Discrete Mathematics: Lecture 29

Tree, Tree Traversals, Spanning Trees, DFS, BFS

Xuming He
Associate Professor

School of Information Science and Technology
ShanghaiTech University

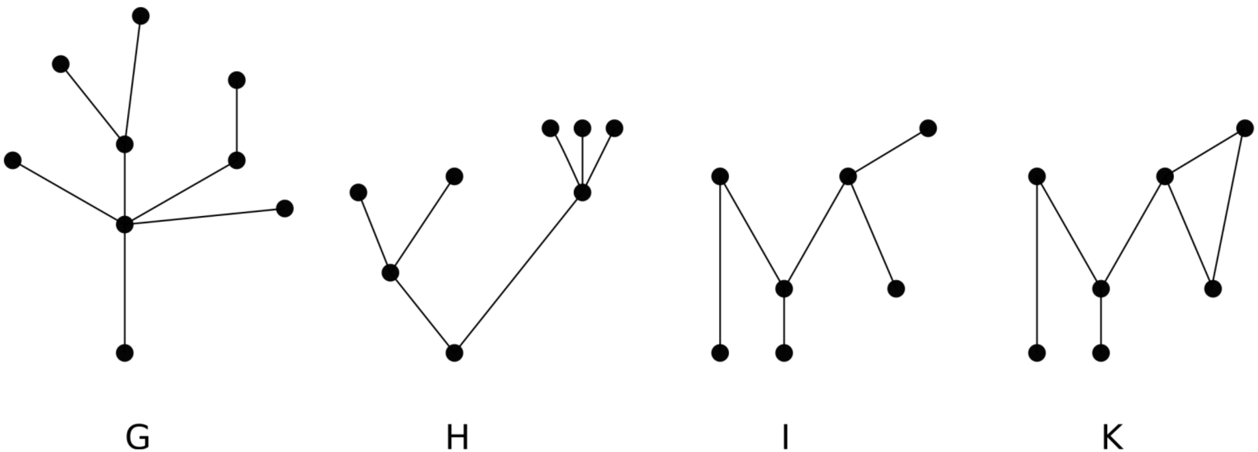
Spring Semester, 2022

Notes by Prof. Liangfeng Zhang

Tree

Definition

- A **tree** is a connected undirected graph with no simple circuits.
- A **forest** is an graph such that each of its connected components is a tree.



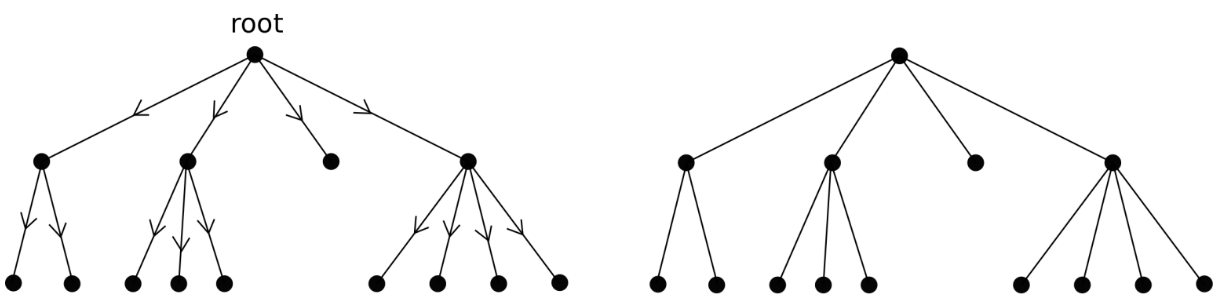
G, H, I are trees, but K is not a tree.

Rooted 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.

Remarks: • A rooted tree is a directed graph.
• We usually draw a rooted tree with its root at the top of the graph.
• We usually omit the arrows on the edges to indicate the direction because it is uniquely determined by the choice of the root.
• Any non rooted tree can be changed to a rooted tree by choosing a vertex for the root.



Properties of Tree

Tree = connected with no simple circuit (definition)

(1) connected
(2) no simple circuit
(3) $(n-1)$ edges (n=nb of vertices)

Previous theorem: $(1) + (2) \Rightarrow (3)$

We also have: $(1) + (3) \Rightarrow (2)$
$(2) + (3) \Rightarrow (1)$

Example: For what value of m, n the complete bipartite graph $K_{m,n}$ is a tree?

$K_{m,n}$ is connected, has $m+n$ vertices and $m \times n$ edges.
It is a tree if:

$$m \times n = m + n - 1 \Longleftrightarrow (n-1)m = n - 1$$

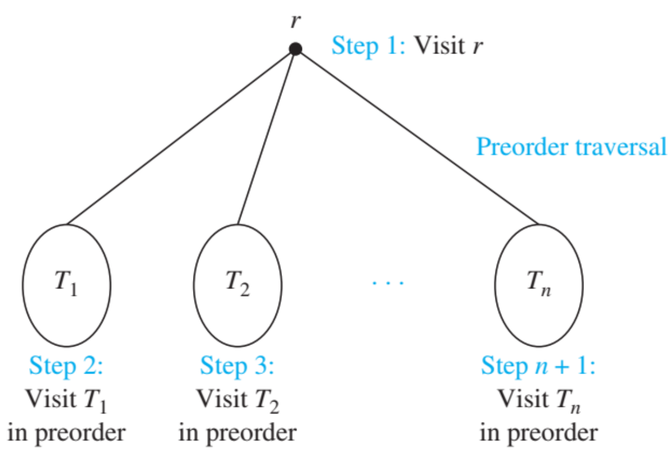
If $n \neq 1$: $m = 1$
If $n = 1$: $m \in \mathbb{N}^*$

Tree Traversals

Preorder traversal algorithm

Recursive definition: Let T be a rooted tree with root r

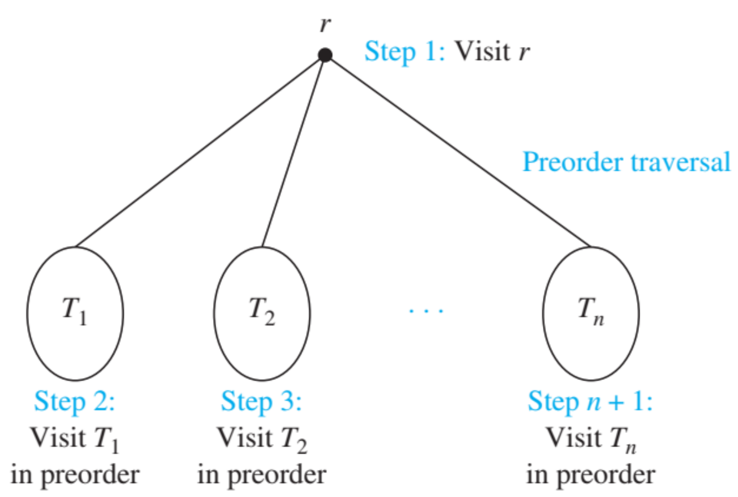
- if T consists only on r: r is the preorder traversal of T.
- otherwise, denote by $T_1, \ldots, T_n$ the subtrees rooted at the children of r, from left to right.
  The preorder traversal of T begins by visiting r, then traverses T_1 in preorder, then T_2 in preorder,..., and finally T_n in preorder.



Tree Traversals

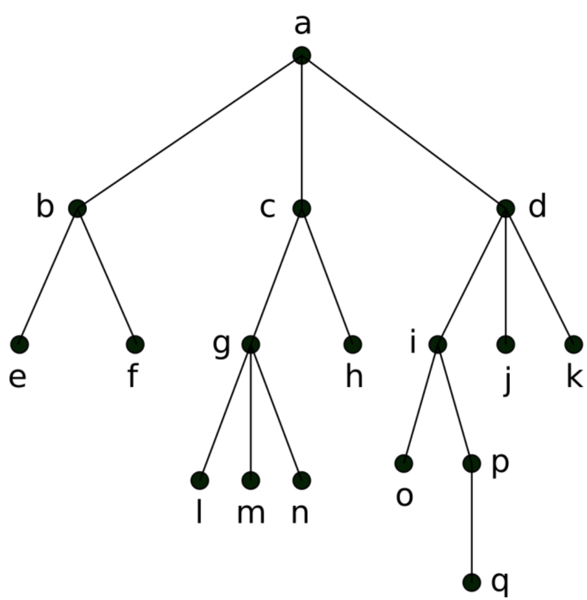
Recursive algorithm:

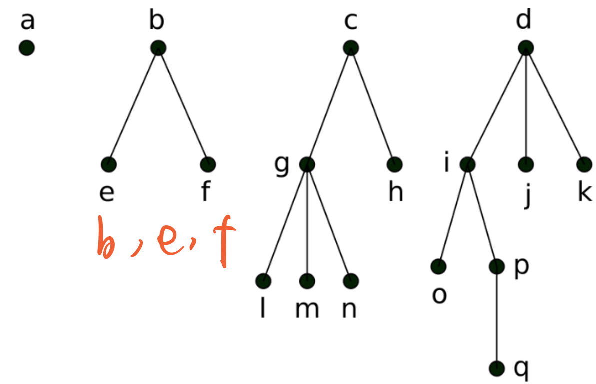
preorder(T: ordered rooted tree)
r:=root of T
list r (add r in the preorder list of the vertices of T)
for each child c of r from left to right
    $T(c)$:= subtree of T with c as its root
    preorder(T(c))

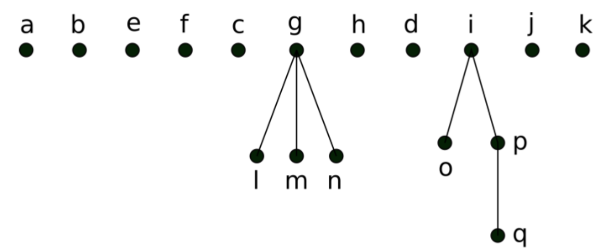


Tree Traversals

Preorder traversal algorithm







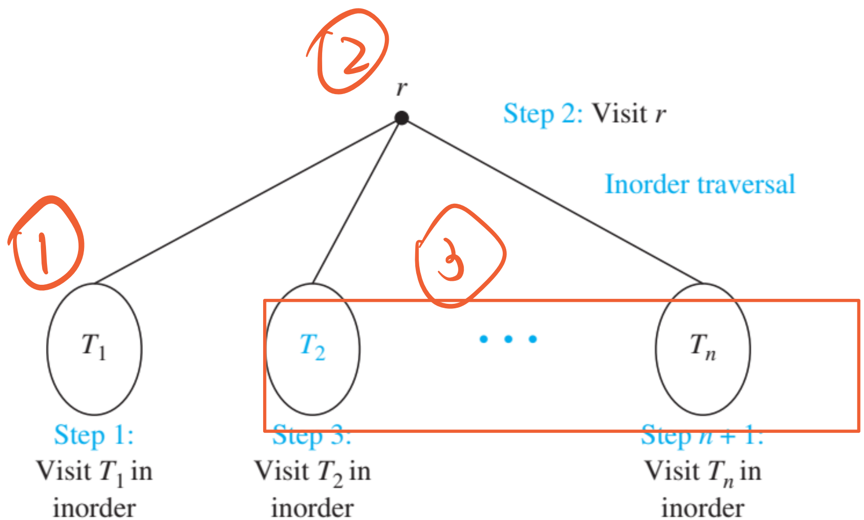
In order

Tree Traversals

Inorder traversal algorithm

Recursive definition: Let T be a rooted tree with root r

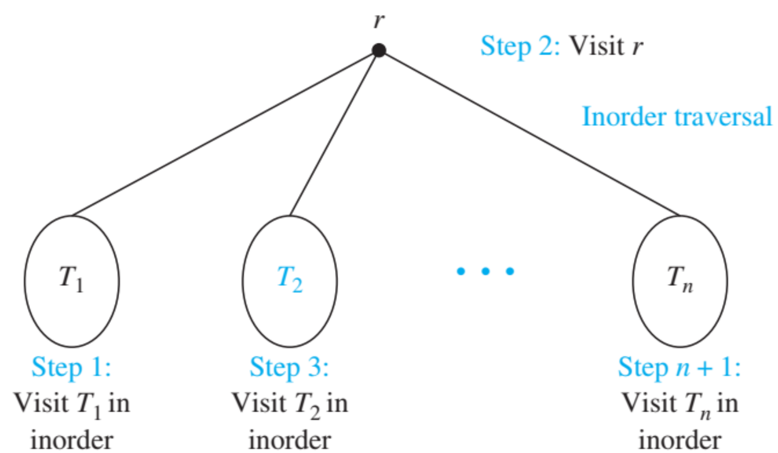
- if T consists only on r: r is the inorder traversal of T.
- otherwise, denote by $T_1, \ldots, T_n$ the subtrees rooted at the children of r, from left to right.
  The inorder traversal of T begins by traversing T_1 in inorder, then visiting r, then traversing T_2 in inorder, then T_3 in inorder,..., and finally T_n in inorder.



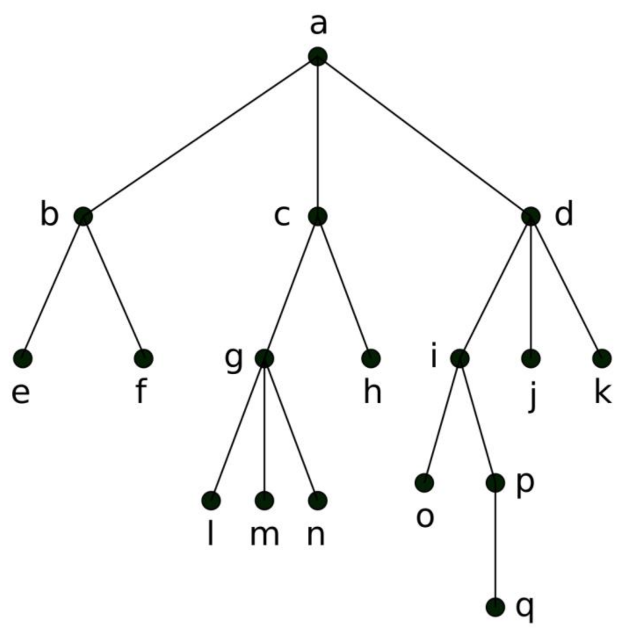
Tree Traversals

Recursive algorithm:

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



Tree Traversals



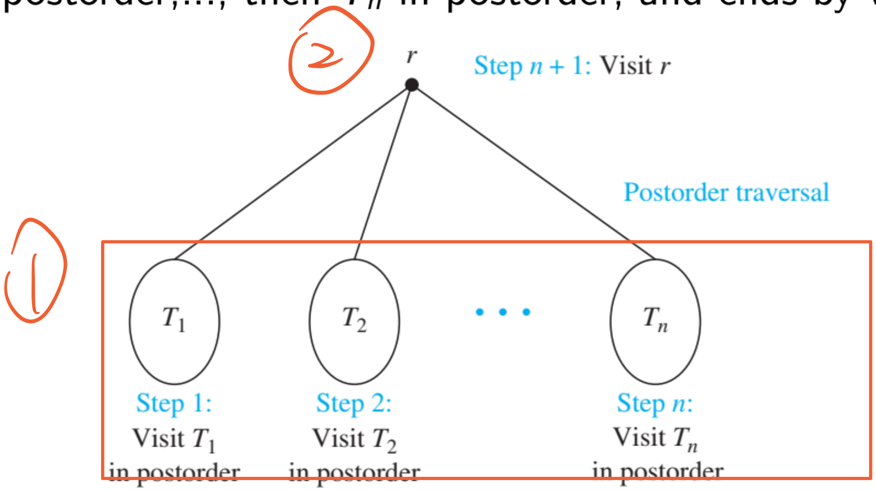
Inorder traversal: e, b, f, a, l, g, m, n, c, h, o, i, q, p, d, j, k

Tree Traversals

Postorder traversal algorithm

Recursive definition: Let T be a rooted tree with root r

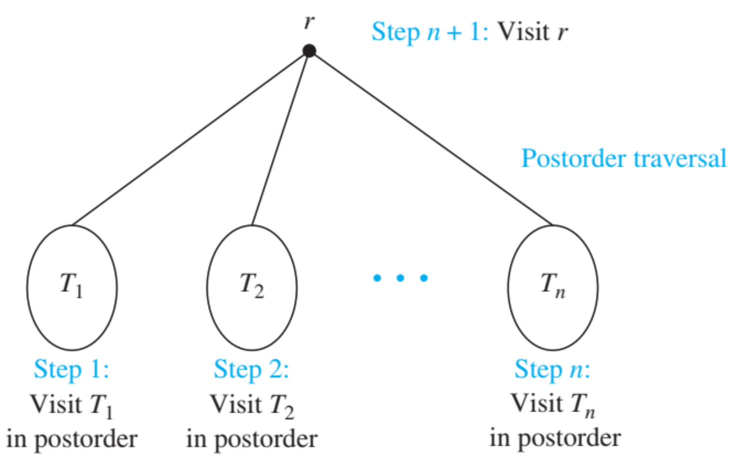
- if T consists only on r: r is the postorder traversal of T.
- otherwise, denote by $T_1, \ldots, T_n$ the subtrees rooted at the children of r, from left to right.
  The postorder traversal of T begins by traversing T_1 in postorder, then T_2 in postorder,..., then T_n in postorder, and ends by visiting the root r.



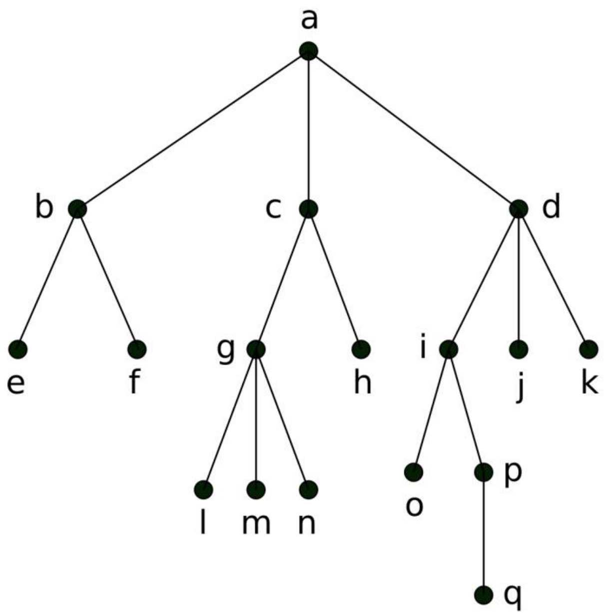
Tree Traversals

Recursive algorithm:

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



Tree Traversals



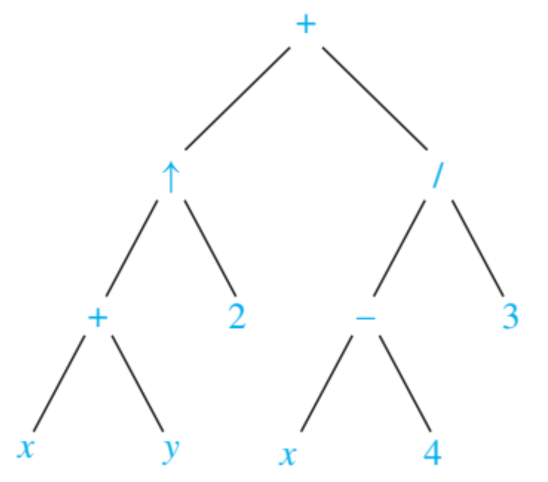
Postorder traversal: e, f, b, l, m, n, g, h, c, o, q, p, i, j, k, d, a

Infix, Prefix, Postfix Notation

Goal: Using ordered rooted trees to represent arithmetic expressions or compound propositions.

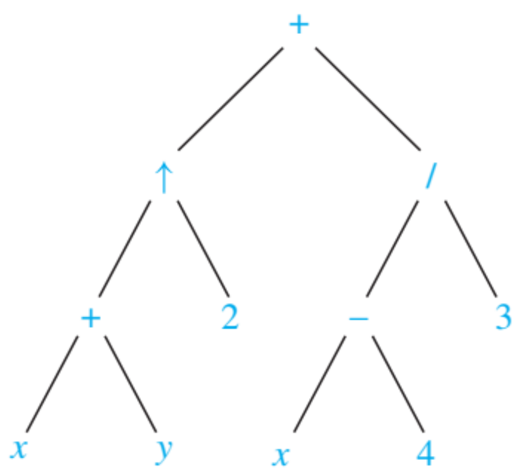
- leaves: numbers or variables,
- internal vertices: operations, where each operation operates on its left and right subtrees in that order (or its only subtree if it is a unary operation).

$$((x + y) \uparrow 2) + ((x - 4)/3)$$



Infix, Prefix, Postfix Notation

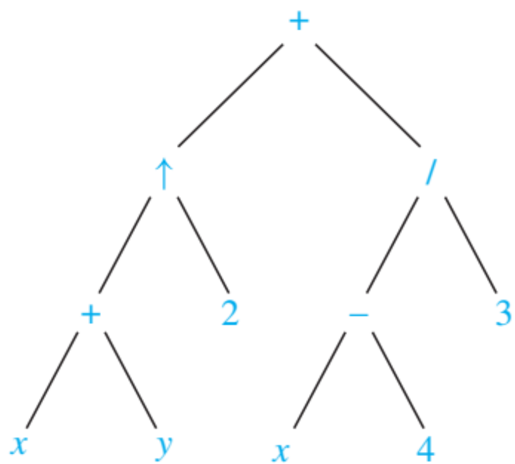
⇒ An inorder traversal of a binary tree representing an expression produces the original expression with the elements and operations in the same order as they originally appear, except for unary operation.
But: inorder traversals give ambiguous expressions ⇒ need to include parentheses ⇒ **infix form** (fully parenthesized)



$$((x + y) \uparrow 2) + ((x - 4)/3)$$

Infix, Prefix, Postfix Notation

The **prefix form (Polish notation)** of an expression is obtained by traversing its corresponding rooted tree in preorder.
An expression in prefix form (where each operation has a specified number of operands) is unambiguous.

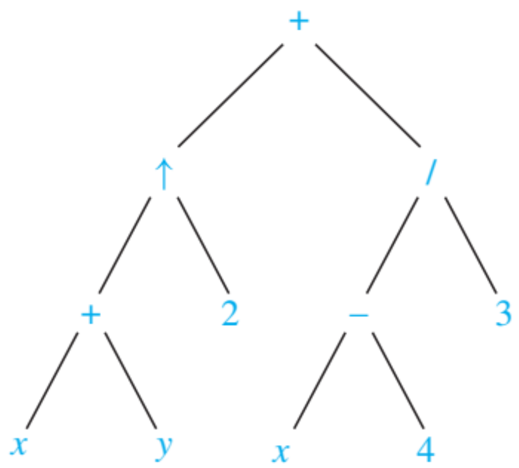


$+ \uparrow + x\ y\ 2\ / - x\ 4\ 3$

- Evaluate an expression in prefix form by working from right to left.
- When we encounter an operator, we perform the corresponding operation with the two operands immediately to the right of this operand.

Infix, Prefix, Postfix Notation

The **postfix form (reverse Polish notation)** of an expression is obtained by traversing its corresponding rooted tree in postorder. An expression in postfix form (where each operation has a specified number of operands) is unambiguous.



$$x\ y + 2 \uparrow x\ 4 - 3 / +$$

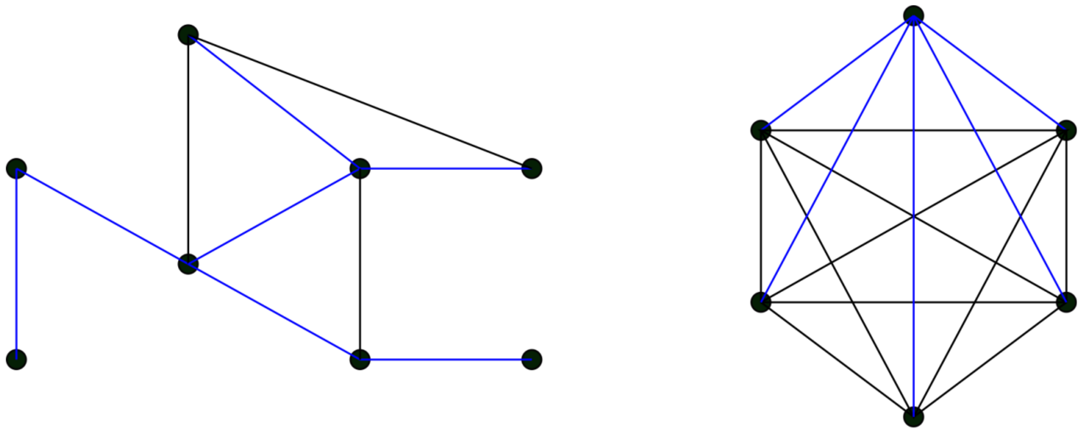
- Work from left to right, carrying out operations whenever an operator follows two operands.
- After an operation is carried out, the result of this operation becomes a new operand.

Spanning Trees

Definition

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.

Example:

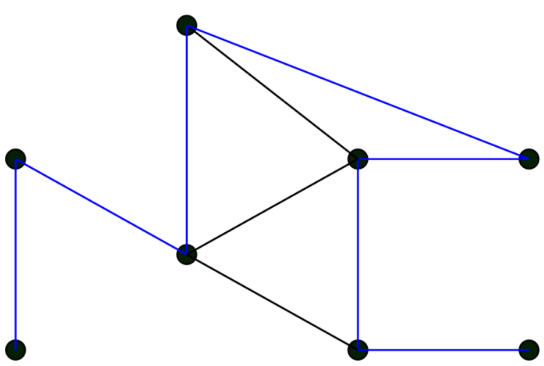


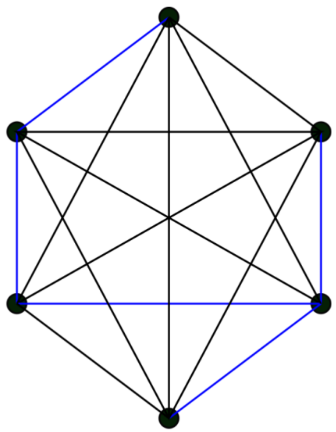
Spanning Trees

Definition

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.

Example:





Spanning Trees

Theorem

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

Proof:

"$\Leftarrow$" Assume G is a simple graph admitting a spanning tree T:

- T subgraph of G containing all vertices of G,
- by definition of tree, their is a path between any two vertices of T

So their is a path between any two vertices of G.

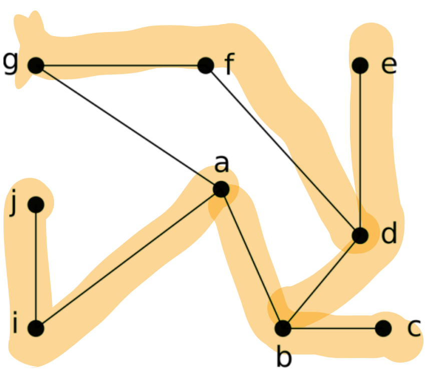
"$\Rightarrow$" Assume G is a simple connected graph.
If it is not a tree, it contains a circuit. Denote G' the subgraph of G obtained by removing one edge of the circuit with endpoints u and v.
There is still a path from u to v $\Rightarrow$ G' is connected.
If G' is not a tree, it contains a circuit, and again take a subgraph removing one edge of the circuit.
Repeat this process until there is no more circuit.
The graph obtained is connected and has no circuit, it is a spanning tree.

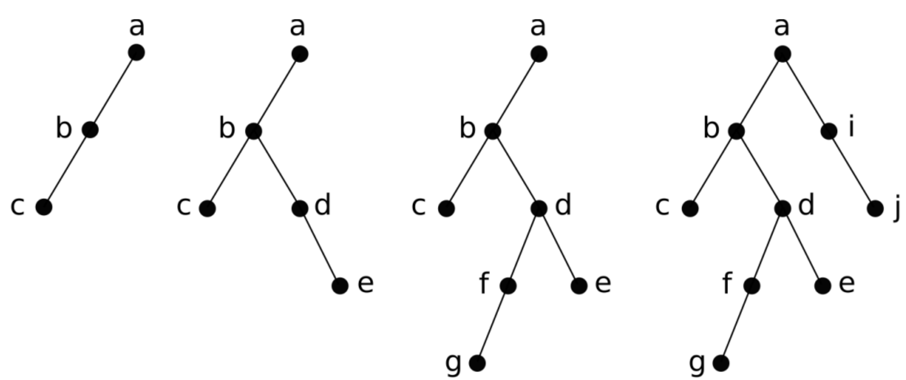
Depth-first Search

Recursive algorithm

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)

visit(v: vertex of G)
for each vertex w adjacent to v and not yet in T
    add vertex w and edge (v, w) to T
    visit(w)





Breadth-first Search

Algorithm

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
    remove the first vertex v from L
    for each neighbour w of v
        if w is not in L and not in T **then**
            add w to the end of the list L
            add w and the edge (v, w) to T

先访完所有
Neighbour