

### Statement Purpose:

Purpose of this Lab is to familiarize the students with the some other abstract data structures like graphs, heaps and hash tables. Another aim is to teach the students how to traverse the graphs, how to construct min-heap and max-heap and how to use different collision resolution techniques. The students are given small tasks related to above mentioned abstract data structures which help them understand the concepts well which they learn in their lectures.

### Activity Outcomes:

The students will learn how to

- traverse a graph using Depth-First-Search Traversal and Breadth-First-Search Traversal algorithms
- construct min-heap or max-heap from given data set and delete an element from heap
- construct hash table using various collision resolution techniques like linear probing, quadratic probing and double hashing.

### Theory Review (20 Minutes):

#### Graphs

In computer science, a **Graph**  $G(V, E)$  is an abstract data structure that consists of a finite set of vertices (or nodes or points)  $V$  and a finite set of edges  $E$ , connecting these vertices.

A graph can be implemented using adjacency matrix or adjacency list.

A graph can be traversed using Depth-First-Search traversal or Breadth-First-Search traversal techniques.



## Depth-First-Search Traversal Algorithm

### Depth First Search Algorithm

The *Depth First Search (DFS)* for a connected graph consists of the following steps:

*Step #1: Initialize all vertices to mark as unvisited (ready state)*

*Step #2: Select an arbitrary vertex, push it to stack (wait state)*

*Step #3: Repeat steps # 4 through Step #5 until the stack is empty*

*Step #4: Pop off an element from the stack. Process and mark it as visited (processed)*

*Step #5: Push to stack adjacent vertices of the processed vertex. Go to Step # 3.*

## Breadth-First-Search Traversal Algorithm

### Breadth First Search Algorithm

The *Breadth First Search (BFS)* for a connected graph consists of following steps:

*Step #1: Initialize all vertices to mark as unvisited (ready state)*

*Step #2: Select an arbitrary vertex, add it to queue (wait state)*

*Step #3: Repeat steps # 4 through Step #5 until queue is empty*

*Step #4: Remove an element from the queue. Process and mark it as visited (processed state)*

*Step #5: Enqueue all adjacent vertices of the processed vertex. Go to Step # 3.*



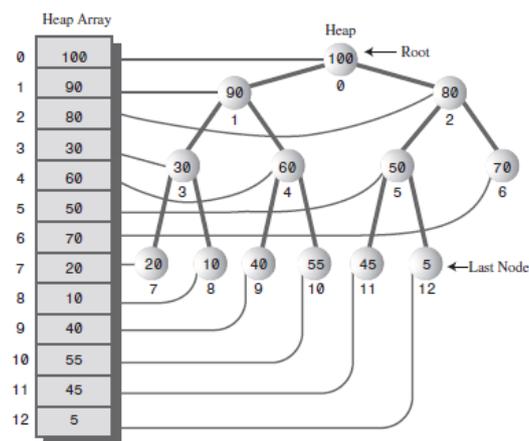
## Heap

A *heap* is a binary tree with the following properties:

- It is a complete binary tree. (In a **complete** binary tree every level, *except possibly the last*, is completely filled, and all nodes in the last level are as far left as possible).
- Each node is greater than or equal to any of its children (in case of Max-heap).

### Note:

- Heap is usually implemented as an array.
- Heaps are mostly used to implement priority queues.



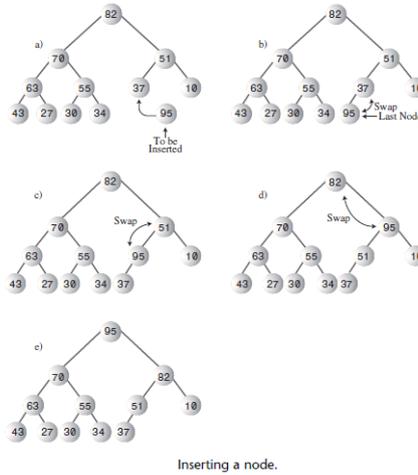
A heap and its underlying array.

### Inserting a node into max-heap

- We add the node at the last position.
- If it avoids the heap property (i.e. in max-heap, each node is greater than or equal to its children), it is swapped with its parent, till it reaches its correct position. This process is called percolate-up or trickle-up.



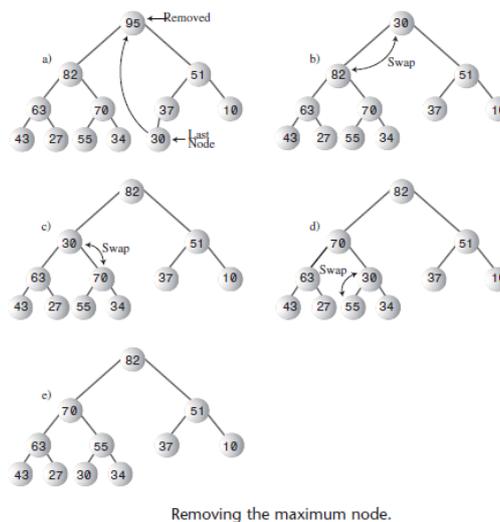
**Example**



**Deleting maximum node from max-heap**

- We Remove the root (as in case of max-heap, maximum value is at root).
- Move the last node into the root.
- Trickle the last node down until it's below a larger node and above a smaller one.
- This process is called percolate-down or trickle-down.

**Example**



## Hash Table

A *hash table* is an abstract data type that stores and retrieves records according to their search key values.

### Note:

- Hash functions are used to store keys into hash table.
- When more than one keys are mapped into same location in the hash table, it is called collision.

## Collision Resolution Techniques

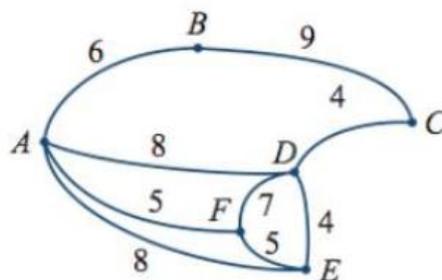
Commonly used collision resolution techniques are

- Open Addressing
  - Linear probing
  - Quadratic probing
  - Double hashing
- Restructuring the hash table
  - Bucket hashing
  - Separate chaining

## Lab Exercises: (60 Minutes)

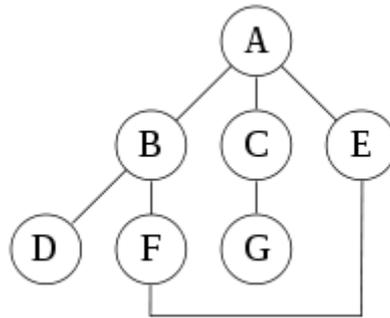
### Q. No. 1 (Graphs):

Perform a Depth-First traversal and Breadth-First traversal of the weighted graph shown below, **starting with vertex A**. Select the smallest-weight edge first, when accessing neighbours of a vertex. What is the order in which the vertices are visited?

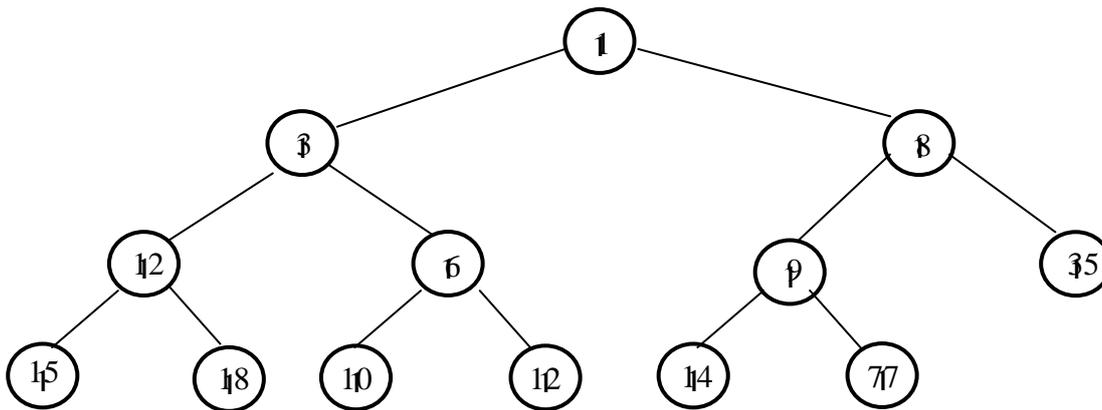


**Q. No. 2 (Graphs):**

Perform a Depth-First traversal and Breadth-First traversal of the unweighted graph shown below, **starting with vertex A**. What is the order in which the vertices are visited?



**Q. No. 3 (Heaps):** Show the result of inserting the item 7 into the min-heap shown below:



**Q. No. 4 (Heaps):** Show the result of removing the minimum element from the min-heap shown above (without inserting 7 into the min-heap above).



**Q. No. 5 (Hash Tables):** Consider a hash table that uses the linear probing, quadratic probing and separate chaining hashing techniques with the following hash function  $h(x) = (5x+4) \pmod{11}$ . (The hash table is of size 11). If we insert the values 3, 9, 2, 1, 14, 6 and 25 into the table, in that order, show where these values would end up in the table?

**Solution:**

**Linear Probing**

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

**Quadratic Probing**

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

**Separate Chaining**

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	



**Q. No. 6 (Hash Tables):** Insert the following numbers (in the order that they are shown.....from left to right) into a hash table with an array of size 10, using the hash function,  $h_1(x) = x \text{ mod } 10$ .

2777, 5322, 1312, 5523, 1789, 4837, 8497, 2400

Show the result of the insertions when hash collisions are resolved through:

**a)** quadratic probing **b)** double hashing **c)** separate chaining.

For part **b)** double hashing, the second hash function used is  $h_2(x) = 4 - (x \text{ mod } 2)$

**Note:** for part (c), separate chaining, insert at the **FRONT** of the linked list.

**Solution:**

**Quadratic Probing**

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

**Double hashing**

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

**Separate Chaining**

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

