**. All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree. These edges are called **back edges**.



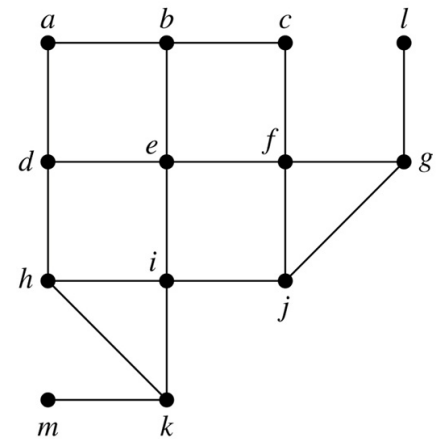
The tree edges (**red**)
and back edges (**black**)

Breadth-First Search (BFS)

```
Procedure BFS(G: connected graph with vertices  $v_1, v_2, \dots, v_n$ )  
   $T :=$  tree consisting only of vertex  $v_1$   
   $L :=$  empty list  
  put  $v_1$  in the list  $L$  of unprocessed vertices  
  while  $L$  is not empty  
  begin  
    remove the first vertex  $v$  from  $L$   
    for each neighbor  $w$  of  $v$   
      if  $w$  is not in  $L$  and not in  $T$  then  
        begin  
          add  $w$  to the end of the list  $L$   
          add  $w$  and edge  $\{v, w\}$  to  $T$   
        end  
      end  
    end  
  end
```

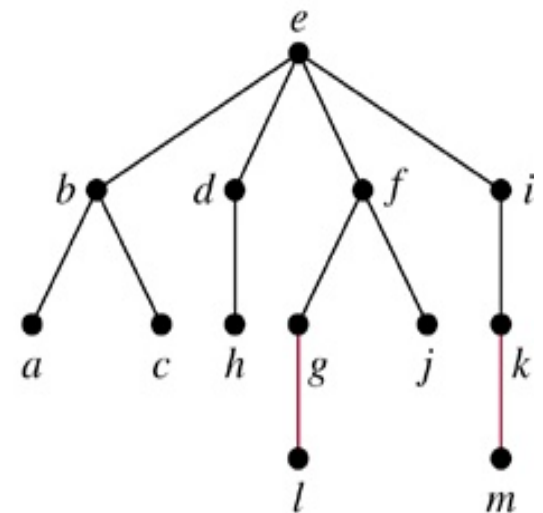
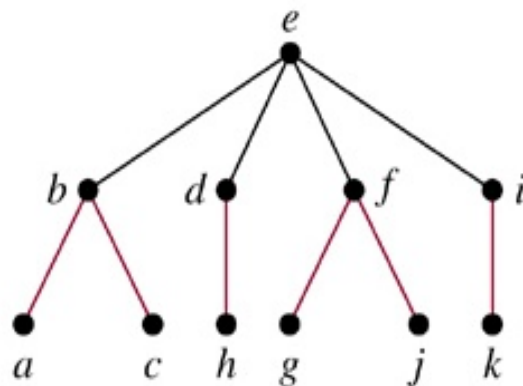
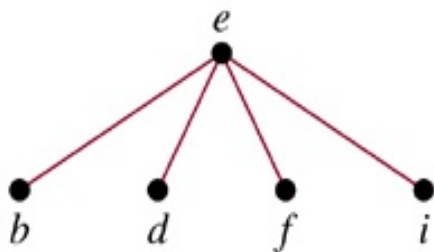
Example

- Use breadth-first search to find a spanning tree for the graph



Solution: (arbitrarily start with the vertex e)

e



Minimum Spanning Trees

- A **minimum spanning tree** of G is a spanning tree with smallest sum of weights of its edges.
- Prim's Algorithm:

Procedure *Prim*(G : connected weighted 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

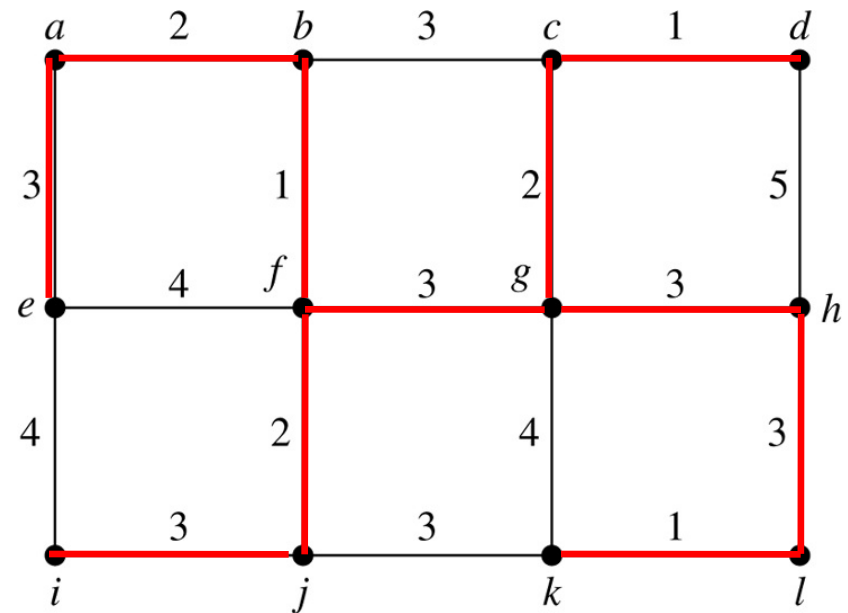
end { T is a minimum spanning tree of G }

Example

- Use Prim's algorithm to find a minimum spanning tree of G .

Solution:

Choice	Edge	Weight
1	$\{b, f\}$	1
2	$\{a, b\}$	2
3	$\{f, j\}$	2
4	$\{a, e\}$	3
5	$\{i, j\}$	3
6	$\{f, g\}$	3
7	$\{c, g\}$	2
8	$\{c, d\}$	1
9	$\{g, h\}$	3
10	$\{h, l\}$	3
11	$\{k, l\}$	1
Total:		24



Kruskal Algorithm

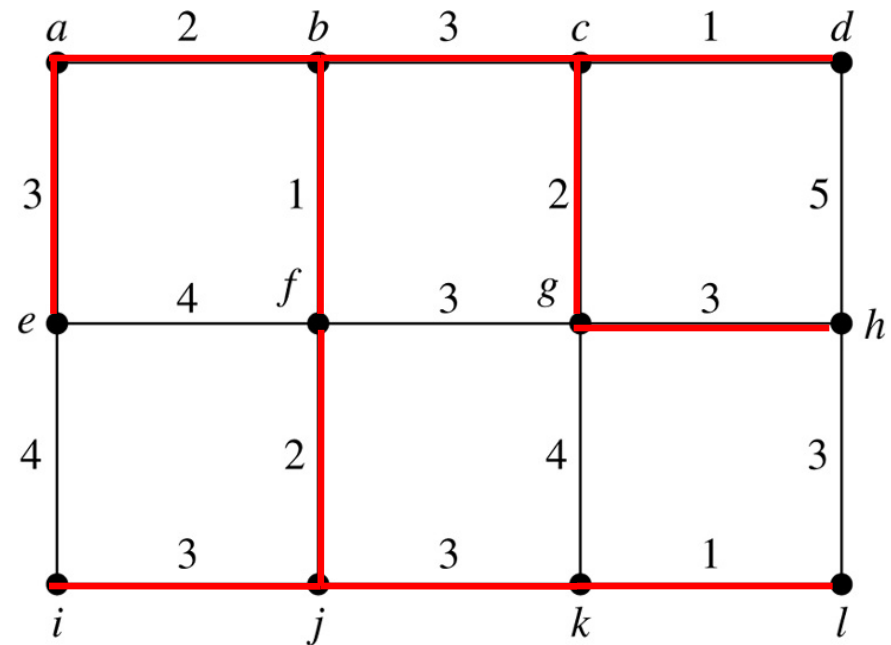
```
Procedure Kruskal( $G$ : connected weighted undirected graph with  $n$  vertices)  
 $T :=$  empty graph  
for  $i := 1$  to  $n-1$   
begin  
     $e :=$  any edge in  $G$  with smallest weight that does not form a simple  
        circuit when added to  $T$   
     $T := T$  with  $e$  added  
end { $T$  is a minimum spanning tree of  $G$ }
```

Example

- Use Kruskal Algorithm to find a minimum spanning tree of G .

Solution:

Choice	Edge	Weight
1	$\{c, d\}$	1
2	$\{k, l\}$	1
3	$\{b, f\}$	1
4	$\{c, g\}$	2
5	$\{a, b\}$	2
6	$\{f, j\}$	2
7	$\{b, c\}$	3
8	$\{j, k\}$	3
9	$\{g, h\}$	3
10	$\{i, j\}$	3
11	$\{a, e\}$	3
Total:		<u>24</u>



Next class

- Topic: Recap and Review (1)
- Pre-class reading: Chap 1-11

