Tree & Graph

ESC108A Elements of Computer Science and Engineering

B. Tech. 2016

Course Leaders:

Roopa G.

Ami Rai E.

Chaitra S.

Objectives

- At the end of this lecture, student will be able to
  - use the structure and operations of a ***binary tree*** data structure
  - use the structure and operations of a ***binary search tree*** data structure
  - use the structure and operations on ***graphs***

Contents

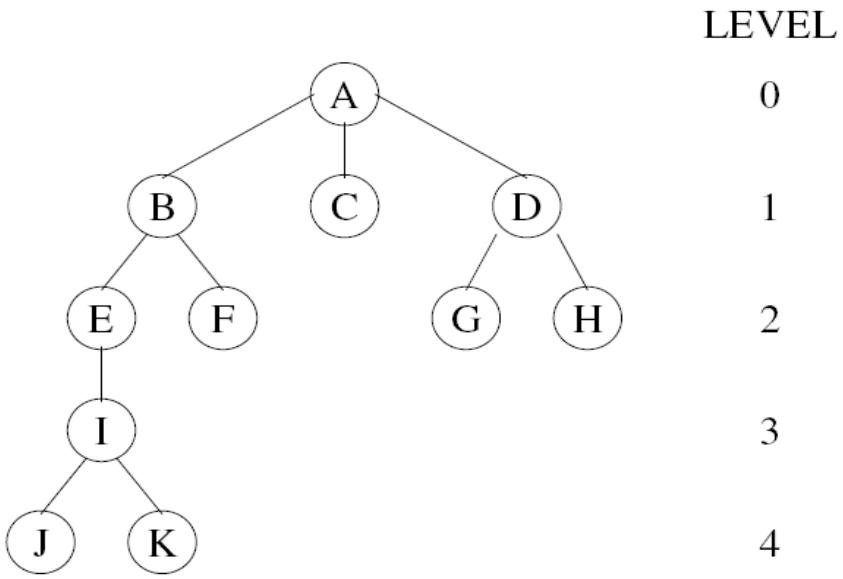
- Tree
- Binary Trees
- Binary Search Trees
- Graph

Trees

- Trees in Computer Science terminology refer to hierarchical structured data representations
- A nonlinear, two-dimensional data structure with special properties
- Trees consist of *nodes* with a parent-child relation

Tree Data Structures

- Tree has *root* at the top and branches grown *down* ending in *leaves*
- Each link in the root node refers to a child
- The left child is the first node in the left subtree
- Right child is the first node in the right subtree

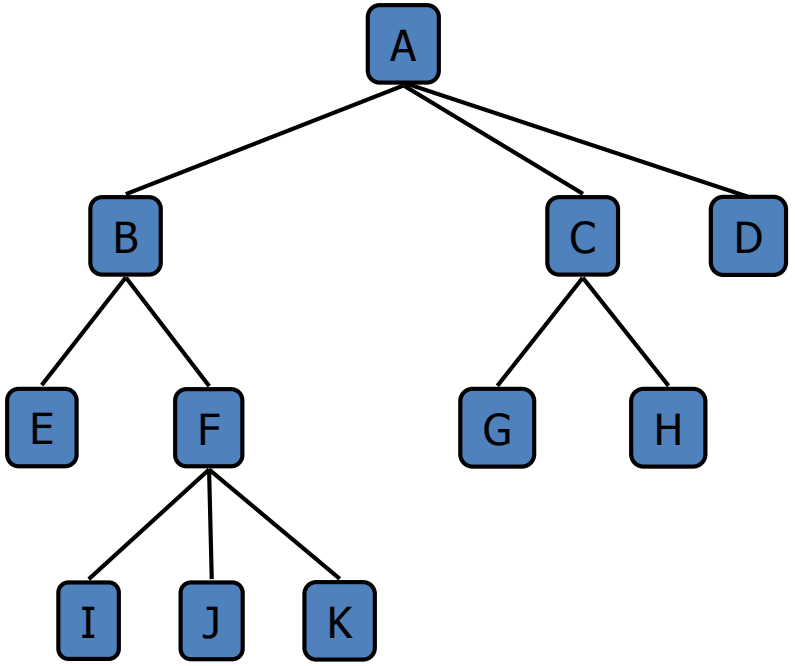


Tree Terminology

- **Root**: A node without a parent
- **Internal Node**: A node with at least one child
- **Leaf (or External Node):** A node without a child
- **Ancestor of a node:** Parent, grand-parent, grand-grand-parent, etc.
- **Descendent of a node:** Child, grand-child, great-grand-child, etc.
- **Depth of a node:** Number of ancestors of the node
- **Height of the Tree:** Maximum depth of any node in the Tree

Example of a Tree

- *root* is node A
- *Internal* (branch) nodes are nodes A, B, C, F
- *External* nodes (*leaves*) are nodes E, I, J, K, G, H, D
- *Depth* of node F is 2
- *Height* of T is 3
- *Ancestors* of node H are C and A
- *Children* of node A are B, C and D
- Nodes B, C and D are *siblings*
- *Descendants* of node B are E, F, I, J and K

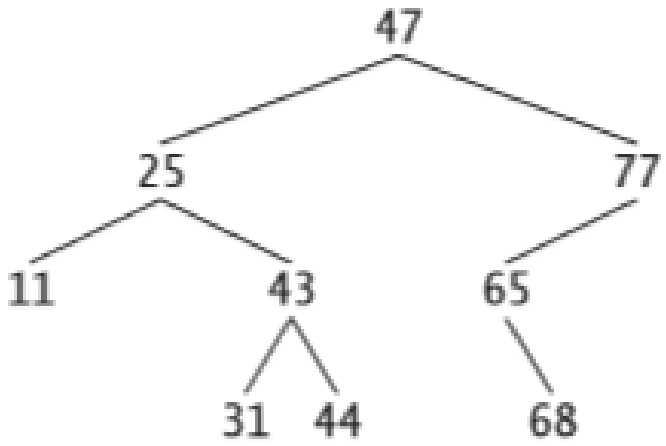


Binary Trees

- Each node has at most two children
  - Left child and right child
- Alternative *recursive definition*
  - A Binary Tree is either a Tree with a single node
  - Or, a Tree whose root has an ordered pair of children, each of which is a Binary Tree
- Applications
  - Binary (*e.g.,* arithmetic) expression evaluation
  - Binary search algorithms
  - Decision logic implementation

Binary Search Tree

- A binary search tree (with no duplicate node values)
  - Values in any left subtree are less than the value in its parent node
  - Values in any right subtree are greater than the value in its parent
- The shape of the binary search tree that corresponds to a set of data can vary, depending on the order in which the values are inserted into the tree
- A binary search tree with 12 values



Tree Traversals

- The fundamental *algorithm* on Trees is Traversal
  - Visit each node of the Tree
  - Optionally perform some operation during each visit

- Types of Tree traversals
  1. Preorder traversal
  2. Postorder traversal
  3. Inorder traversal

- Tree traversal is recursive in nature; the traversal algorithms are expressed recursively

Preorder Traversal

- Each node is visited before all of its descendents

- The steps for a preOrder traversal
  1. Process the value in the node
  2. Traverse the left subtree preorder
  3. Traverse the right subtree preorder

- The value in each node is processed as the node is visited

Postorder Traversal

- Each node is visited *after* its descendents
- The steps for a postOrder traversal
  1. Traverse the left subtree postOrder
  2. Traverse the right subtree postOrder
  3. Process the value in the node

- The value in each node is not printed until the values of its children are printed

Inorder Traversal

- The steps for an inOrder traversal
  1. Traverse the left subtree inOrder
  2. Process the value in the node
  3. Traverse the right subtree inOrder

- The value in a node is not processed until the values in its left subtree are processed
- The inOrder traversal of a binary search tree prints the node values in ascending order

Traversal - Example

- Preorder

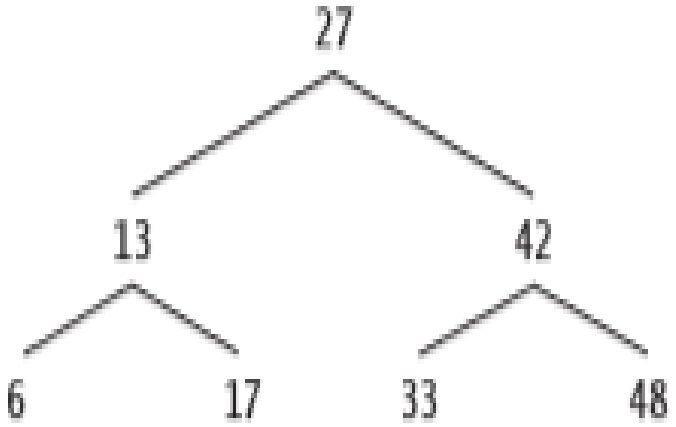
  27 13 6 17 42 33 48

- Postorder

  6 17 13 33 48 42 27

- Inorder

  6 13 17 27 33 42 48



Array Based Tree Implementation

Use a Vector data structure to store the Tree nodes

Rank of a node is defined by

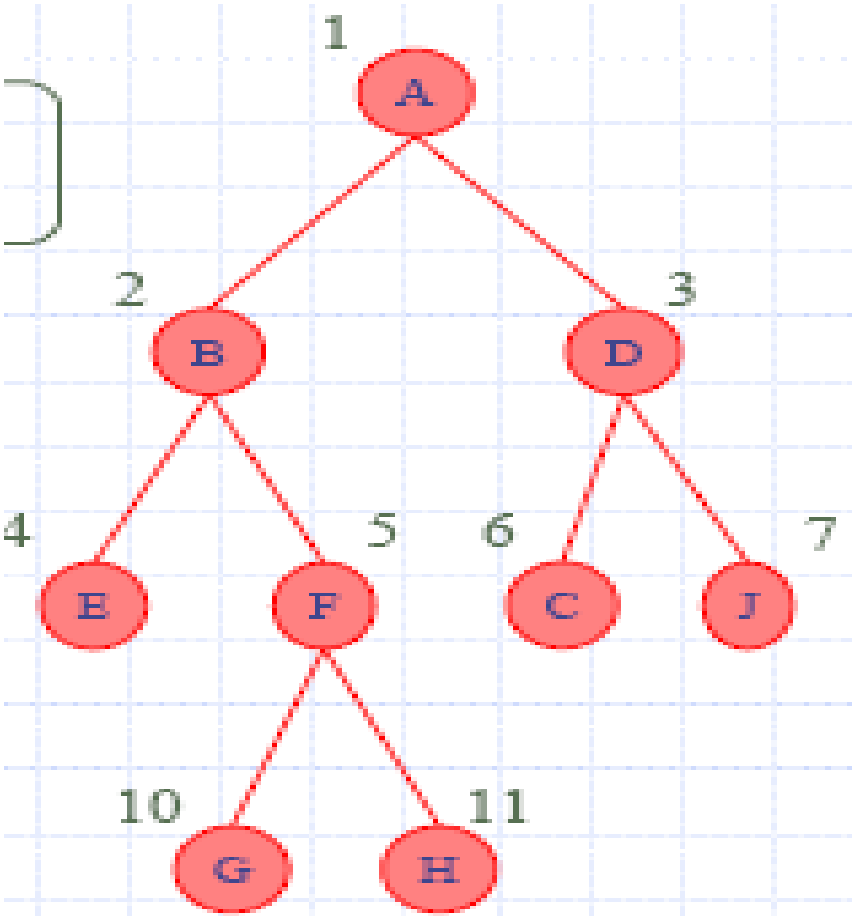
rank (root) = 1

If node is left child

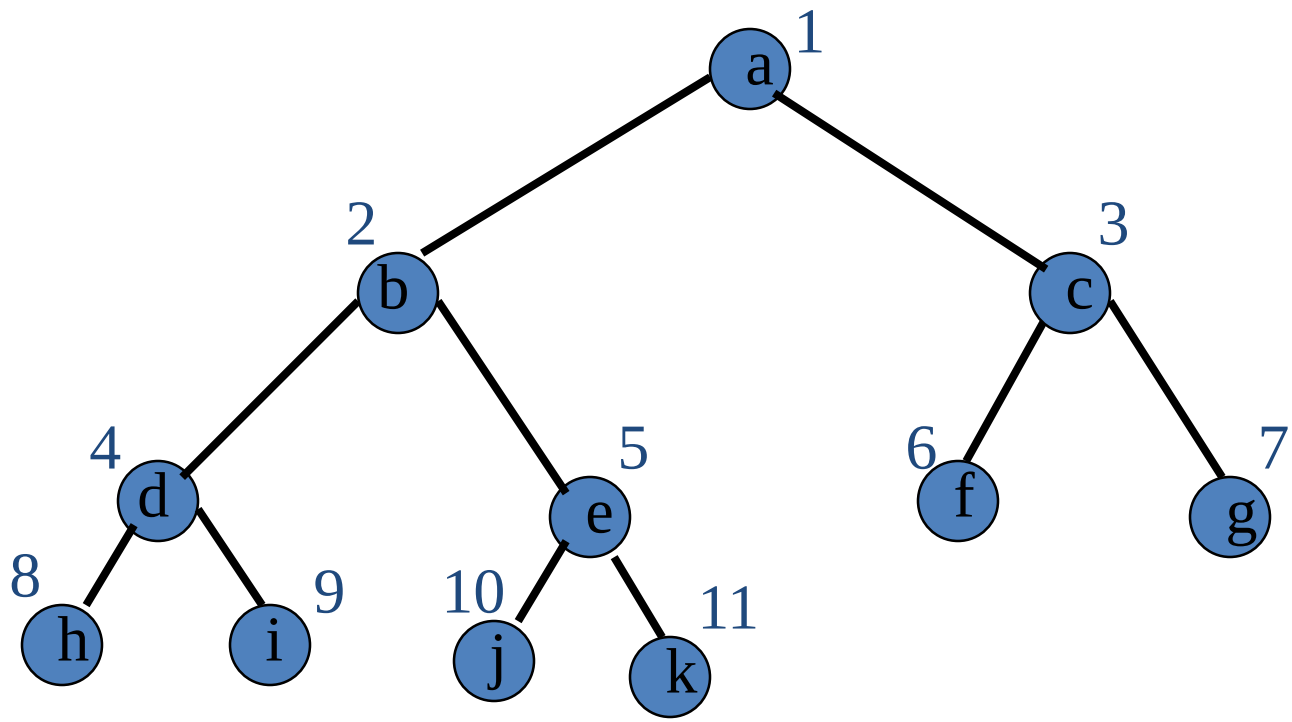
rank (node) = 2 *rank* (*parent* (node))

If node is right child

rank (node) = 2 *rank* (*parent* (node)) + 1



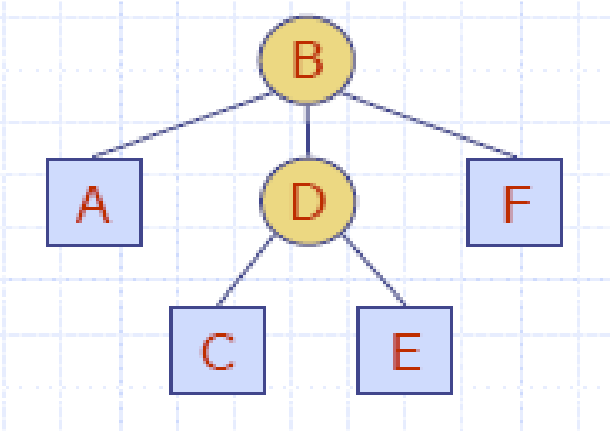
Array Representation

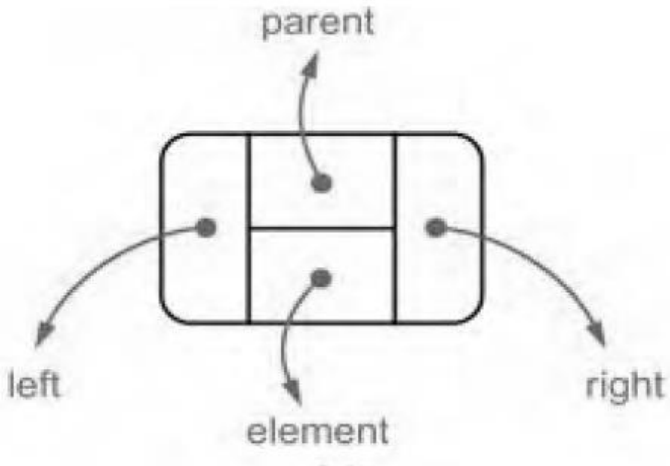


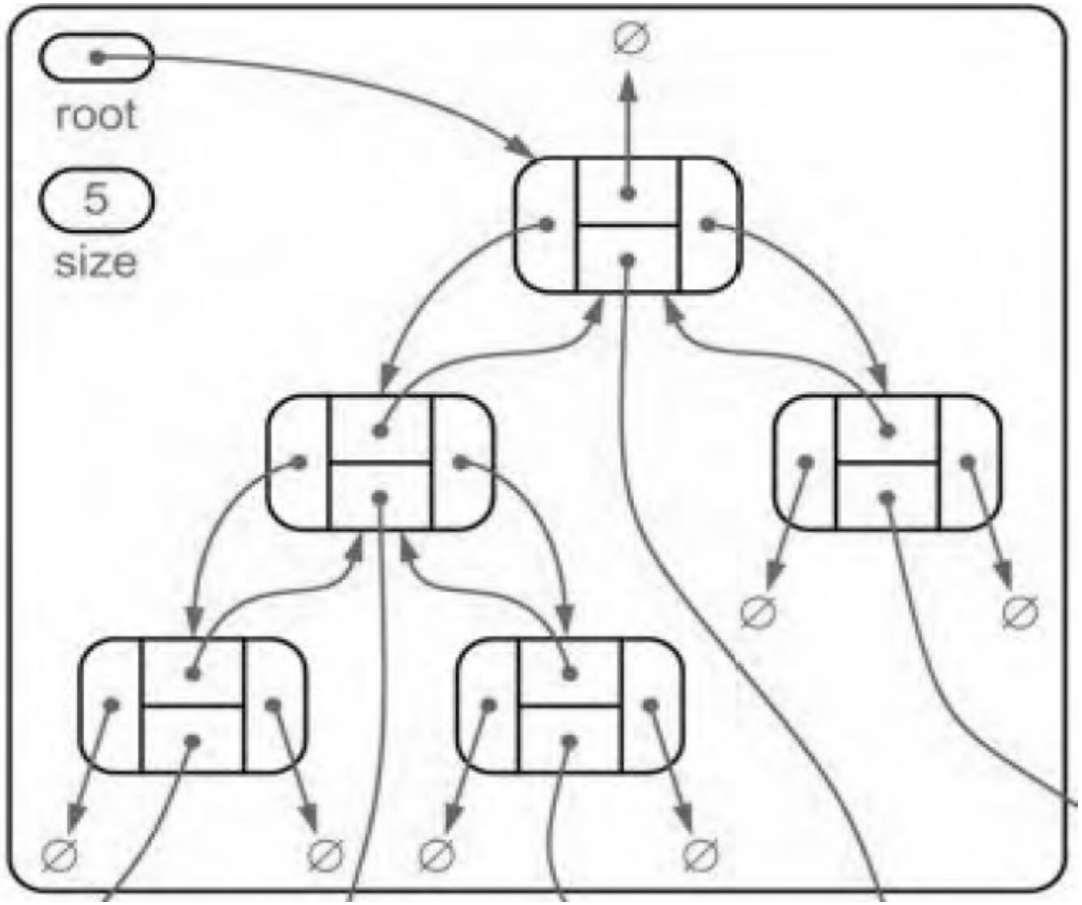
List Implementation of Binary Tree

The Binary Tree node now has four components

- element, parent, left and right

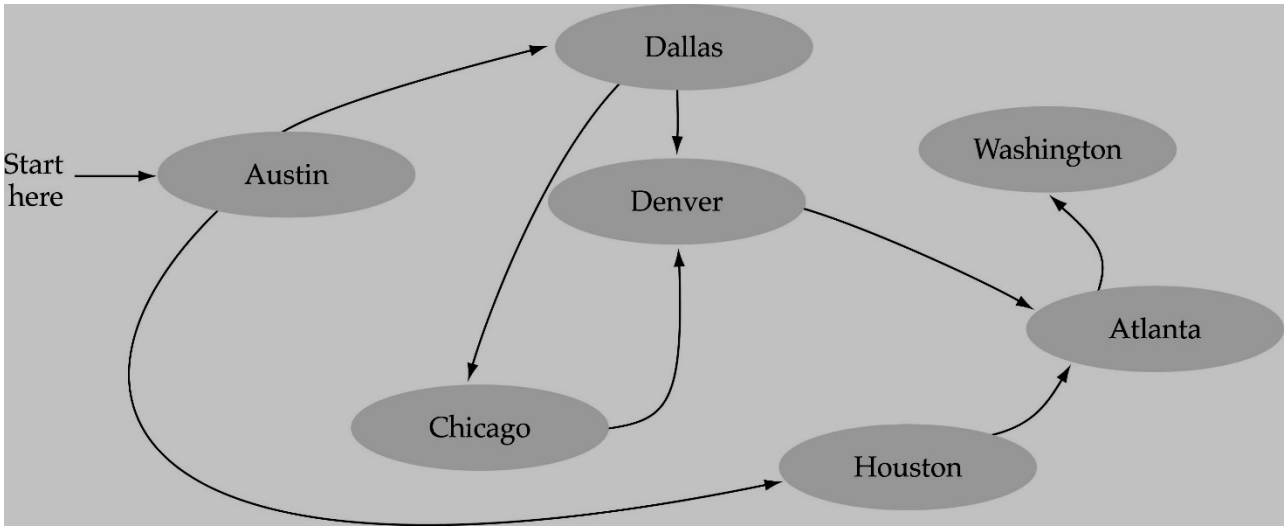






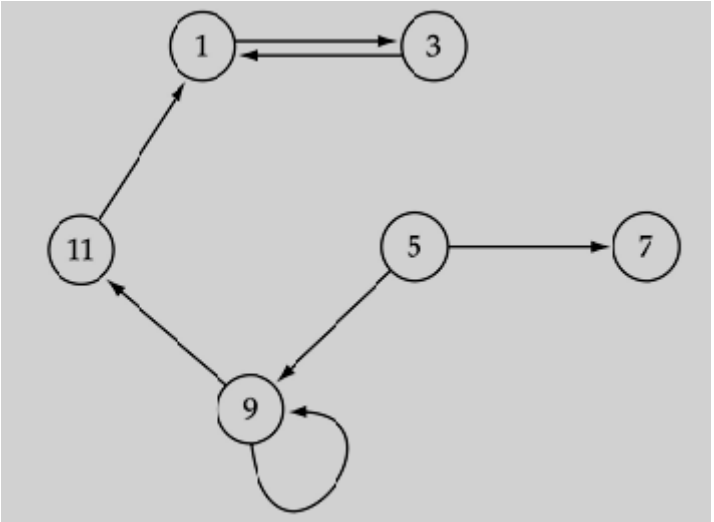
Graph

- A graph is a finite set of nodes with edges between nodes
- The set of edges describes relationships among the vertices

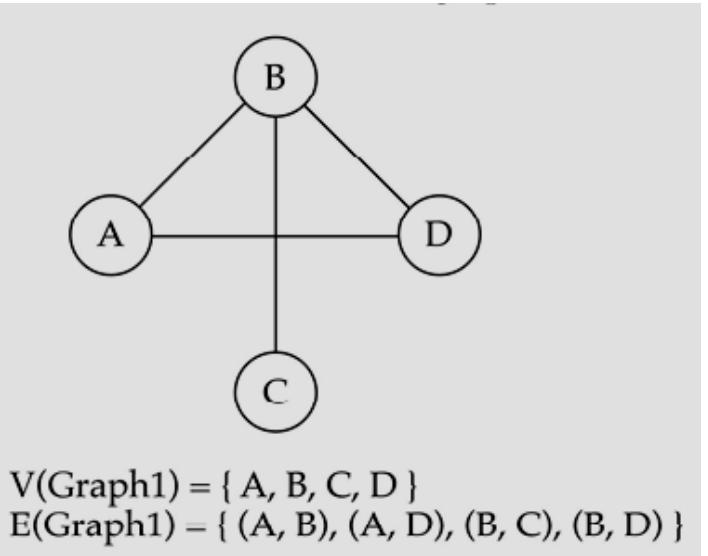


Directed vs. Undirected Graphs

- Directed graph or digraph
  - If the directions of the edges matter, then we show the edge directions



- Undirected graph
  - If the relationships represented by the edges are symmetric, then we don't show the directions of the edges



Graph Terminology

- Adjacent nodes
  - two nodes are adjacent if they are connected by an edge
- Path
  - a sequence of vertices that connect two nodes in a graph
  - Length of a Path is number of edges in a path
- Loops
  - Edges joining a vertex to itself
- Complete graph
  - a graph in which every vertex is directly connected to every other vertex
- Weighted graph
  - a graph in which each edge carries a value

Graph Traversal Techniques

- To find a path between two nodes of the graph

- Two standard graph traversal techniques:
  1. Breadth-First Search (BFS)
  2. Depth-First Search (DFS)

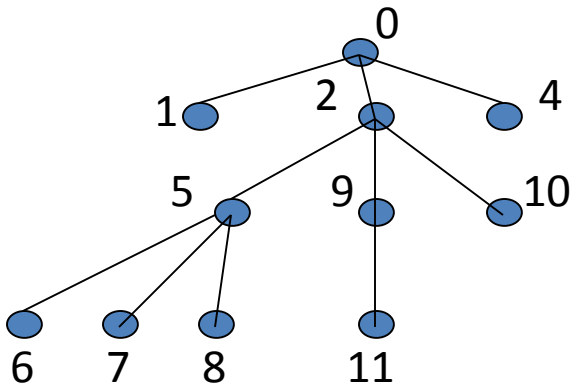
- Both BFS and DFS give rise to a tree
  - When a node x is visited, it is labeled as visited, and it is added to the tree
  - If the traversal got to node x from node y, y is viewed as the parent of x, and x a child of y

Breadth First Search (BFS)

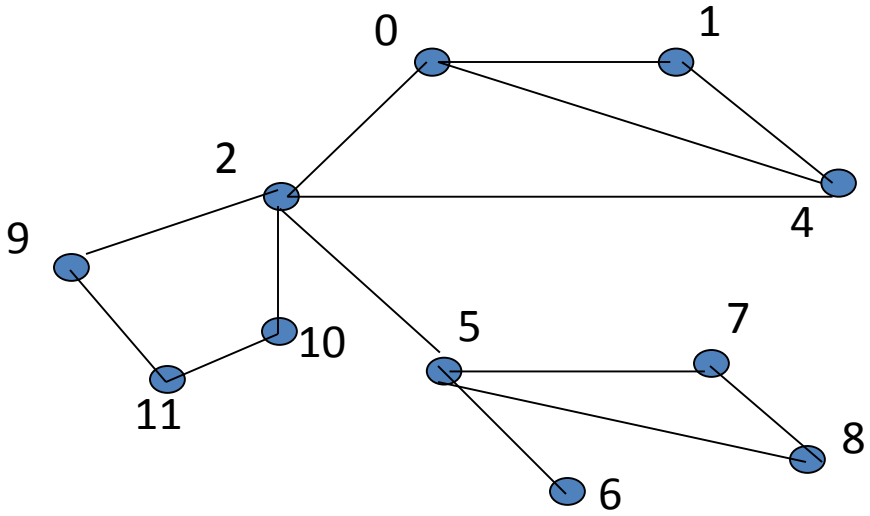
- Idea behind BFS
  - Look at all possible paths at the same depth before you go at a deeper level
  - Back up *as far as possible* when you reach a "dead end" (i.e., next vertex has been "marked" or there is no next vertex)

- BFS can be implemented efficiently using a queue
  - the first node visited in each level is the first node from which to proceed to visit new nodes

Illustration of BFS



BFS Tree



Graph G

BFS - Pseudocode

```
BFS(input: graph G) {
    Queue Q;    Integer x, z, y;
    while (G has an unvisited node x) {
        visit(x); Enqueue(x,Q);
        while (Q is not empty){
            z := Dequeue(Q);
            for all (unvisited neighbor y of z){
            visit(y); Enqueue(y,Q);
            }
        }
    }
}
```

Depth First Search (DFS)

- Idea behind DFS
  - Travel as far as you can down a path
  - Back up as little as possible when you reach a "dead end" (i.e., next vertex has been "marked" or there is no next vertex)

- DFS can be implemented efficiently using a stack
  - the first node visited in each level is the last node from which to proceed to visit new nodes

DFS - Pseudocode

```
DFS(input: Graph G) {
    Stack S; Integer x, t;
    while (G has an unvisited node x){
        visit(x); push(x,S);
        while (S is not empty){
            t := peek(S);
            if (t has an unvisited neighbor y){
            visit(y); push(y,S); }
            else
                pop(S);
        }
    }
}
```

Summary

- Trees are two dimensional data structures incorporating parent-child relationship
- Binary Trees are among the most used data structures
- Arithmetic expressions (and polynomials) are best represented and evaluated using Binary Trees
- The height of a Tree is crucial for efficient implementation of algorithms using it
- A graph is a finite set of nodes with edges between nodes
- Two graph traversal techniques – BFS and DFS