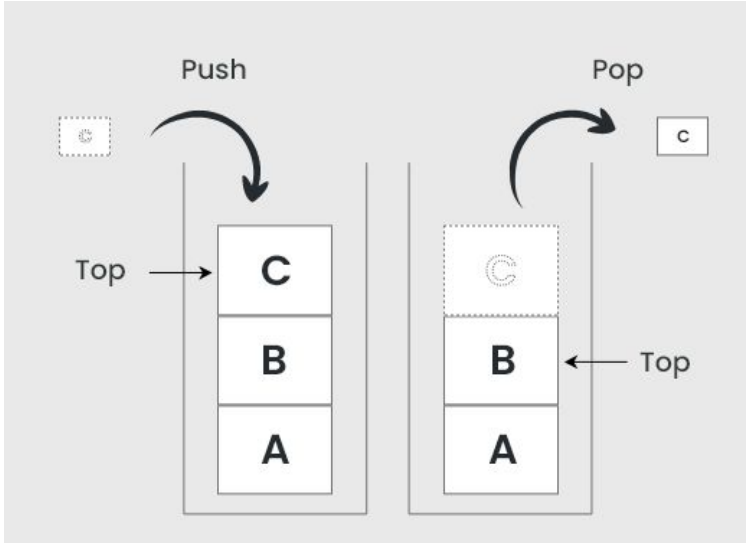
BASIC INTRODUCTION TO STACKS, QUEUES, TREES AND GRAPHS - GENERAL SEARCH ALGORITHMS – SEARCHING FOR SOLUTIONS – PROBLEM-SOLVING AGENTS – CONTROL STRATEGIES – UNINFORMED SEARCH METHODS – BREADTH FIRST SEARCH –UNIFORM COST SEARCH - DEPTH FIRST SEARCH -DEPTH LIMITED SEARCH – INFORMED SEARCH - GENERATE AND TEST - BEST FIRST SEARCH - A* ALGORITHM.

Unit II

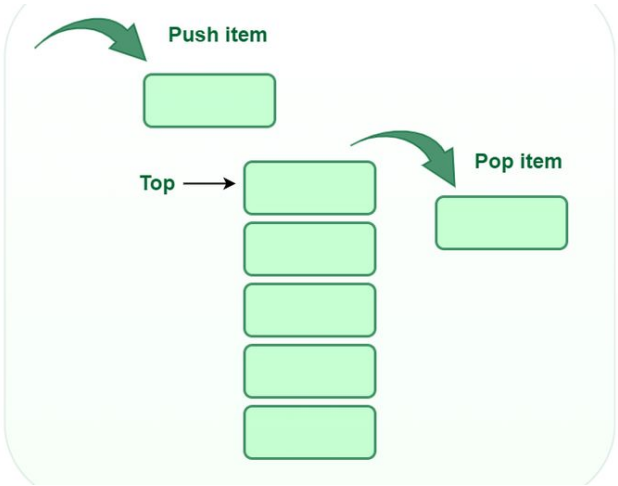
STACKS

- **Stack** is a linear data structure that follows a particular order in which the operations are performed. The order may be **LIFO(Last In First Out)** or **FILO(First In Last Out)**.



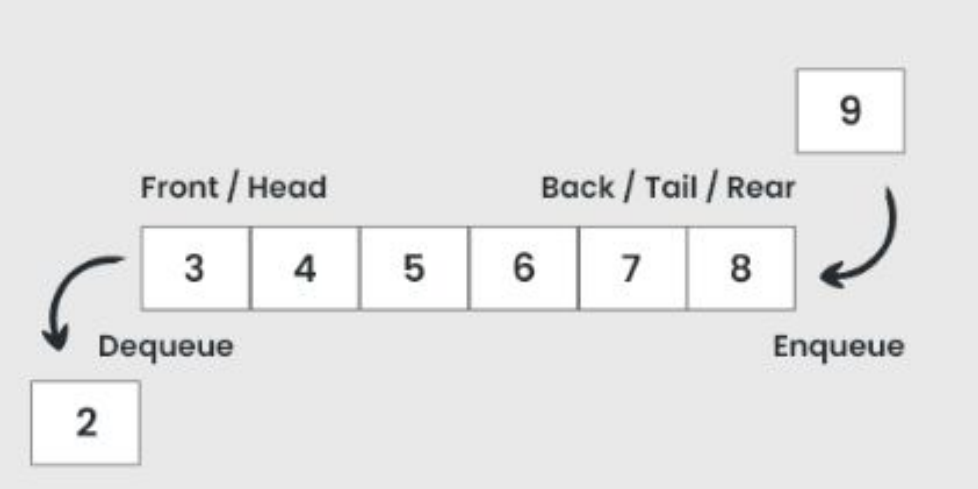
OPERATIONS ON STACK

- In order to make manipulations in a stack, there are certain operations provided to us.
- **push()** to insert an element into the stack
- **pop()** to remove an element from the stack
- **top()** Returns the top element of the stack.
- **isEmpty()** returns true if stack is empty else false.
- **size()** returns the size of stack.



QUEUES

- A **Queue** is defined as a linear data structure that is open at both ends and the operations are performed in **First In First Out (FIFO)** order.

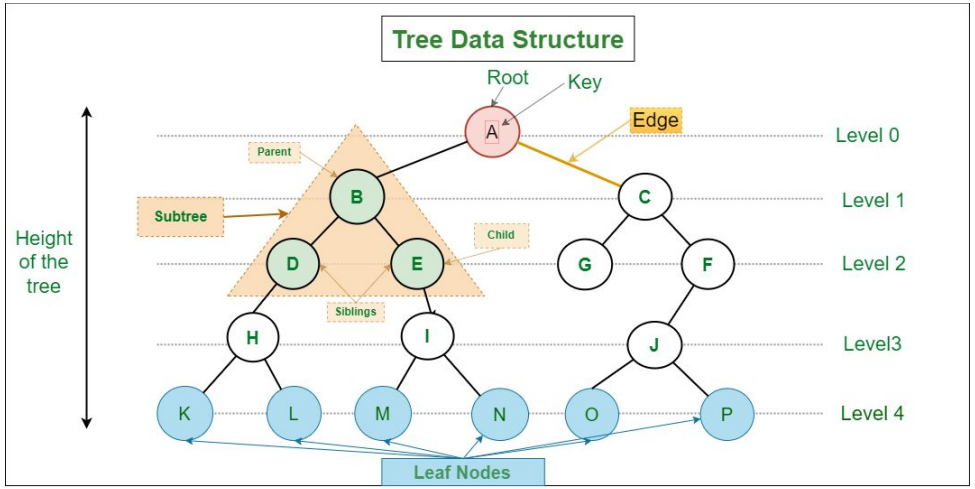


OPERATIONS ON QUEUE

- Some of the basic operations for Queue in Data Structure are:
- **Enqueue() –** Adds (or stores) an element to the end of the queue..
- **Dequeue() –** Removal of elements from the queue.
- **front()-** Acquires the data element available at the front node of the queue without deleting it.
- **rear() –** This operation returns the element at the rear end without removing it.
- **isFull() –** Validates if the queue is full.
- **isNull() –** Checks if the queue is empty.

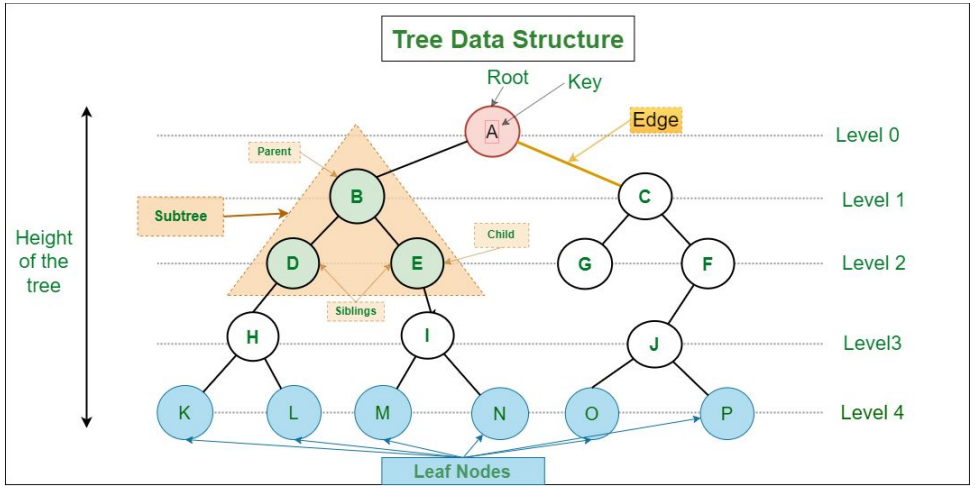
TREES

- A **tree data structure** is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges.
- The topmost node of the tree is called the root, and the nodes below it are called the child nodes. Each node can have multiple child nodes, and these child nodes can also have their own child nodes, forming a recursive structure.



TERMINOLOGIES IN TREE DATA STRUCTURE

- **Parent Node:** The node which is a predecessor of a node is called the parent node of that node. **{B}** is the parent node of **{D, E}**.
- **Child Node:** The node which is the immediate successor of a node is called the child node of that node. Examples: **{D, E}** are the child nodes of **{B}.**
- **Root Node:** The topmost node of a tree or the node which does not have any parent node is called the root node. {A} is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.
- **Leaf Node or External Node:** The nodes which do not have any child nodes are called leaf nodes. **{K, L, M, N, O, P, G}** are the leaf nodes of the tree.
- **Ancestor of a Node:** Any predecessor nodes on the path of the root to that node are called Ancestors of that node. **{A,B}** are the ancestor nodes of the node **{E}**
- **Descendant:** Any successor node on the path from the leaf node to that node. **{E,I}** are the descendants of the node **{B}.**
- **Sibling:** Children of the same parent node are called siblings. **{D,E}** are called siblings.
- **Level of a node:** The count of edges on the path from the root node to that node. The root node has level **0**.
- **Neighbour of a Node:** Parent or child nodes of that node are called neighbors of that node.
- **Subtree**: Any node of the tree along with its descendant.

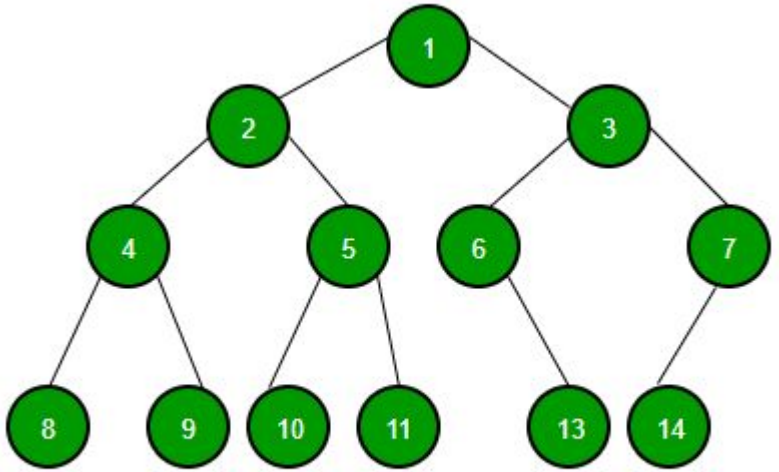


TYPES OF TREE

- Binary tree
- Binary search tree
- AVL tree

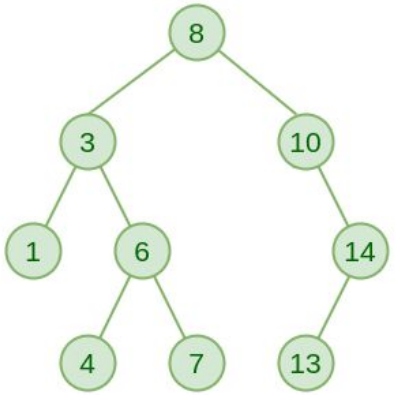
BINARY TREE

- Binary Tree is defined as a tree data structure where each node has at most 2 children. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.
- **Basic Operation On Binary Tree:**
  - Inserting an element.
  - Removing an element.
  - Traversing the tree.



BINARY SEARCH TREE

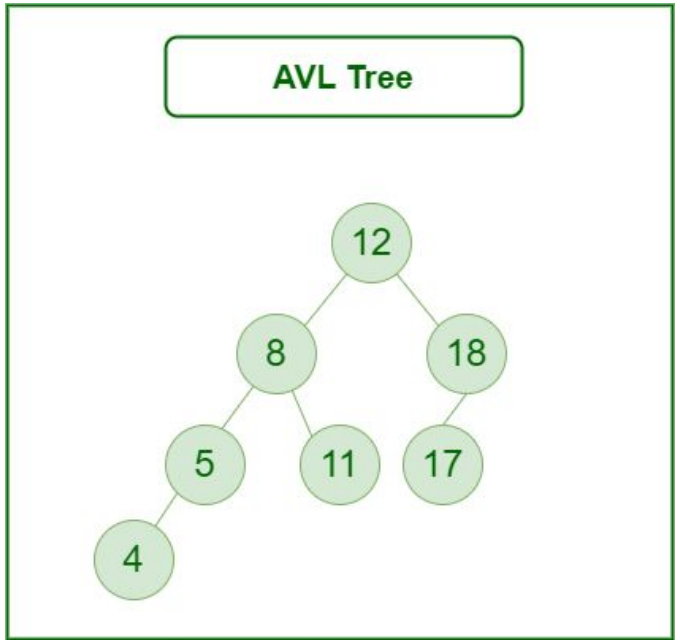
- **Binary Search Tree** is a node-based binary tree data structure which has the following properties:
  - The left subtree of a node contains only nodes with keys lesser than the node's key.
  - The right subtree of a node contains only nodes with keys greater than the node's key.
  - The left and right subtree each must also be a binary search tree
- **<u>Basic operations:</u>**
  - Insertion
  - Searching
  - Deletion
  - BST traversal- inorder, preorder, postorder



Binary Search Tree

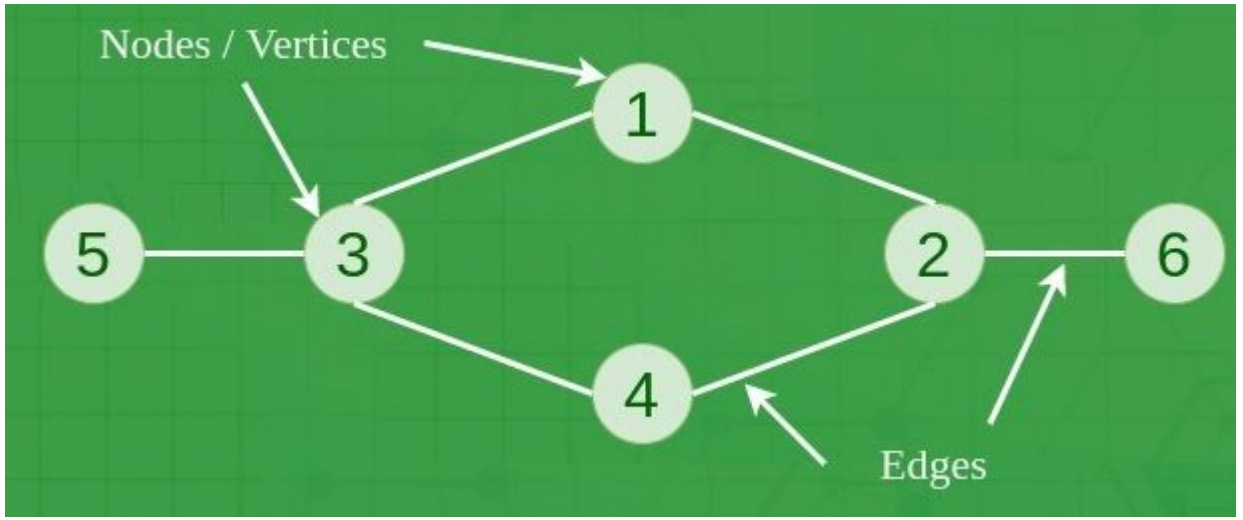
AVL TREE

- An **AVL tree** defined as a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees for any node cannot be more than one.
- The difference between the heights of the left subtree and the right subtree for any node is known as the **balance factor** of the node.
- **Basic operations:**
  - Insertion
  - Deletion
  - searching



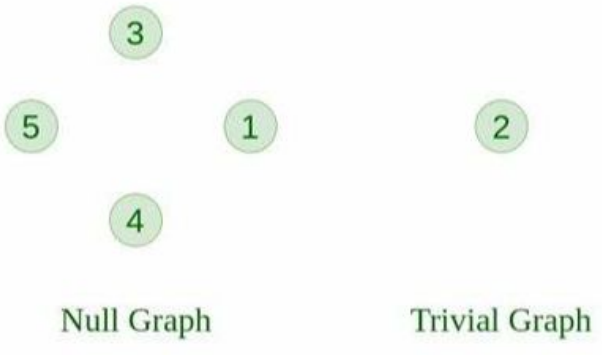
GRAPHS

- A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph.
- More formally a Graph is composed of a set of vertices(V) and a set of edges(E). The graph is denoted by G(E, V).

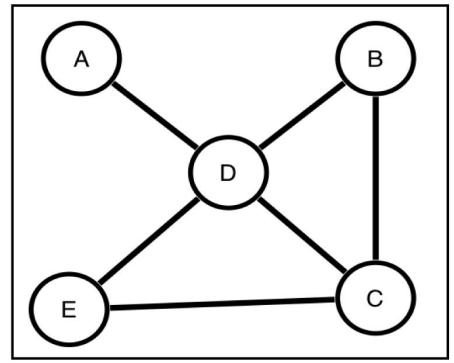


TYPES OF GRAPH

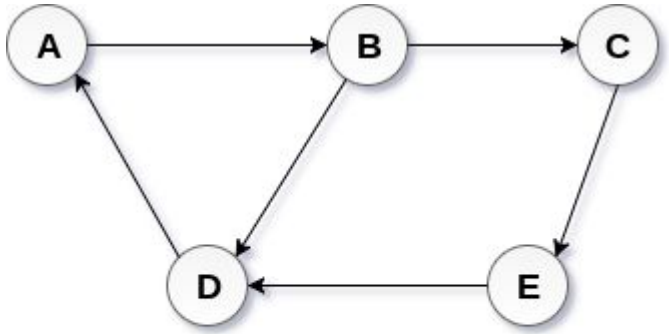
- Null Graph: A graph is known as a null graph if there are no edges in the graph.
- Trivial Graph: Graph having only a single vertex, it is also the smallest graph possible.



- Undirected Graph: A graph in which edges do not have any direction.

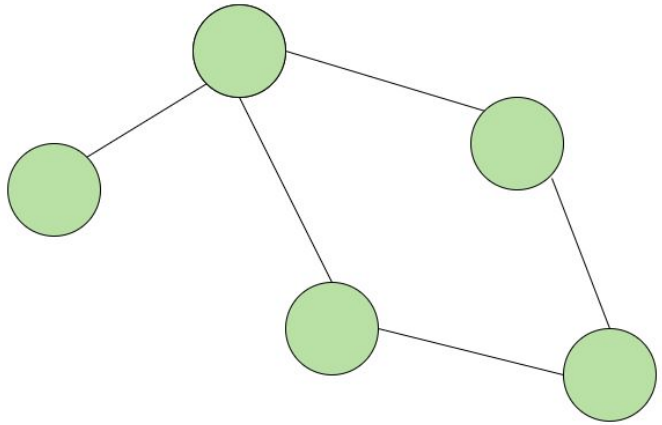


- Directed Graph: A graph in which edge has direction.

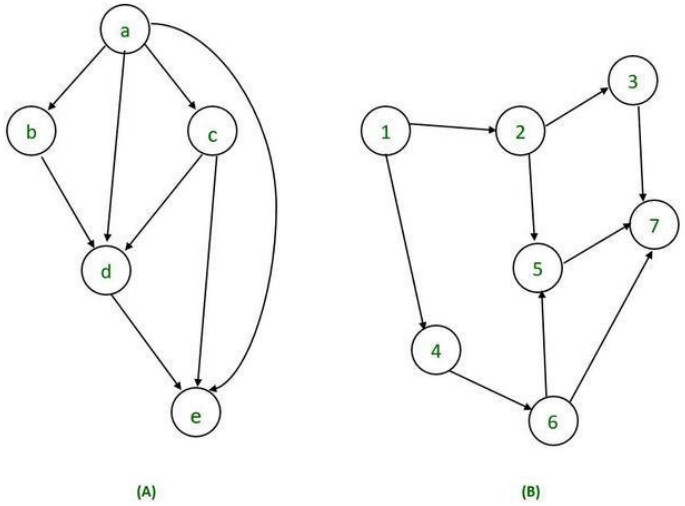


Directed Graph

- Cyclic graph: A **cyclic graph** is defined as a graph that contains at least one cycle which is a path that begins and ends at the same node.

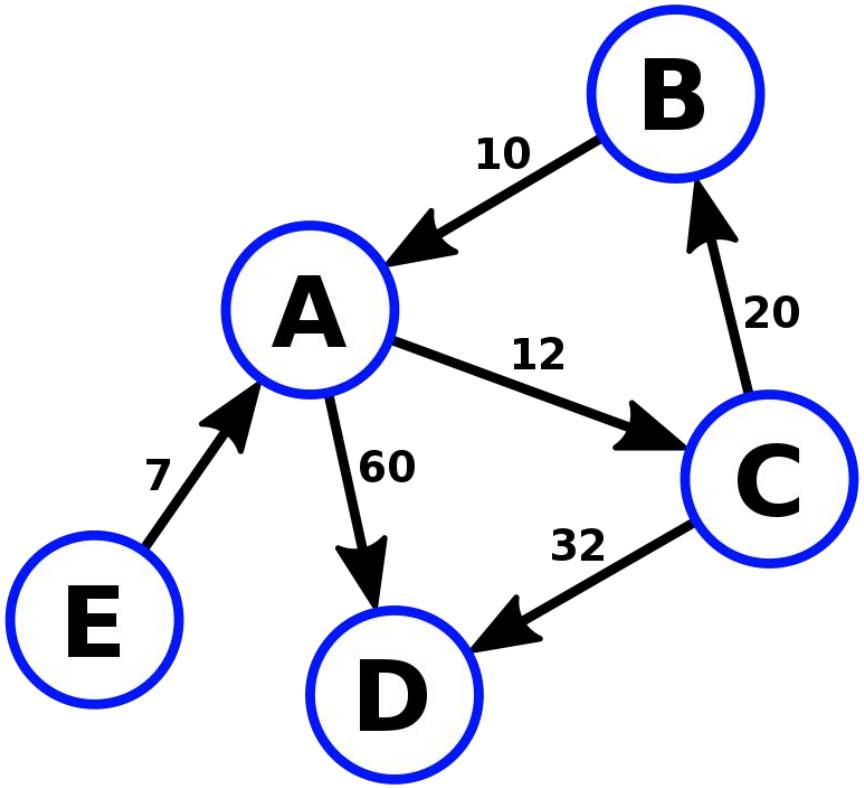


- Directed acyclic graph: A **Directed Acyclic Graph (DAG)** is a directed graph that does not contain any cycles.

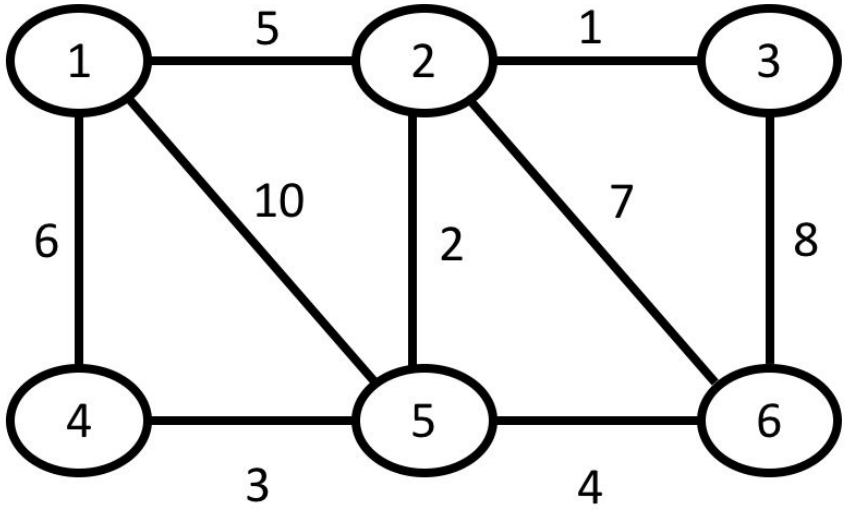


- Weighted Graph: A graph in which the edges are already specified with suitable weight is known as a weighted graph. Weighted graphs can be further classified as:
  - directed weighted graphs and
  - undirected weighted graphs.

Directed weighted graph



Undirected weighted graph



GENERAL SEARCH ALGORITHM

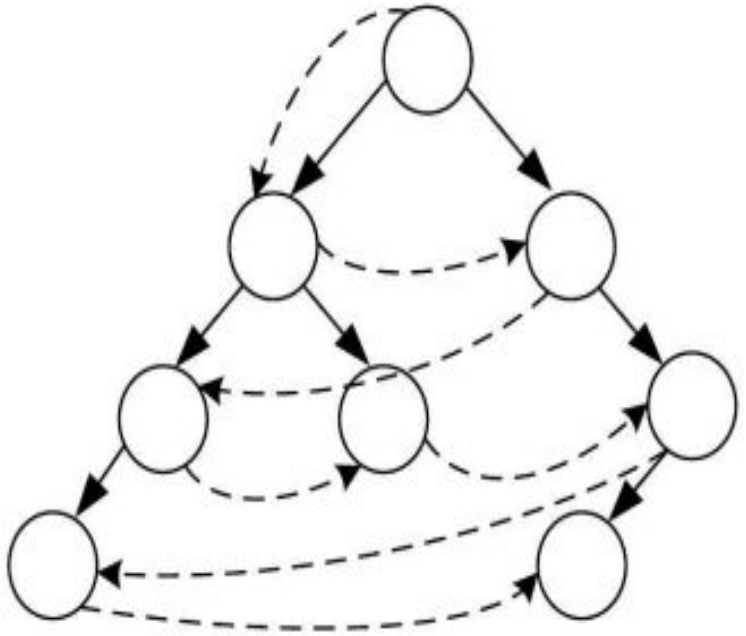
- Searching for solutions:
  - It is typically finding out path or route to goal state. While doing that, we enter into various states. It is referred as intermediate step during the course of search.
  - State space – is defined as the collection of all possible configurations of the system i.e. it is a set of all states that can be reached from initial states with all possible legal actions.
  - State space search- is the process of searching the state space for a solution to reach the goal.
- Problem solving agent:
  - An agent can be termed as an entity that can perceive the environment and act on it.
  - An agent acts and tries to improve its performance based on the outcome.
  - An agent formulates the goal as well as the problem. These are the basic steps that are required for solving any problem.
  - The following steps describe the process:
    - Formulate the goal
    - Formulate the problem which has the goal and initial states
    - Now search with the given problem- sequences of actions
    - Action on the sequence- act according to the sequence of actions

CONTROL STRATEGIES

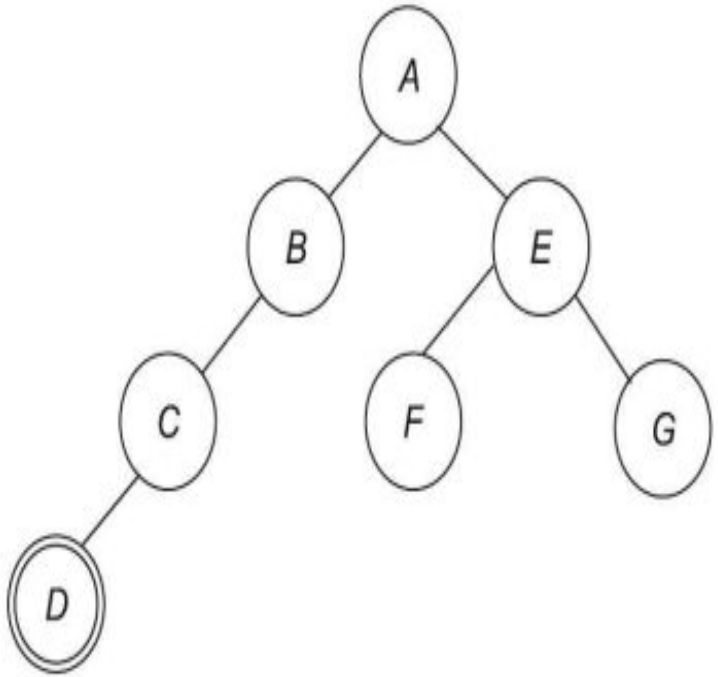
- Control strategies define how to reach or what way to follow while searching for a solution.
- The rules play an important role in the decision making, where and which rule is to be applied is decided based on the search strategy.
- The most common strategies used are:
  - Forward search
  - Backward search
  - Systematic search
  - Heuristic search
- Parameters for search evaluation:
  - Completeness
  - Space and time complexity
  - Optimality

UNINFORMED SEARCH

- This search method belongs to the category of methods that have no prior knowledge of the problem.
- They are generally exhaustive methods that search all possible paths. It also referred to as blind search.
- Example: searching a boy named Ram in a school.
- Breadth first search and Depth first search are the two basic approaches of uninformed search.
- Breadth first search:
  - Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.
  - To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.



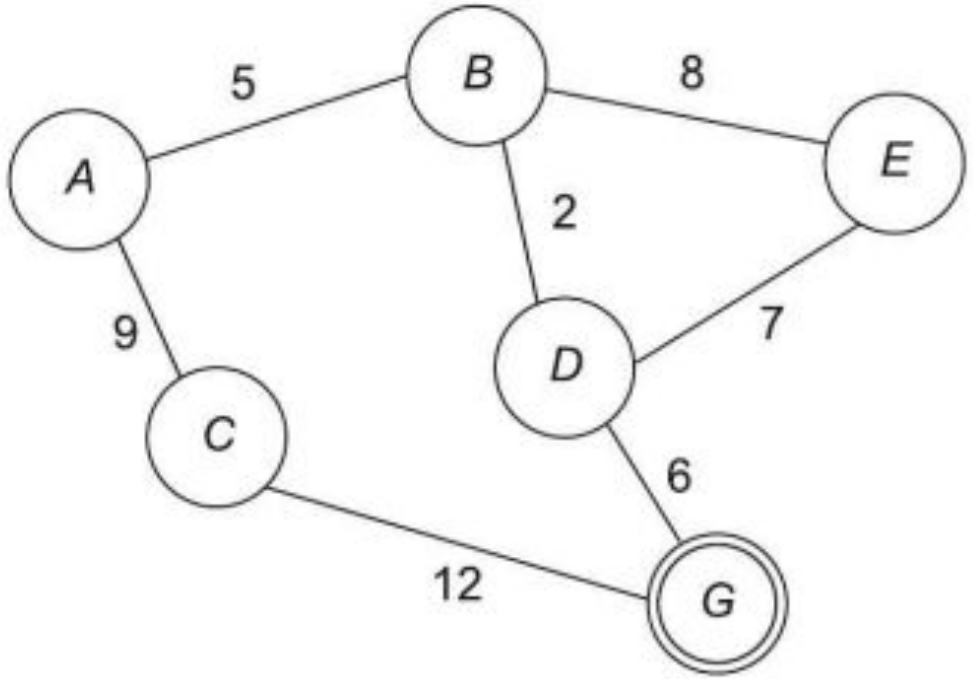
Breadth first search tree



In BFS, the step-wise working is as follows:

Queue	*Check*
A	—
	Removed *A*, is it goal? No, add children. So, *B* and *E* are added.
BE	
E	Removed *B*, is it goal? No, add children.
	So, *C* is added at the end of queue.
EC	
C	Removed *E*, is it goal? No, add children.
	So, *F* and *G* are added.
CFG	
FG	Removed *C*, is it goal? No, add children.
	So, *D* is added.
FGD	
GD	Removed *F*, is it goal? No, add children.
	Cannot be added, leaf node, so remove the next node.
D	Removed *G*, is it goal? No, add children.
	Cannot be added, leaf node, so remove the next node.
Empty	Removed *D*, is it goal? YES!

UNIFORM COST SEARCH

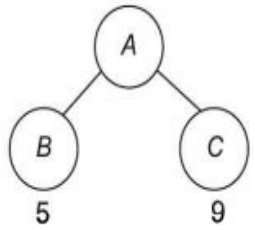


Let us discuss how the uniform cost search proceeds step-wise.

Step 1: Start from *A* Expanded none

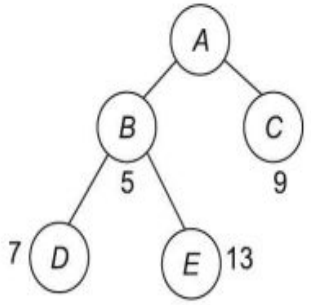
Step 2: Frontier Expanded none

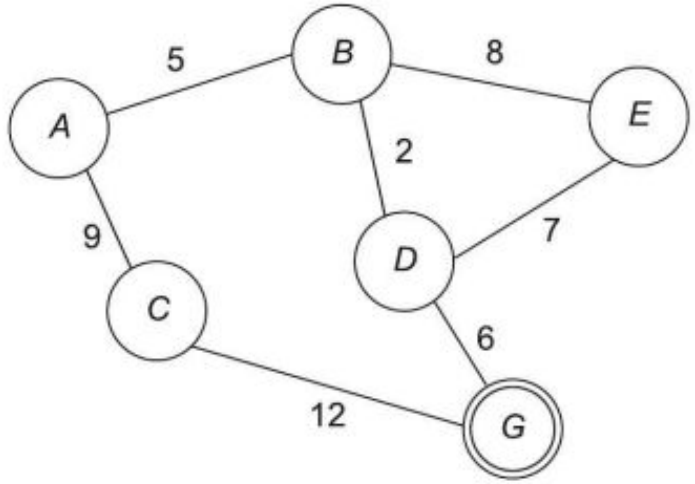
Step 3: Frontier: *B* and *C* Expanded *A*



Select the lowest, i.e., *B*

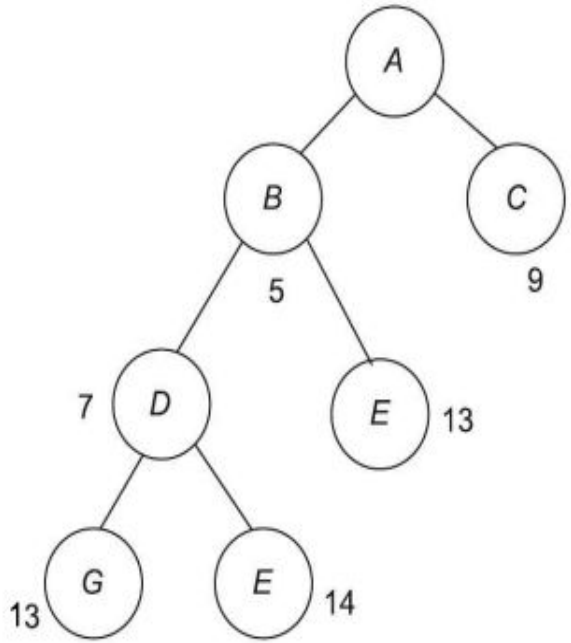
Step 4: Frontier: *D*, *C*, *E* (priority queue) Expanded *A*, *B*

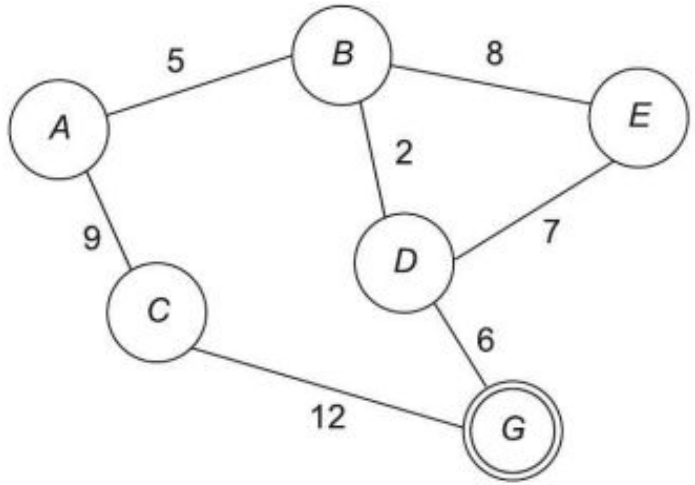




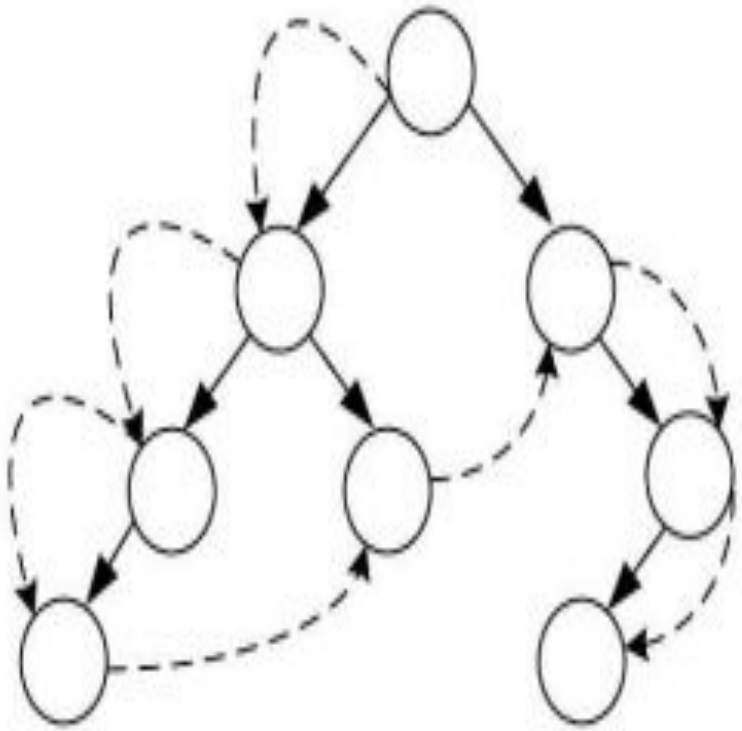
Select the lowest, i.e., *D*.

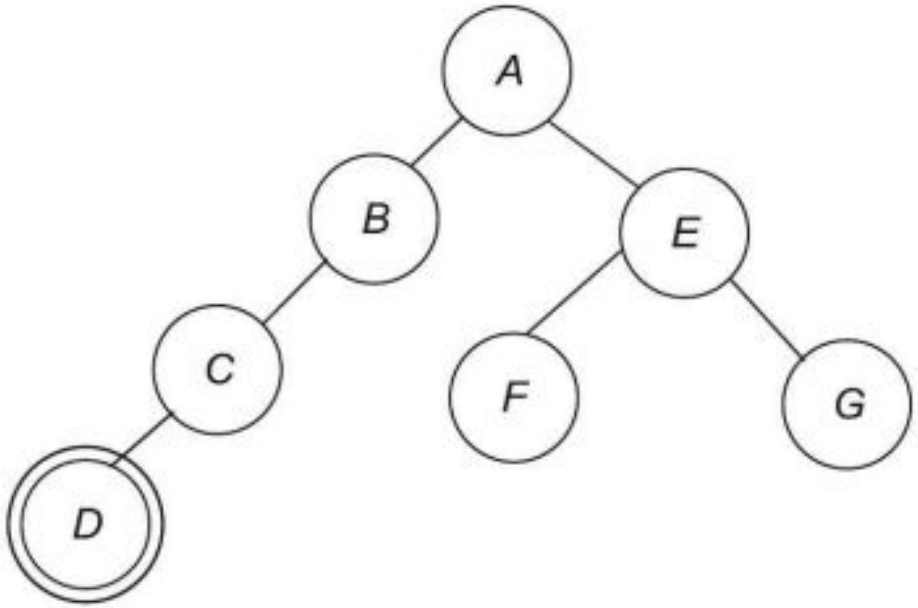
Step 5: Frontier (*C*, *E*, *E*, *G*) Expanded *A*, *B*, *D*



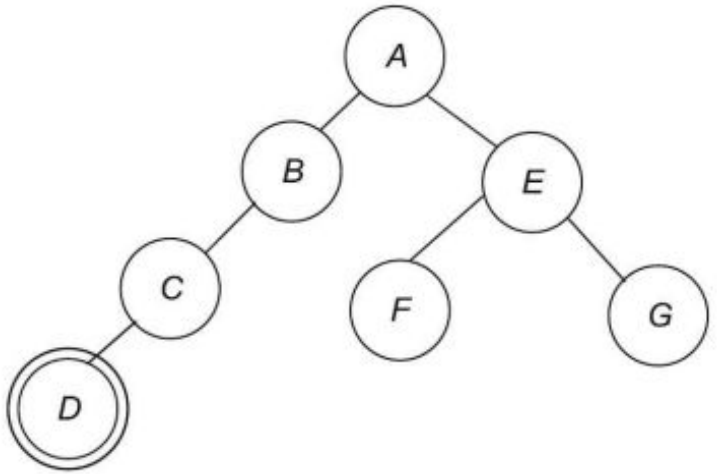


DEPTH FIRST SEARCH





	Stack	
Step 1:	$A \leftarrow$ top	Pop A, is it goal? No, push its children on the stack.
	$B \leftarrow$ top E	A's successor is pushed.
Step 2:	$B \leftarrow$ top E	Pop B, is it goal? No, push its children on the stack.
	$C \leftarrow$ top E	B's successor is pushed.
Step 3:	$C \leftarrow$ top	Pop C, is it goal? No, push its children on the stack.
	$D \leftarrow$ top E	C's successor is pushed.
Step 4:	$D \leftarrow$ top E	Pop D, is it goal?? YES!



	Stack	
Step 1:	$E \leftarrow$ top	After having traversed the steps till D, stack will have E. So, pop E, is it goal? No, push its children.
Step 2:	$F \leftarrow$ top G	E's successor is added.
Step 3:	$F \leftarrow$ top G	Pop F, is it goal?? YES!
	$G \leftarrow$ top