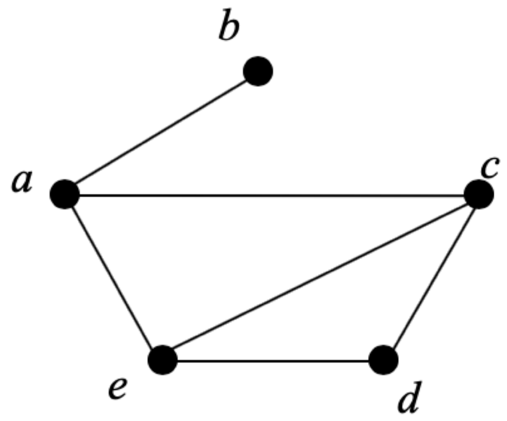
Lecture 12: Trees

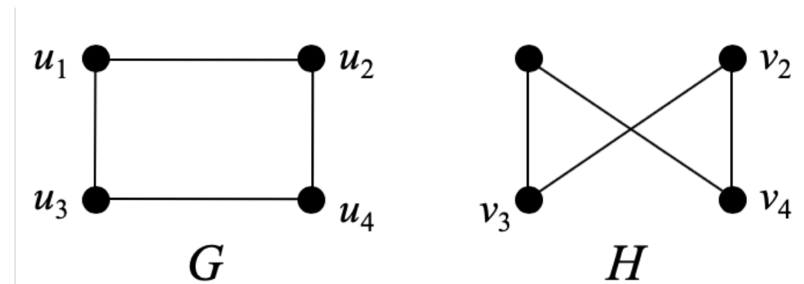
Dr. Chengjiang Long
Computer Vision Researcher at Kitware Inc.
Adjunct Professor at SUNY at Albany.
Email: **clong2@albany.edu**

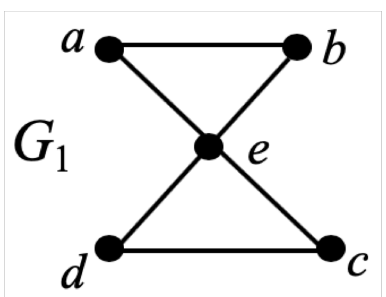
Recap Previous Lecture

- Graph and Its Representations
- Euler and Hamiltonian Paths

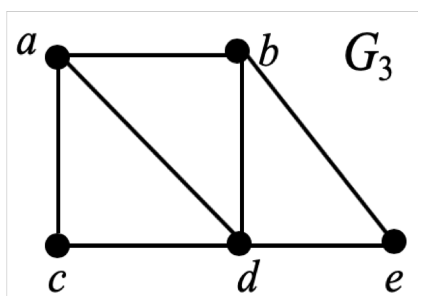


Vertex	Adjacent Vertices
a	*b,c,e*
b	*a*
c	*a,d,e*
d	*c,e*
e	*a,c,d*

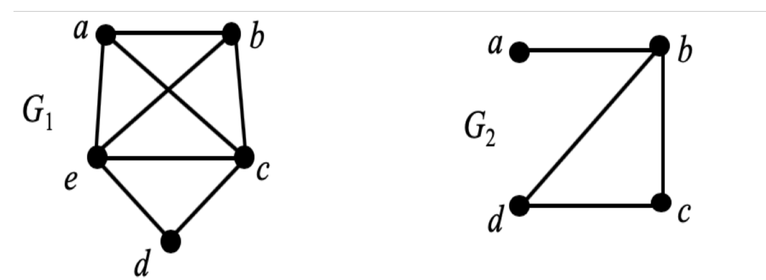




Euler circuit



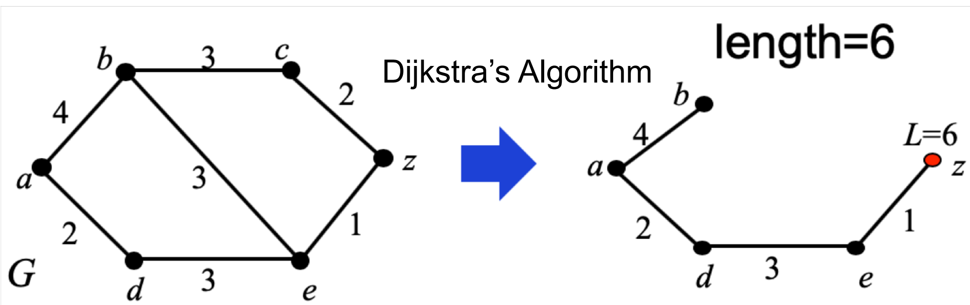
Euler path

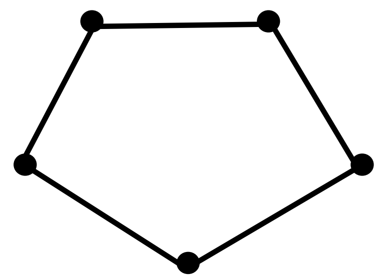


Hamilton circuit: G_1 Hamilton path: G_1,G_2

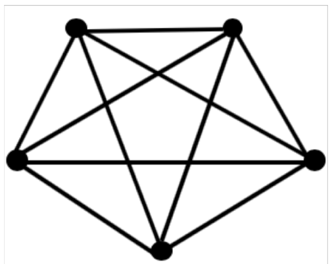
Recap Previous Lecture

- Shortest Path Problem
- Planar Graph and Graph Coloring

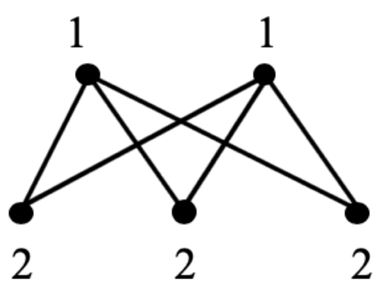




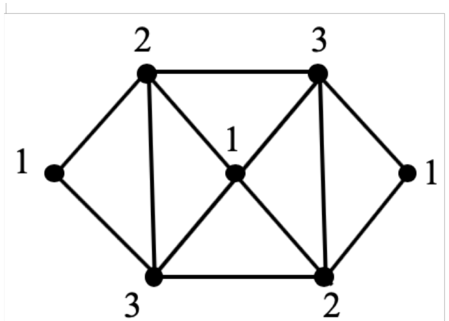
$\chi(C_n) = 2$ if n is even, $\chi(C_n) = 3$ if n is odd.



$\chi(K_n)=n$

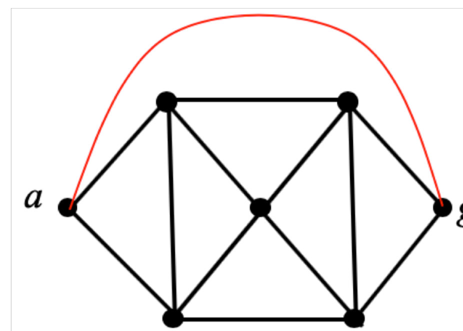


$\chi(G) = 2.$



G

$\chi(G)\geq 3$



H

$\chi(H) \geq 4$

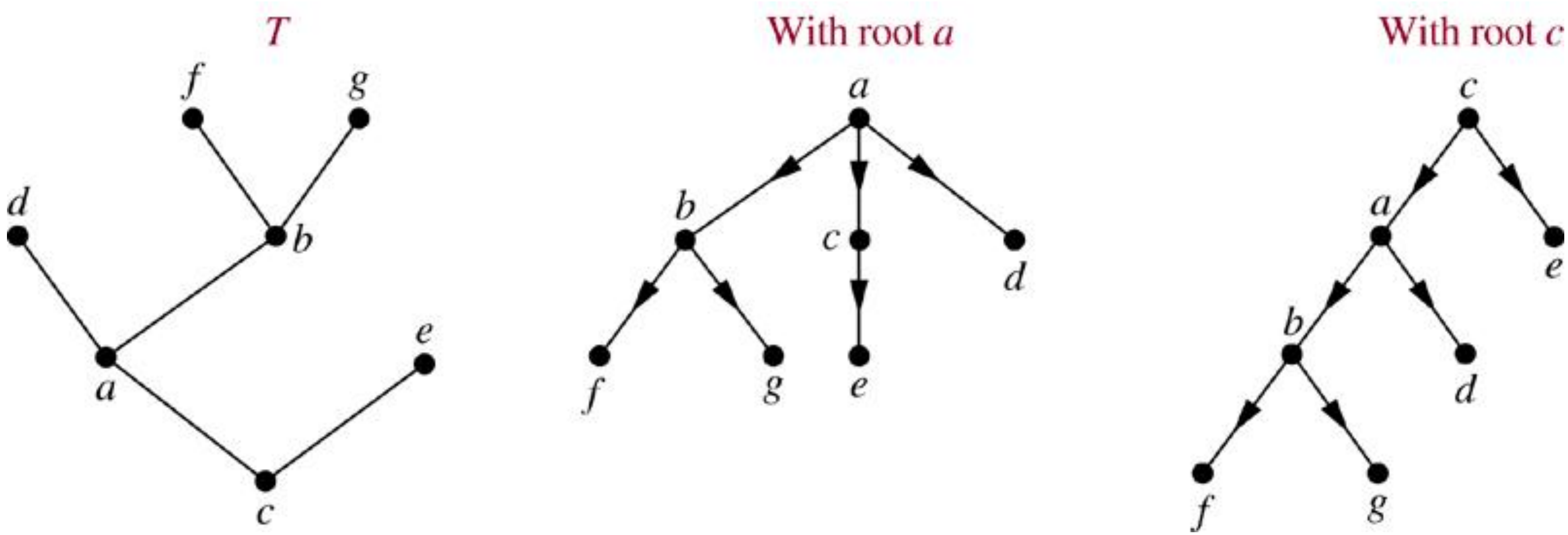
Outline

- Trees
- Applications of Trees
- About Final Project

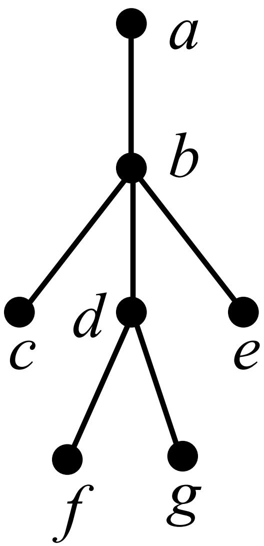
Trees

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**,

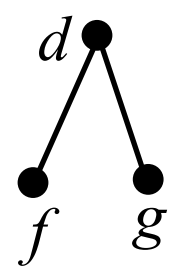
a, b, d are **ancestors** of f

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

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

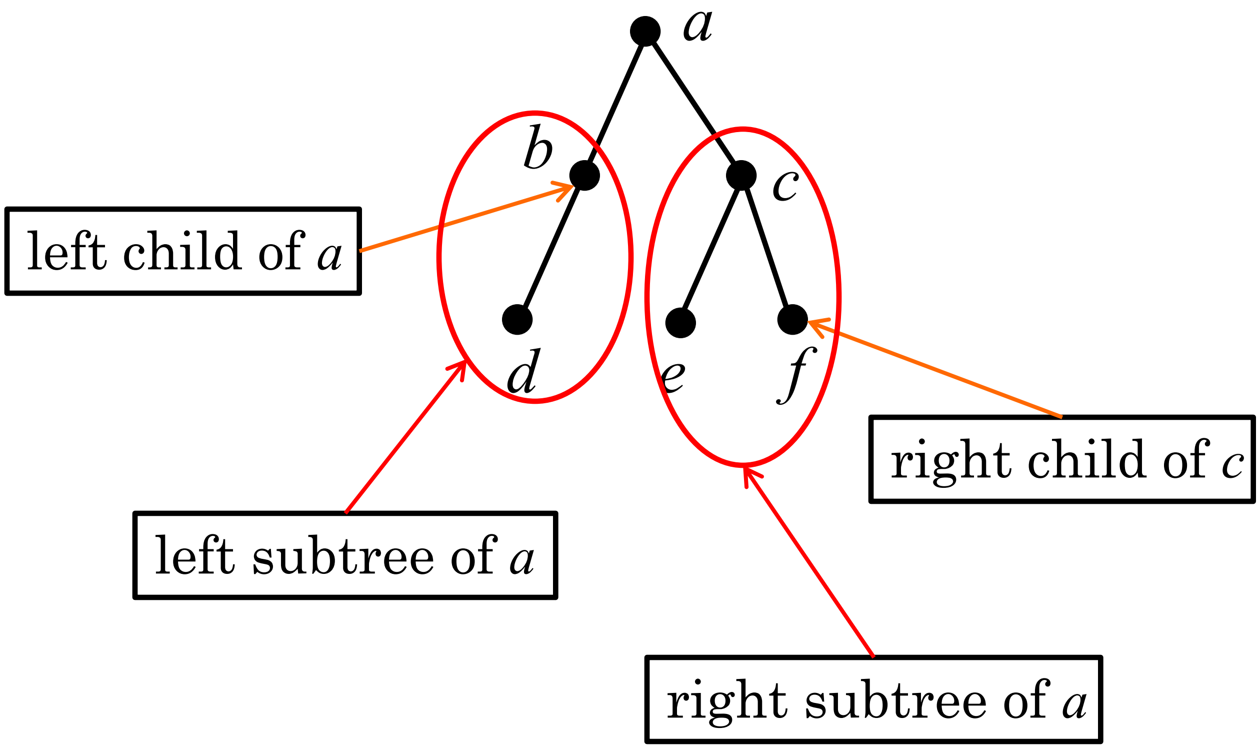
a, b, d are **internal vertices** of the tree (at least one child)

subtree with d as its root:



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

Binary Tree


a
b
c
left child of a
d
e
f
right child of c
left subtree of a
right subtree of a

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.

height = 4

level

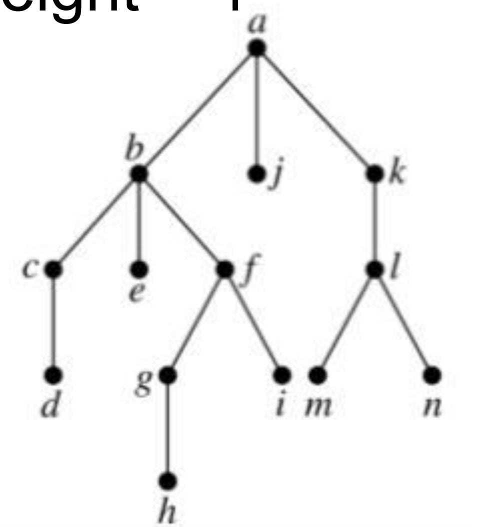
0

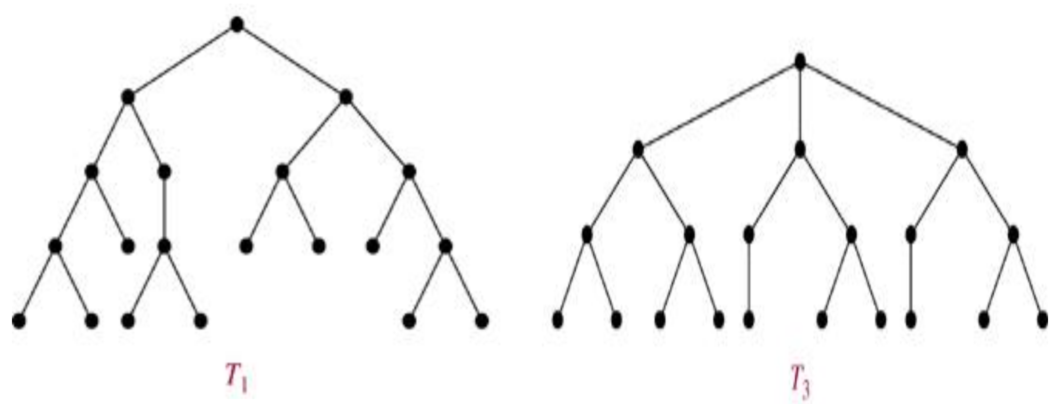
1

2

3

4

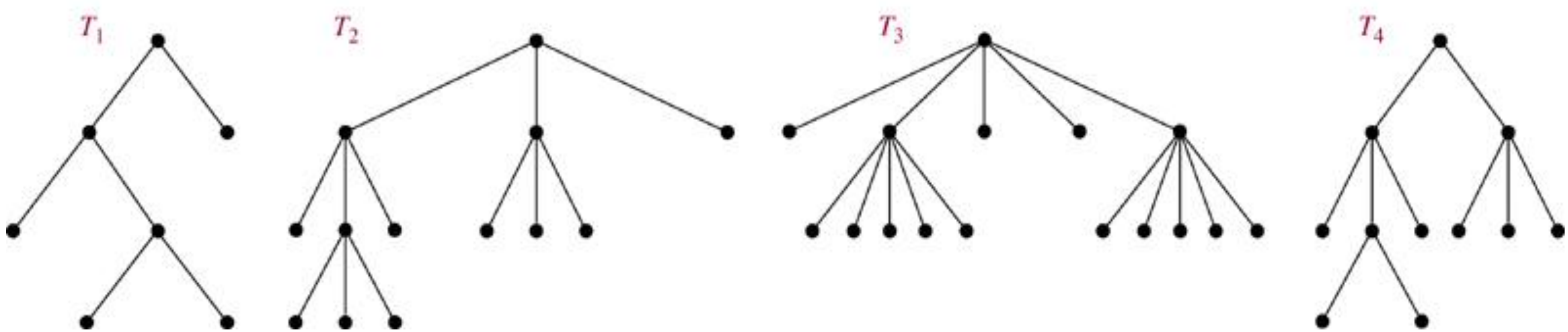




Remark: A rooted m-ary tree of height h is balanced if all leaves are at levels h or $h-1$.

m-ary Tree

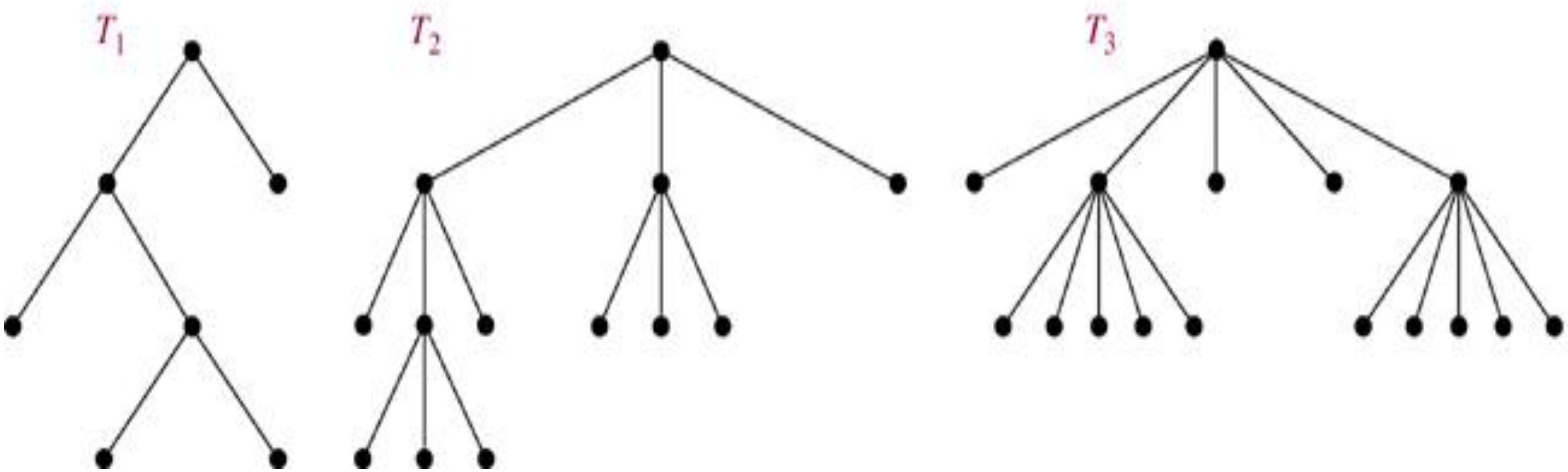
- A rooted tree is called an m-ary tree if every internal vetex has no more than m children.
- The tree is called a full m-ary tree if every internal vertex has exactly m children. An m-ary tree with m=2 is called a binary tree.



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?

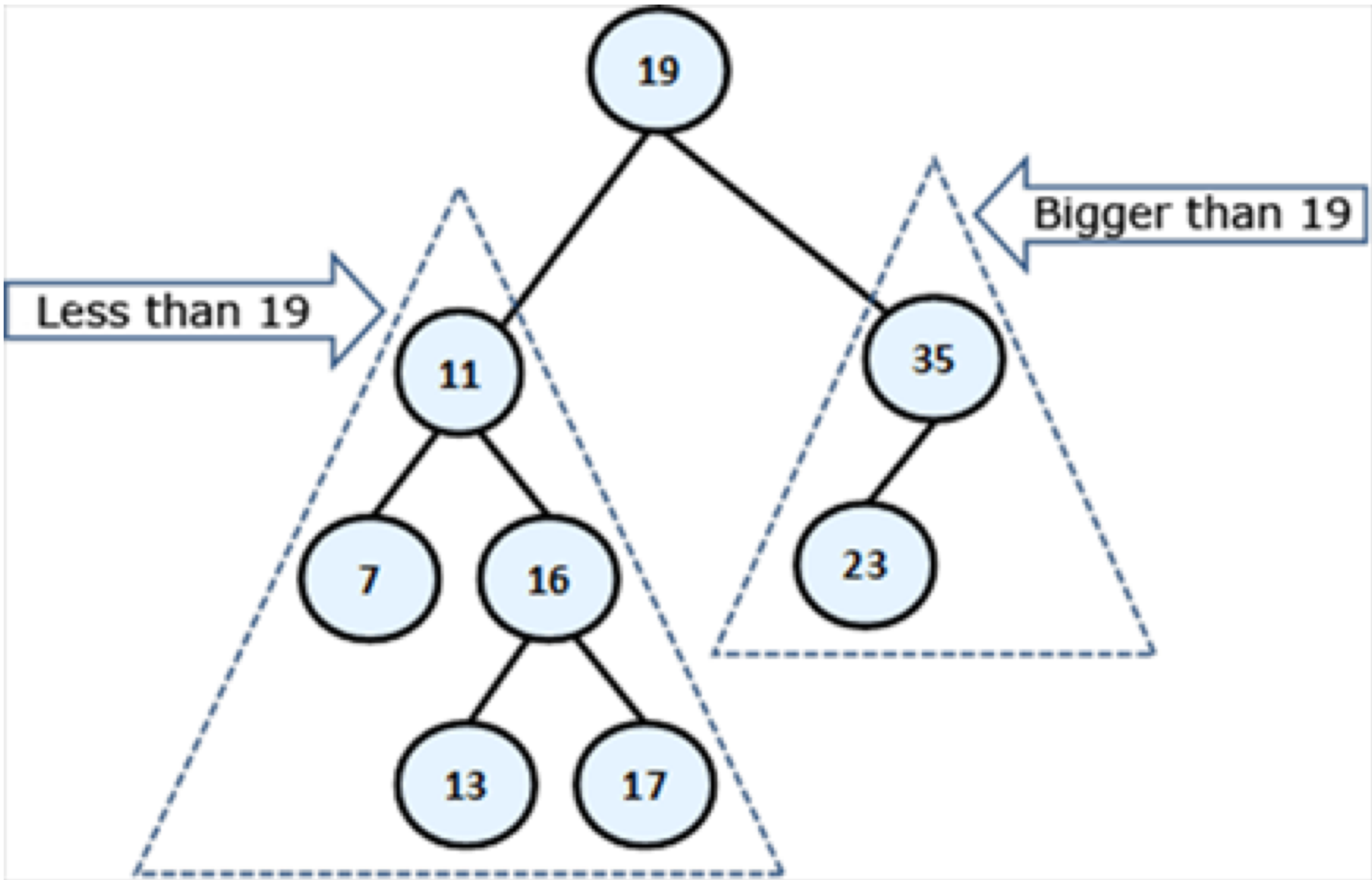
Solution:

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

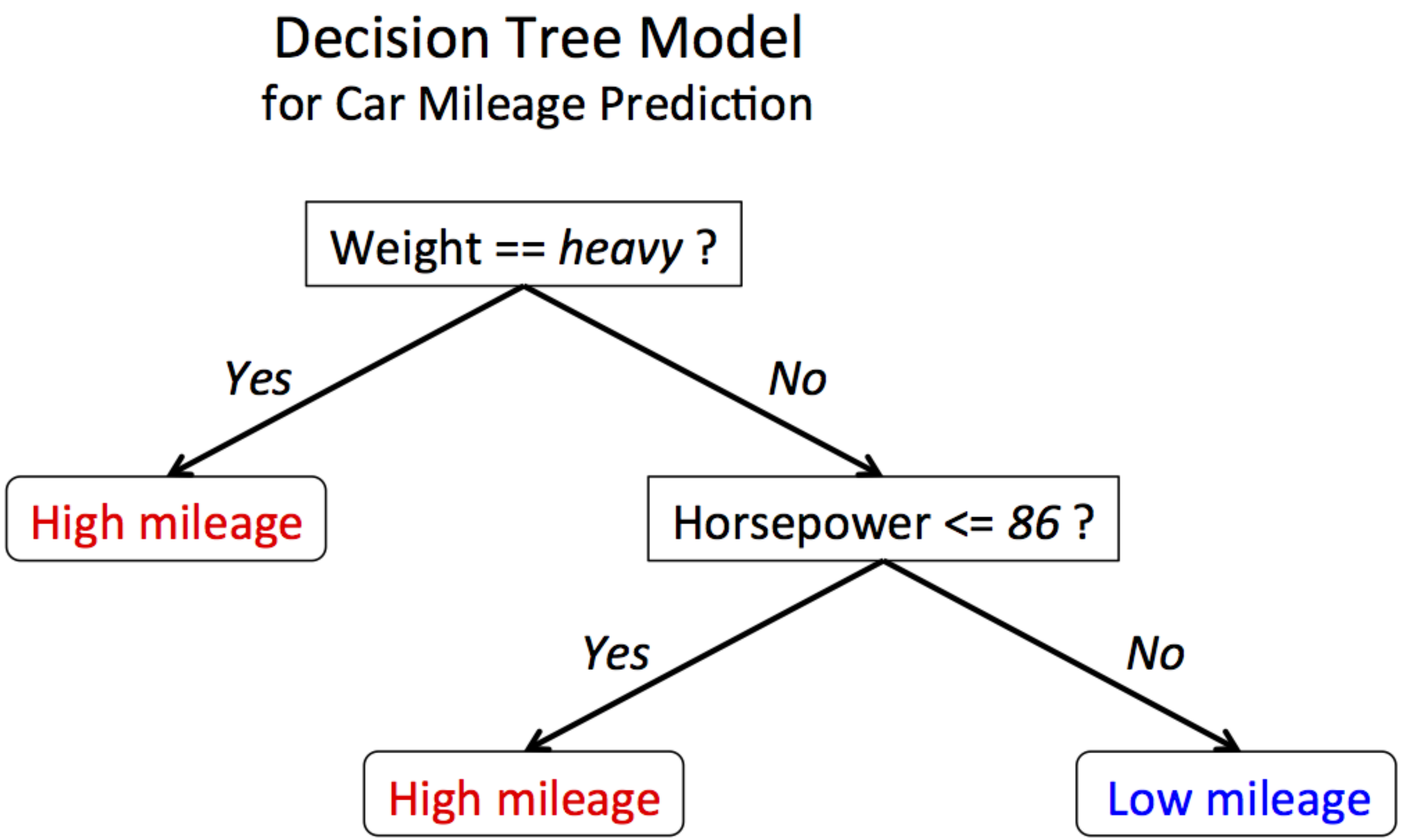
# of leaves = m^h

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

Applications: Binary Search Trees

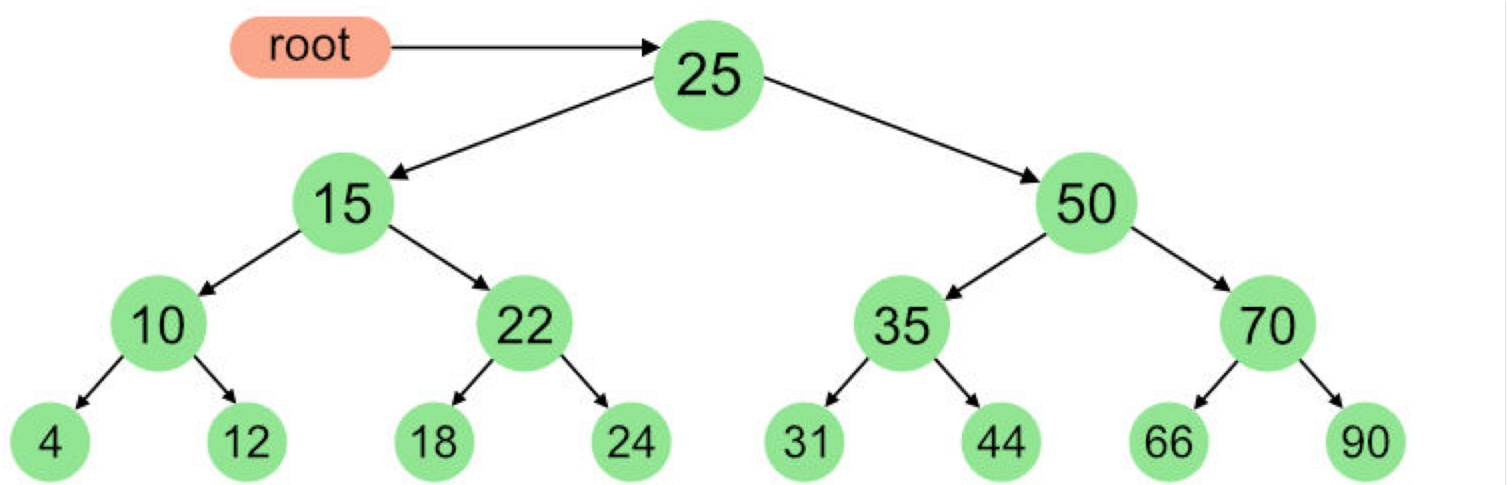


Applications: Decision Trees



Tree Traversal

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



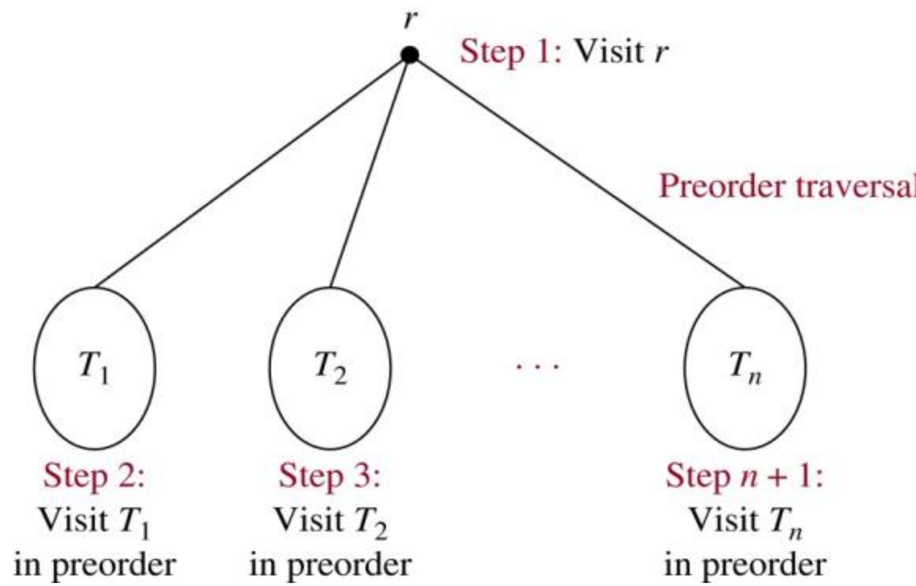
A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

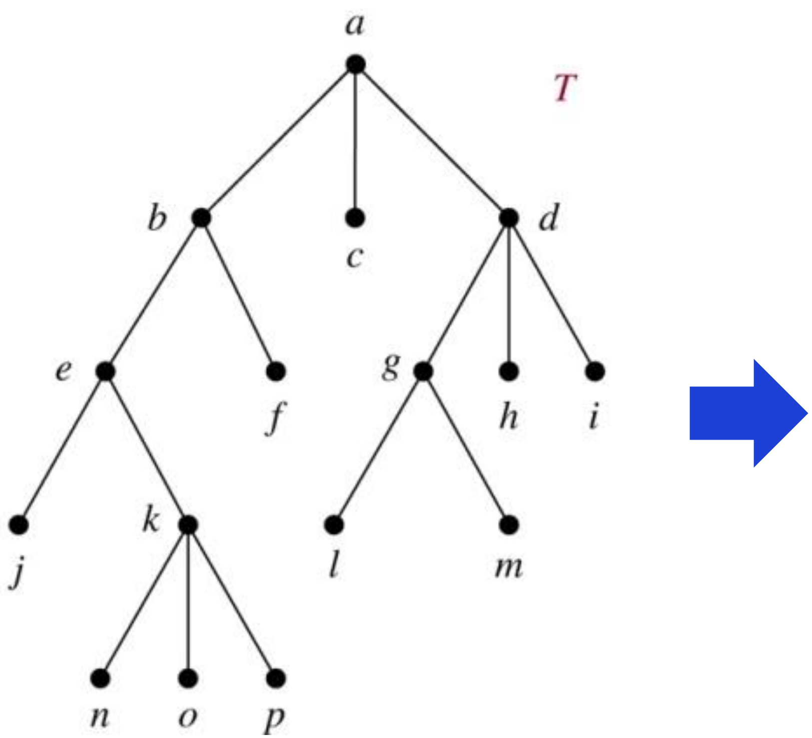
A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25

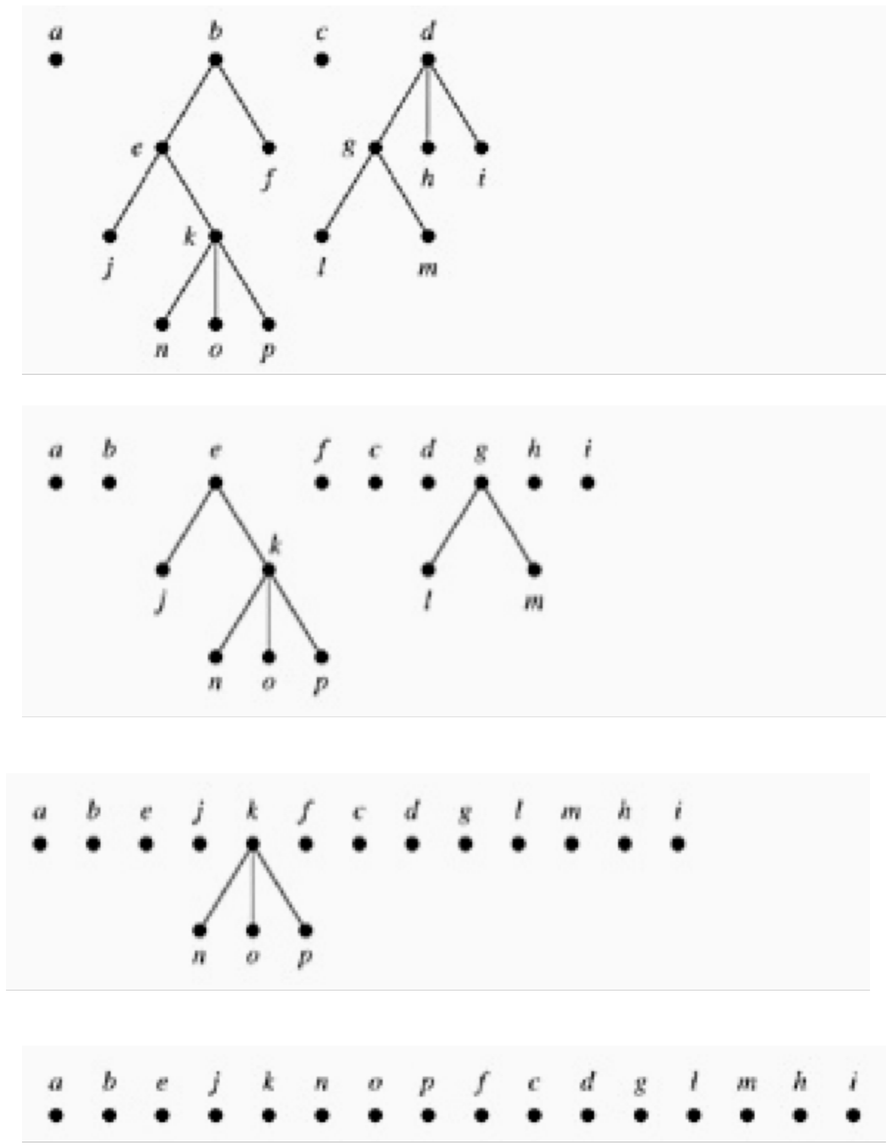
Preorder traversal (Preorder)

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



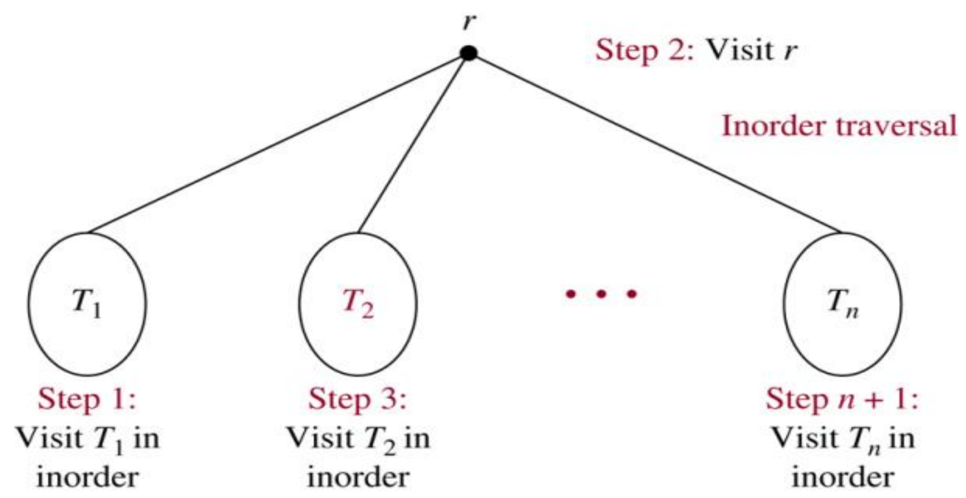
Example



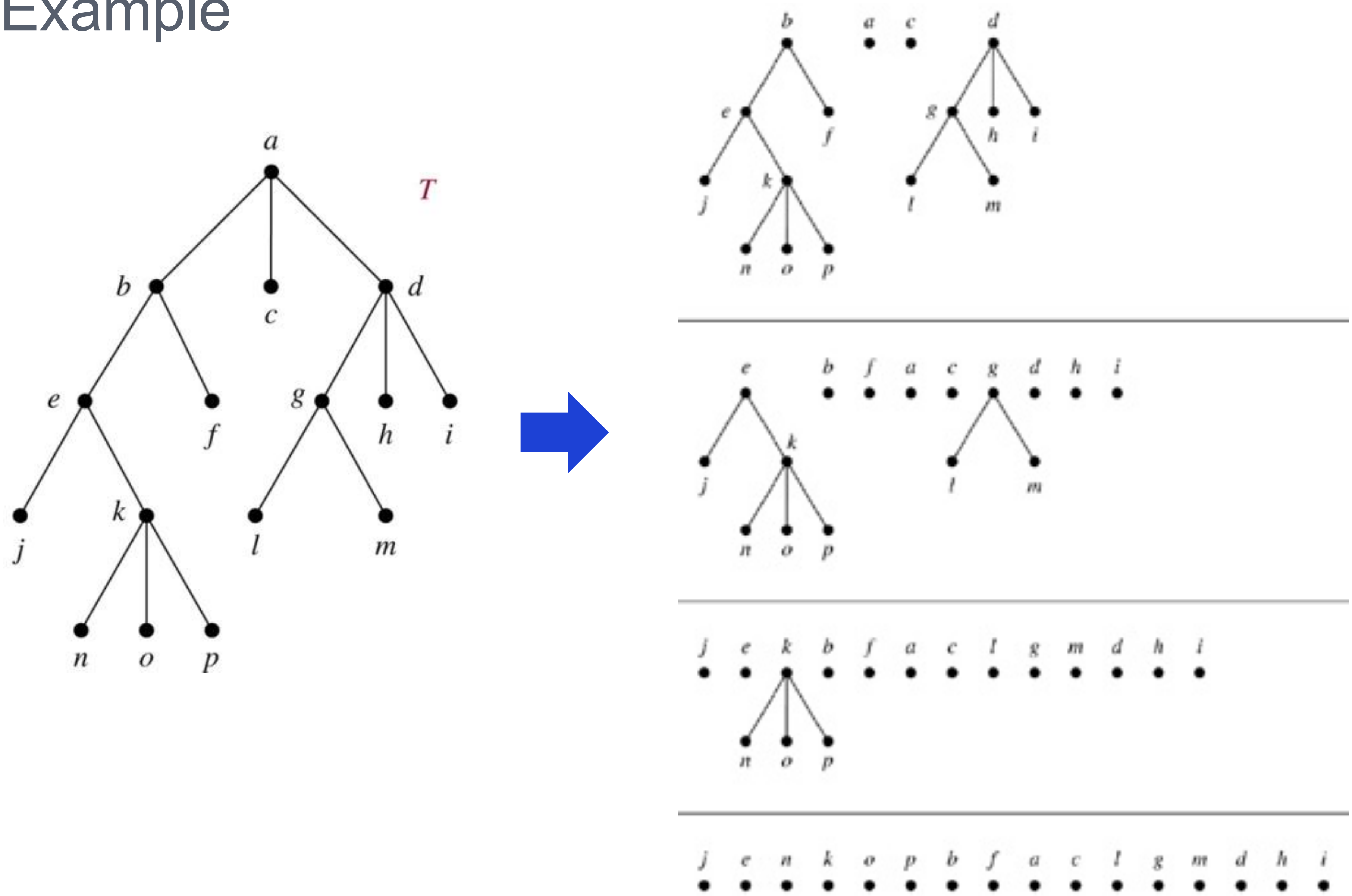


Inorder traversal (Ineorder)

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

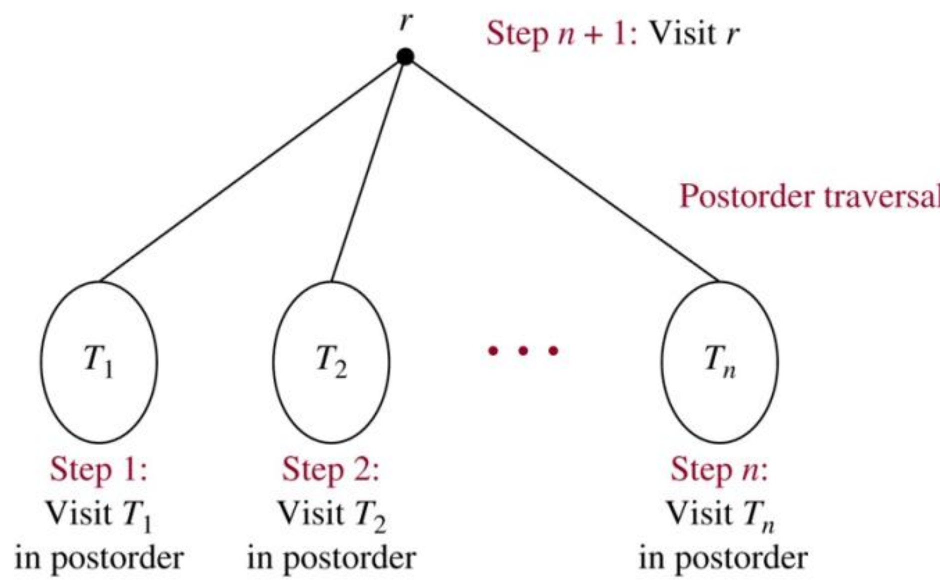


Example

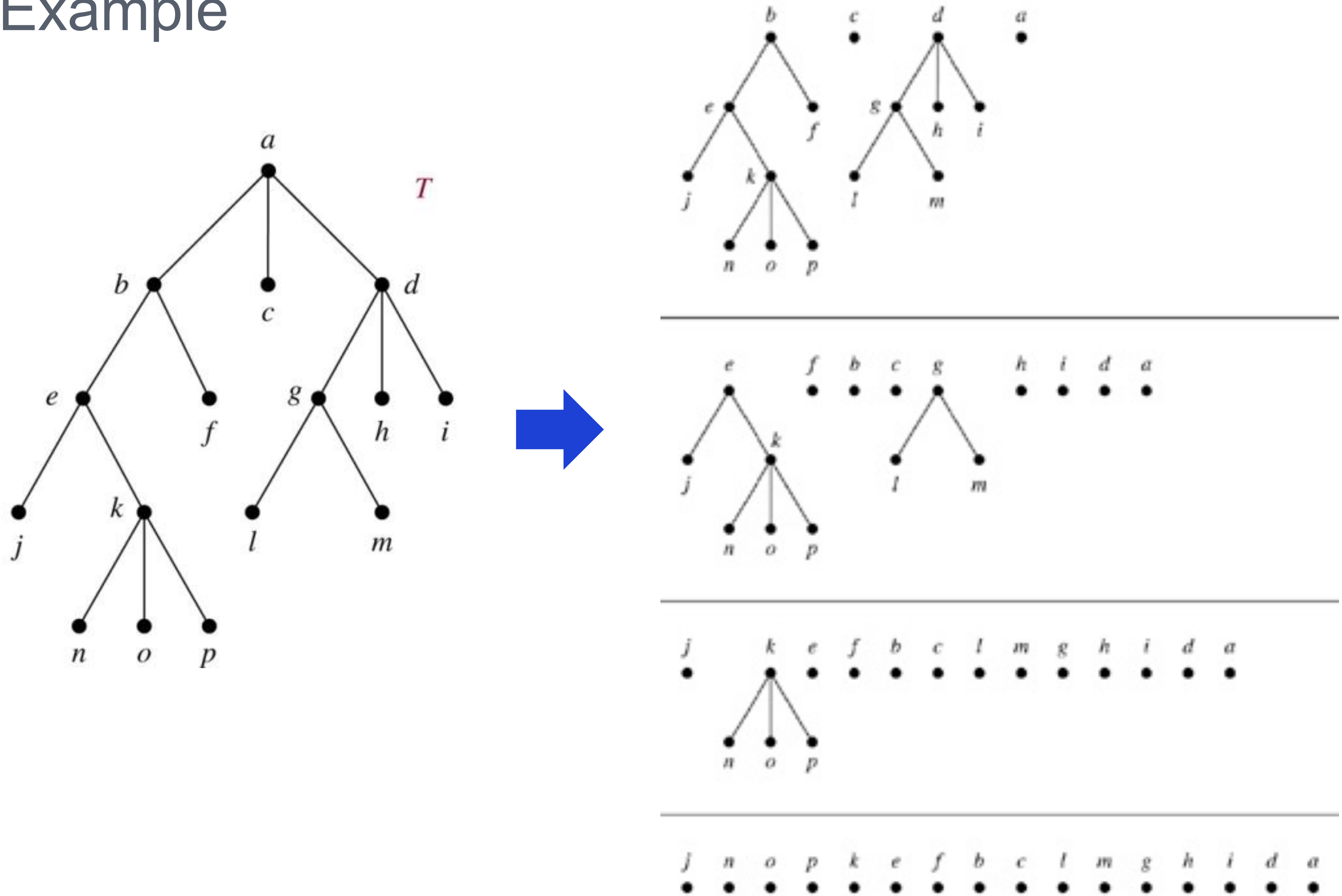

T
a
b c d
e f g h i
j k l m
n o p
b a c d
e f g h i
j k l m
n o p
e b f a c g d h i
j k l m
n o p
j e k b f a c l g m d h i
n o p
j e n k o p b f a c l g m d h i

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



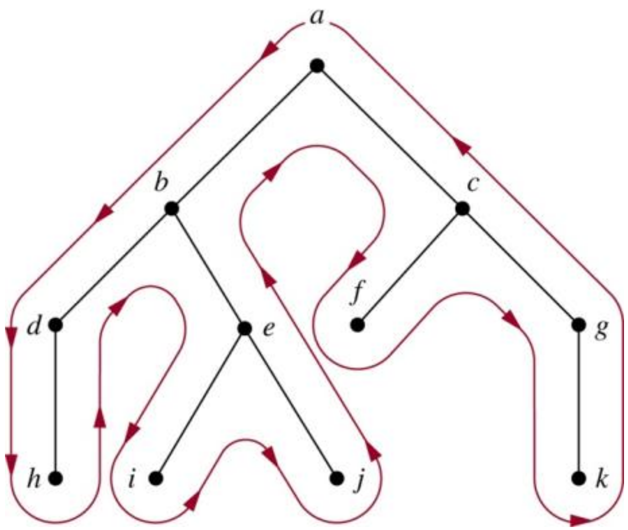
Example



a
T
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
b c d a
e f g h i
j k l m
n o p
e f b c g h i d a
j k l m
n o p
j k e f b c l m g h i d a
n o p
j n o p k e f b c l m g h i d a

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

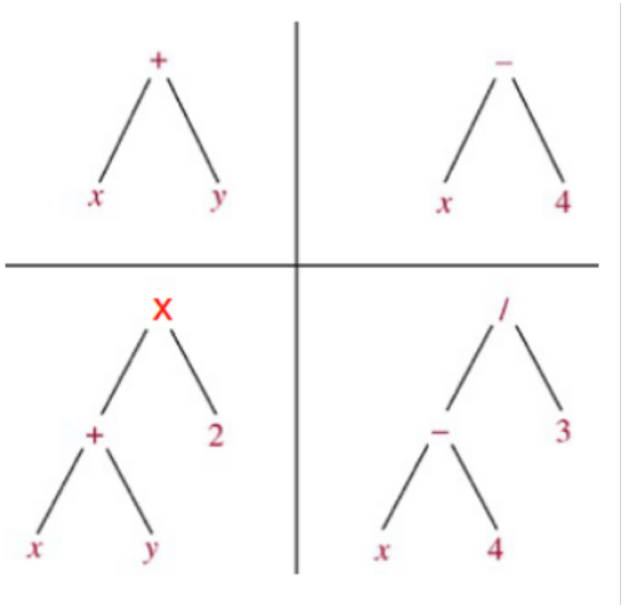
- We can represent complicated expressions, such as compound propositions, combinations of sets, and arithmetic expressions using ordered rooted trees.

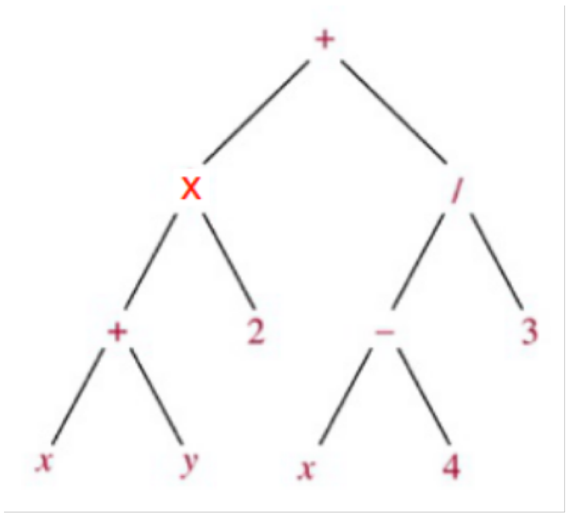
Example Find the ordered rooted tree for $((x+y) \times 2)+((x-4)/3)$.

Solution:

leaf:
variable

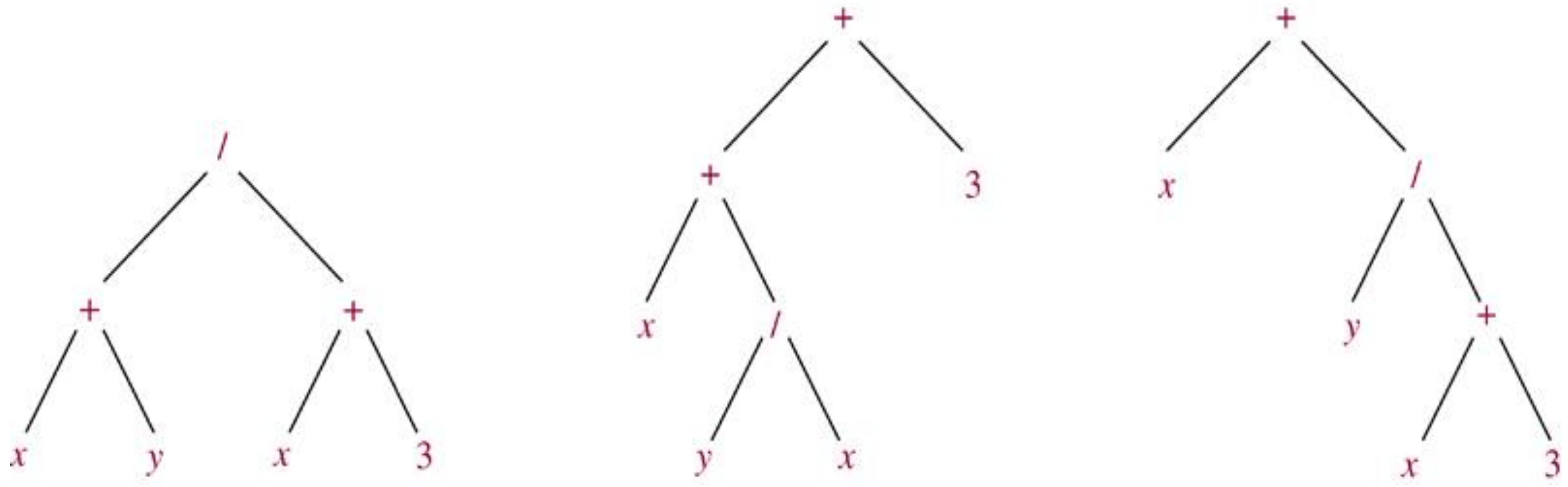
internal vertex:
operation on its left and right subtrees





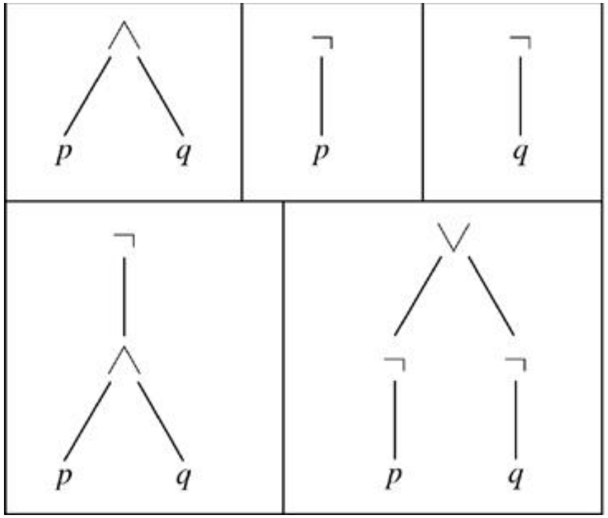
Example

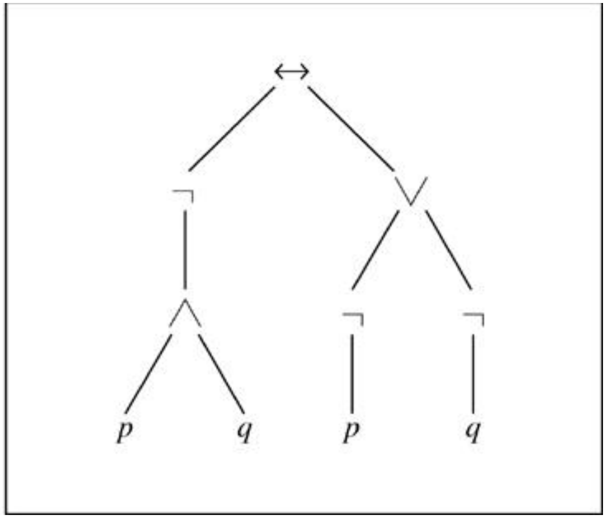
- The following binary trees represent the expressions: (x+y)/(x+3), (x+(y/x))+3, x+(y/(x+3)).
All their inorder traversals lead to x+y/x+3 ⇒ ambiguous ⇒ need parentheses



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.





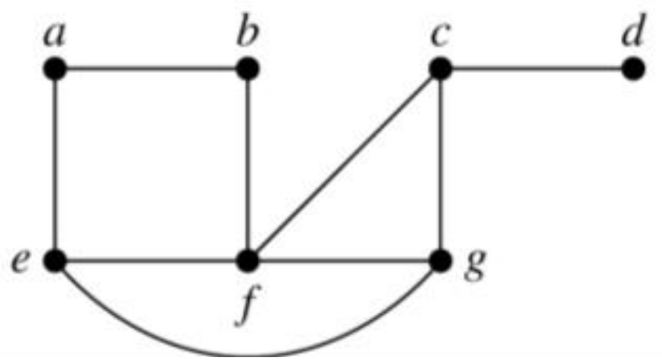
prefix: $\leftrightarrow \neg \wedge p\ q \vee \neg p \neg q$

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

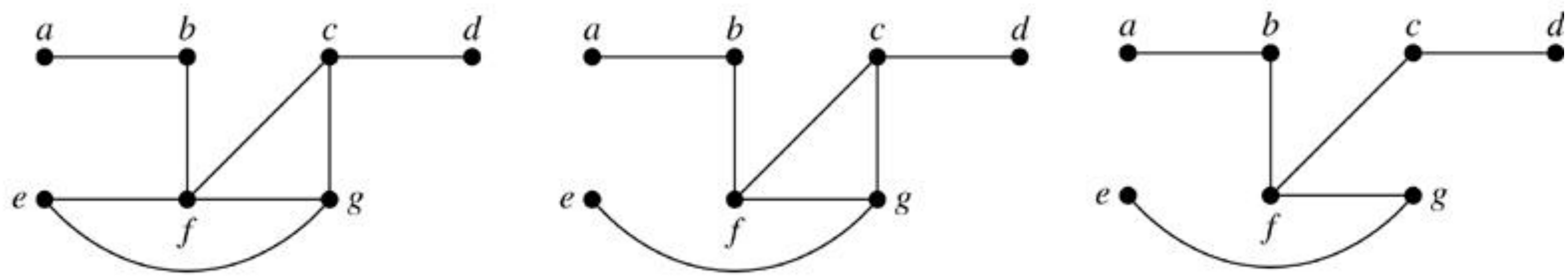
postfix: $p\ q \wedge \neg p \neg q \neg \vee \leftrightarrow$

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.



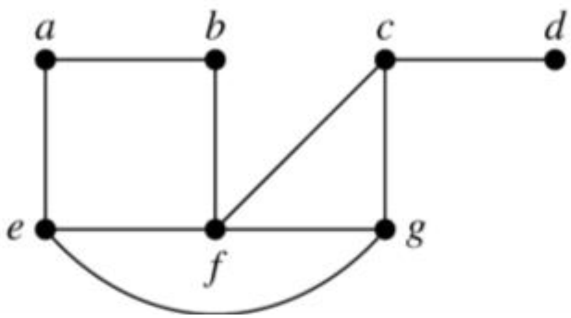
Remove an edge from any circuit.
(repeat until no circuit exists)



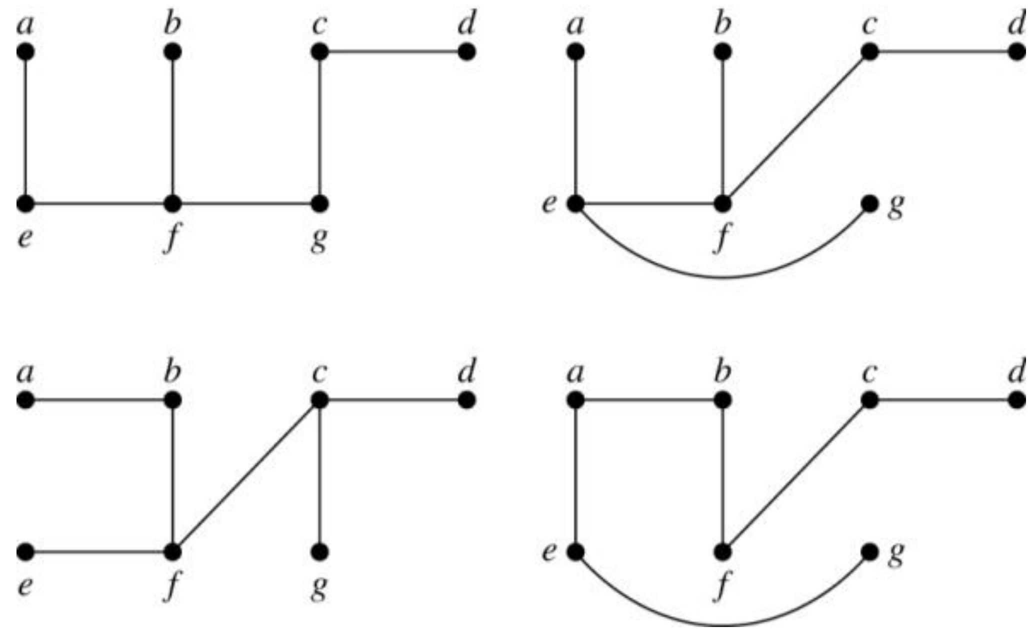
Edge removed: $\{a, e\}$ $\{e, f\}$ $\{c, g\}$

(a) (b) (c)

Spanning Trees



Four spanning trees of G:



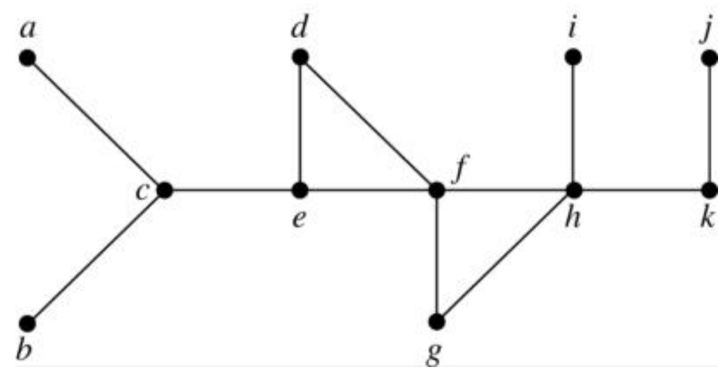
Theorem: A simple graph is connected if and only if it has a spanning tree.

Depth-First Search (DFS)

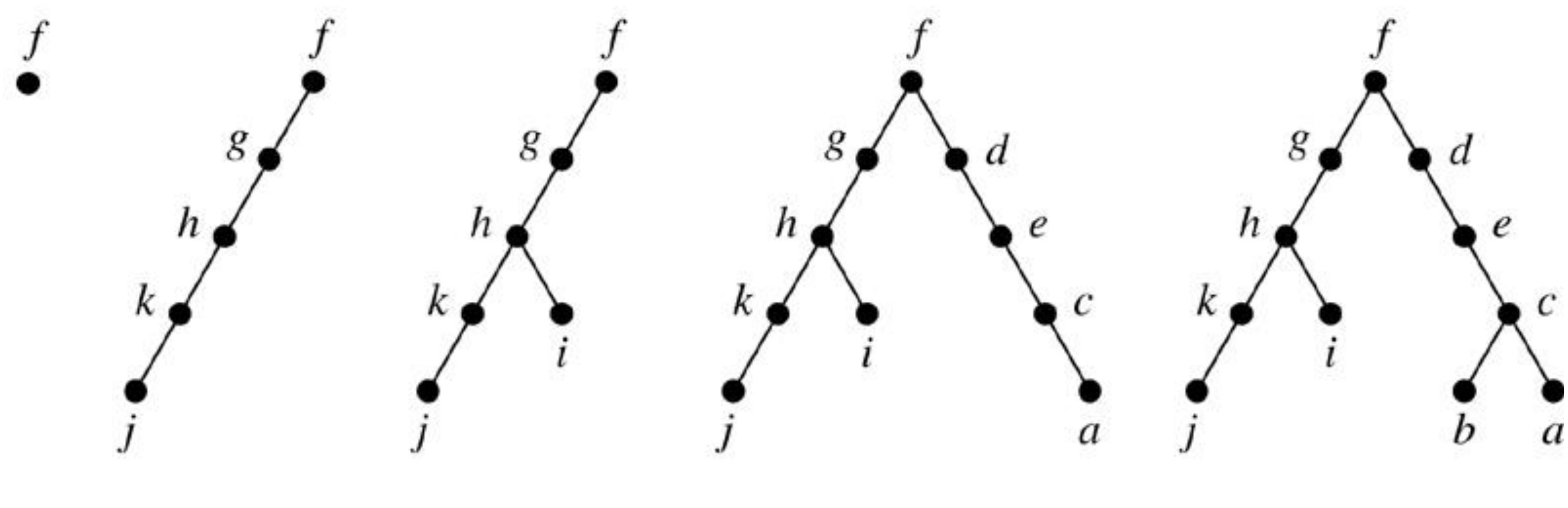
Procedure *DFS*(G: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of the vertex v_1
visit(v_1)
procedure *visit*(v: vertex of G)
for each vertex w adjacent to v and not yet in T
begin
    add vertex w and edge $\{v, w\}$ to T
    visit(w)
end

Example

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



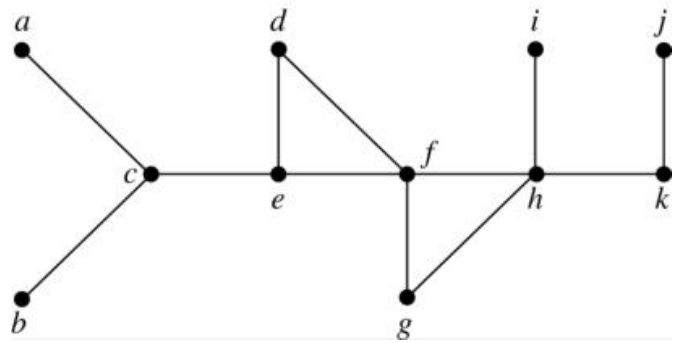
Solution. (arbitrarily start with the vertex *f*)

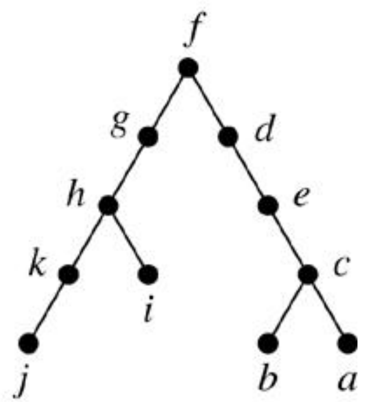


(a) (b) (c) (d) (e)

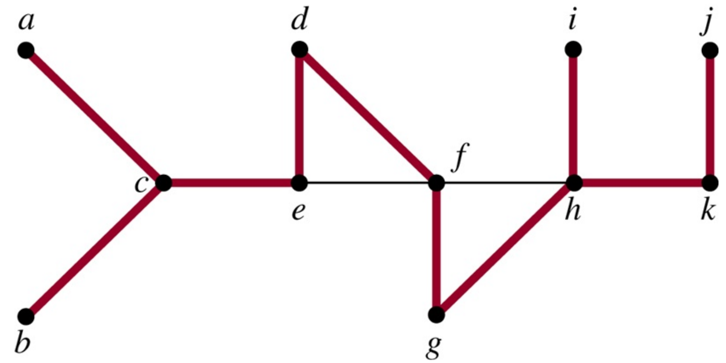
Example

- 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.





⇒



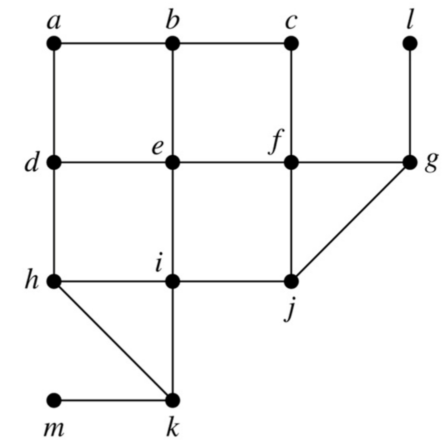
The tree edges (**red**)
and back edges (**black**)

Breadth-First Search (BFS)

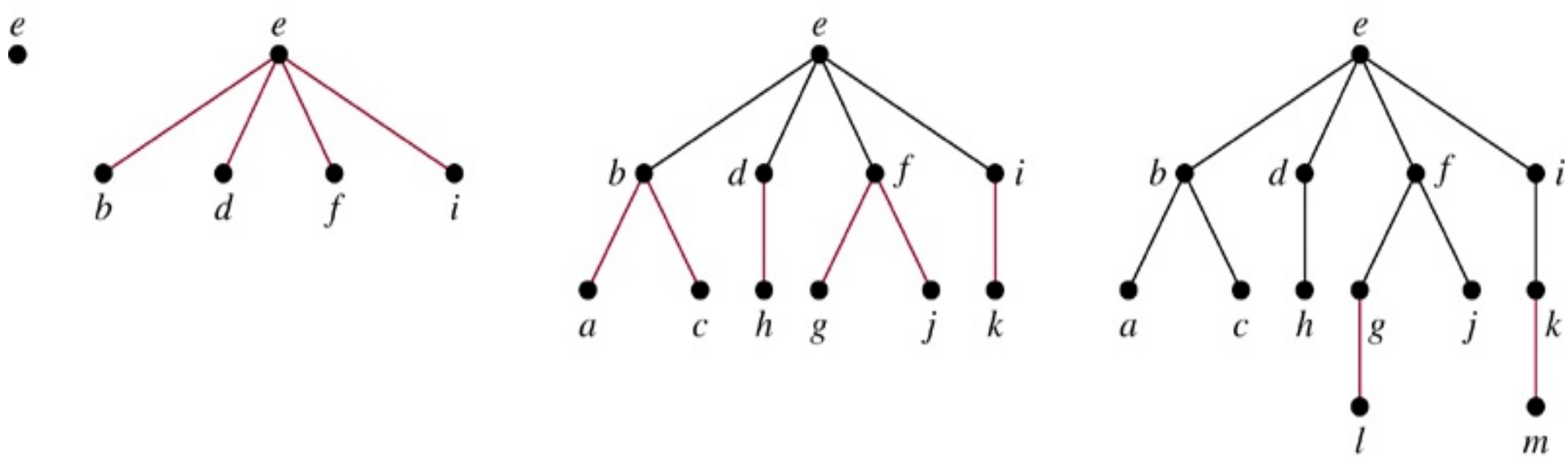
Procedure *BFS*(G: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of vertex v_1
L := empty list
put v_1 in the list L of unprocessed vertices
while L is not empty
begin
    remove the first vertex v from L
    for each neighbor w of v
        if w is not in L and not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

Example

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



Solution: (arbitrarily start with the vertex 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

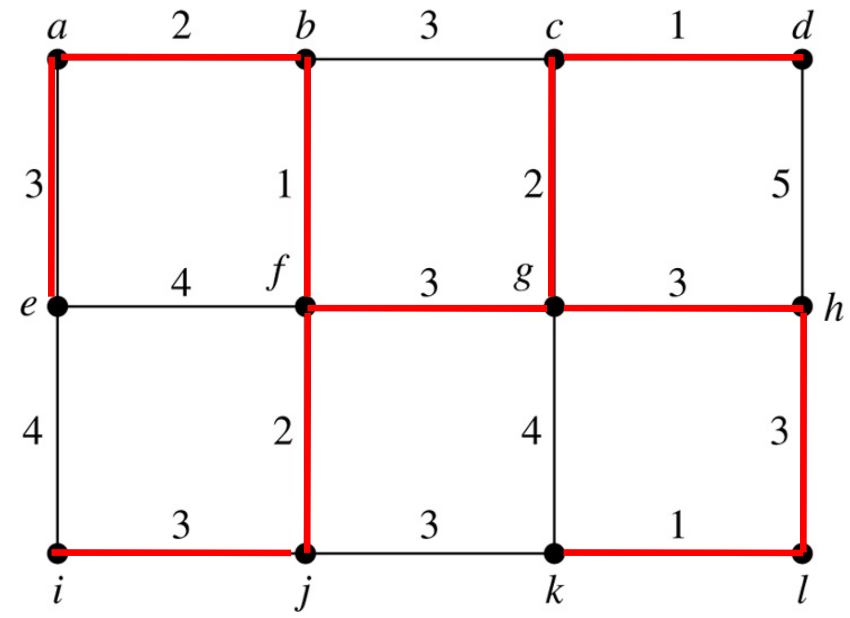
end {T is a minimum spanning tree of G}

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

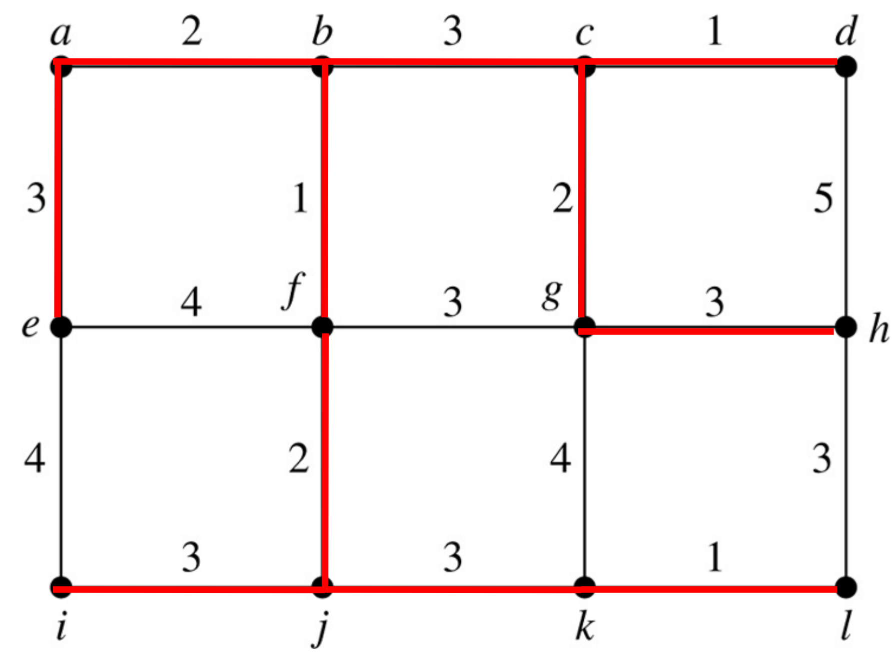
end {T is a minimum spanning tree of G}

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:	24

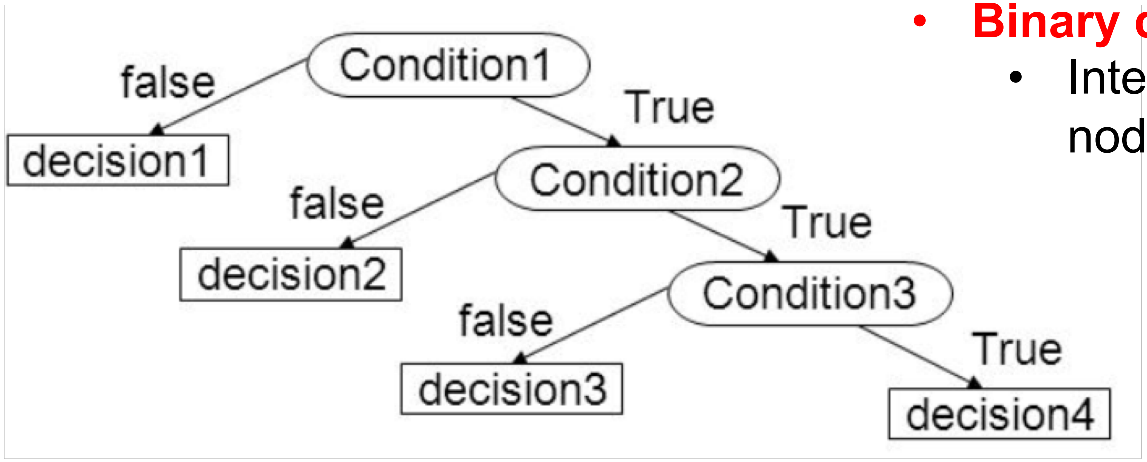


Applications of Trees

Applications of Trees

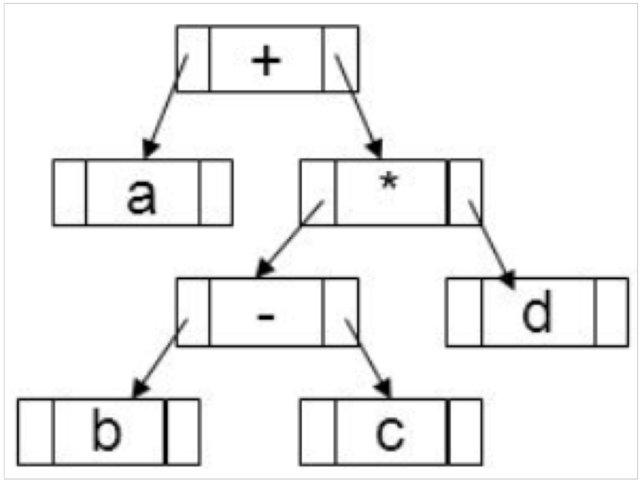
- File Systems
  - Hierarchical files systems include Unix and DOS
  - In DOS, each \ represents an edge (In Unix, it's /)
  - Each directory is a file with a list of all its children
- Store large volumes of data
  - data can be quickly inserted, removed, and found
- Data structure used in a variety of situations
  - implement data vas management systems
  - compliers, expression tree, symbol tree

Applications of Binary Trees

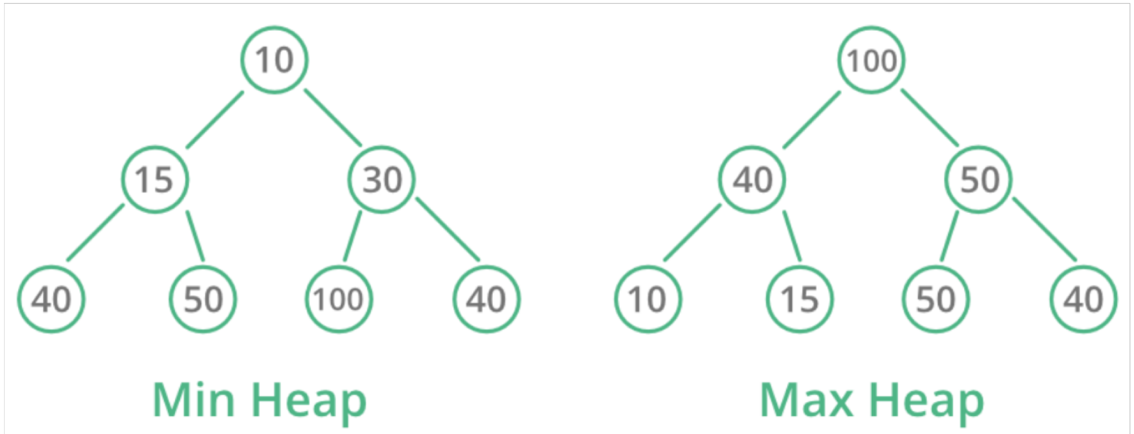


- **Binary decision trees**
  - Internal nodes are conditions. Leaf nodes denote decisions.

Expression Trees



Heap Data Structure

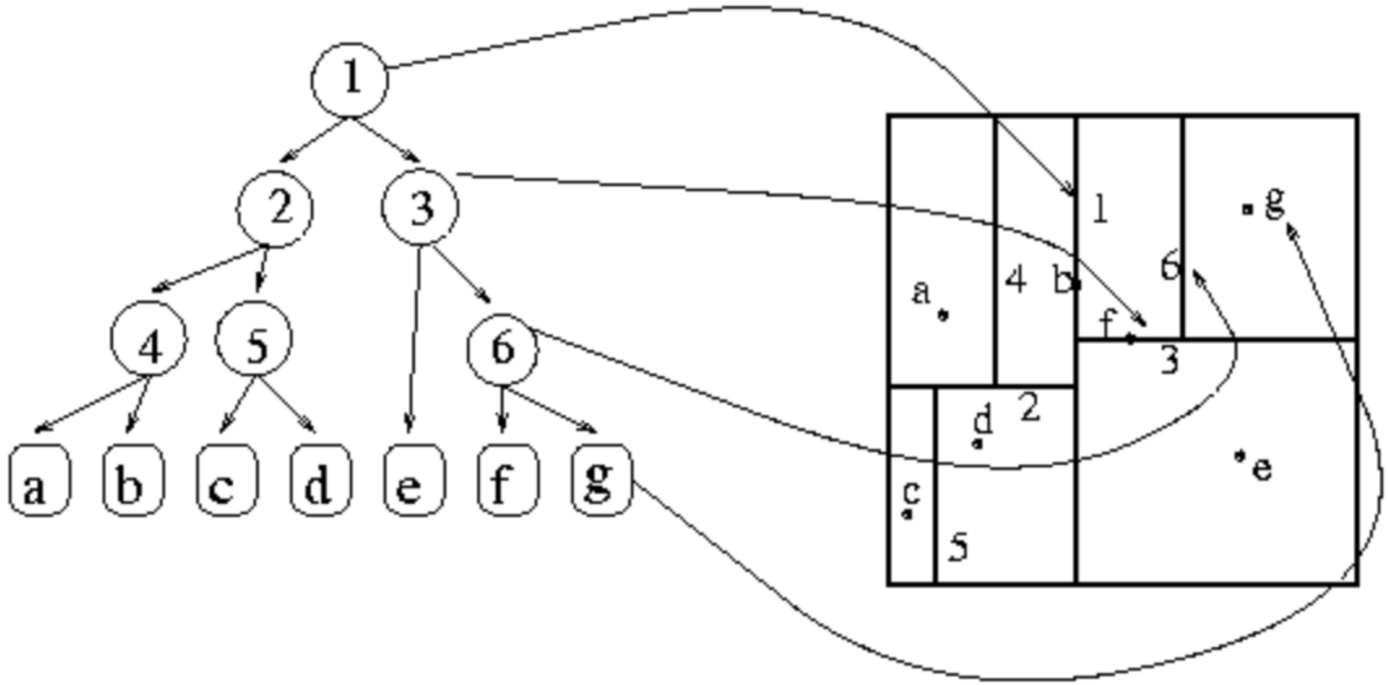


- Binary Heap
- Time Complexity of building a heap
- Applications of Heap Data Structure
- Binomial Heap
- Fibonacci Heap
- Leftist Heap
- K-ary Heap
- Heap Sort
- Iterative Heap Sort
- K'th Largest Element in an array
- K'th Smallest/Largest Element in Unsorted Array | Set 1
- Sort an almost sorted array/
- Tournament Tree (Winner Tree) and Binary Heap
- Check if a given Binary Tree is Heap
- How to check if a given array represents a Binary Heap?
- Connect n ropes with minimum cost
- Design an efficient data structure for given operations
- Merge k sorted arrays | Set 1
- Merge Sort Tree for Range Order Statistics
- Sort numbers stored on different machines
- Smallest Derangement of Sequence
- ……

1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.

K-D Tree

Each node an axis parallel split, with points in leaves.

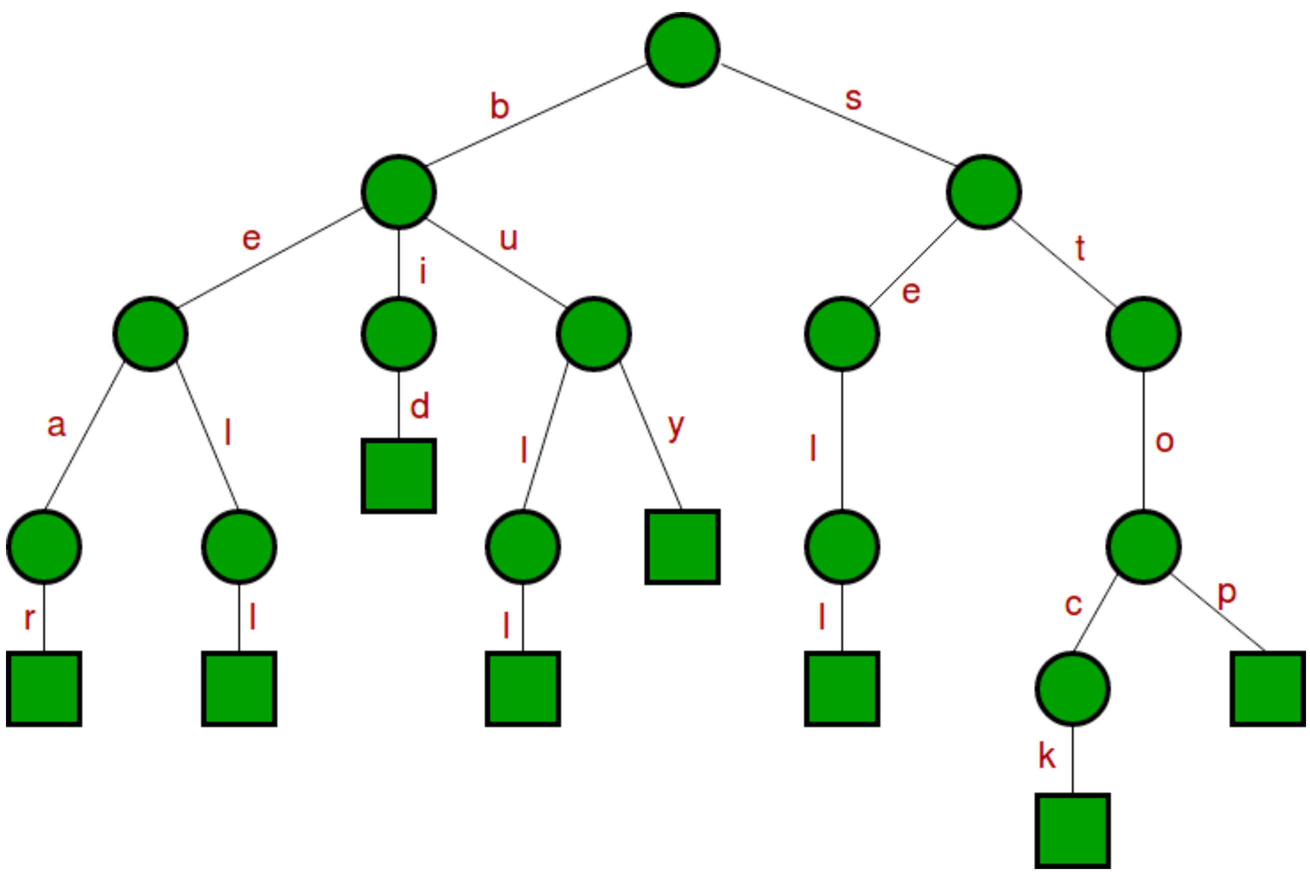


Nice features:

- Search for points in a rectangular window in O(sqrt(n) + k)
- Still O(n) storage and O(n log n) construction time

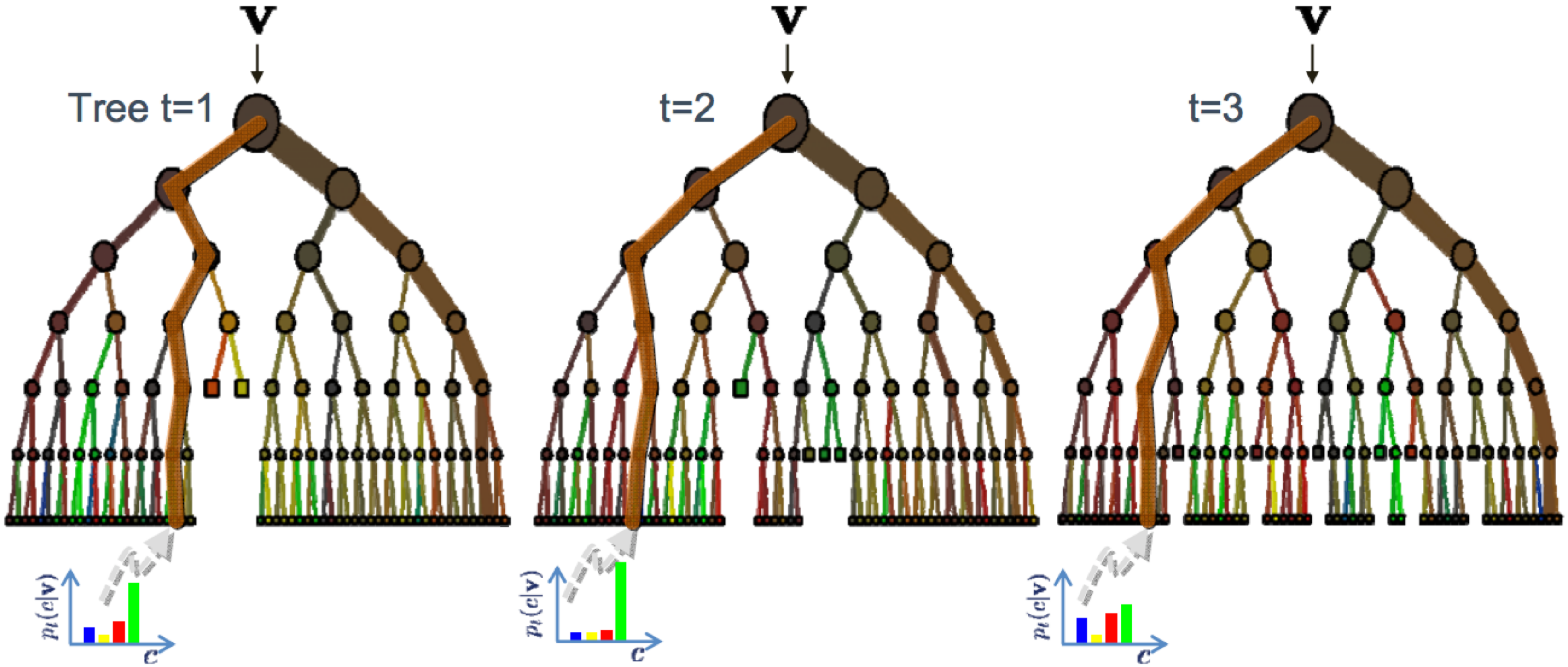
Pattern Searching using Suffix Tree

Given a text txt[0..n-1] and a pattern pat[0..m-1]



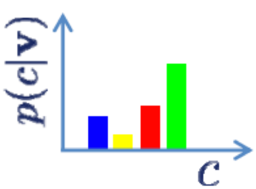
{bear, bell, bid, bull, buy, sell, stock, stop}

Random Tree and Random Forest



The ensemble model

Forest output probability $p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$



About Final Project

About Final Project

- **Research investigations and determine a topic** Identify publications and research that uses some discrete math terminologies.
- Make interesting analogies and application of investigation to your current research.
- e.g., Graph theory, numerical computation, proofs by induction.

About Final Project

- Do some research on your project related to this course.
- Implement the solutions, produce some experimental results, and show the demos.
- Present your work in class during the last week of class (May 2). Presentation should be about 10 minutes/person. Hence a group of 2-3 people should give a 8-10 minute presentation.
- Submit complete project report in IEEE conference proceeding format. (4-8 pages)

Guideline for the final project presentation

- Briefly review the importance, the problem you solved and the objective in your project - 1 or 3 slides (can used some of your proposal slides)
- The details of your solutions you used to solve the project - at least 2 slides.
- Experiment part - at least 3 slides.
- Share the mistakes you encountered and the lessons you learn during completing the final project to others - 1 slide.
- List the references. - 1 slide
- Demo show. 1-3 slides

8-10 min presentation, including Q&A. I would like to recommend you to use informative figures as possible as you can to share what you are going to do with the other classmates

Import dates

- April 26: Determine the project topic and team member in Google Sheet.
- May 1: Draft slides submission
- May 2: Final project presentation
- May 10: Project report (4-8 pages) and code submission

Next class

- Final Project Presentation