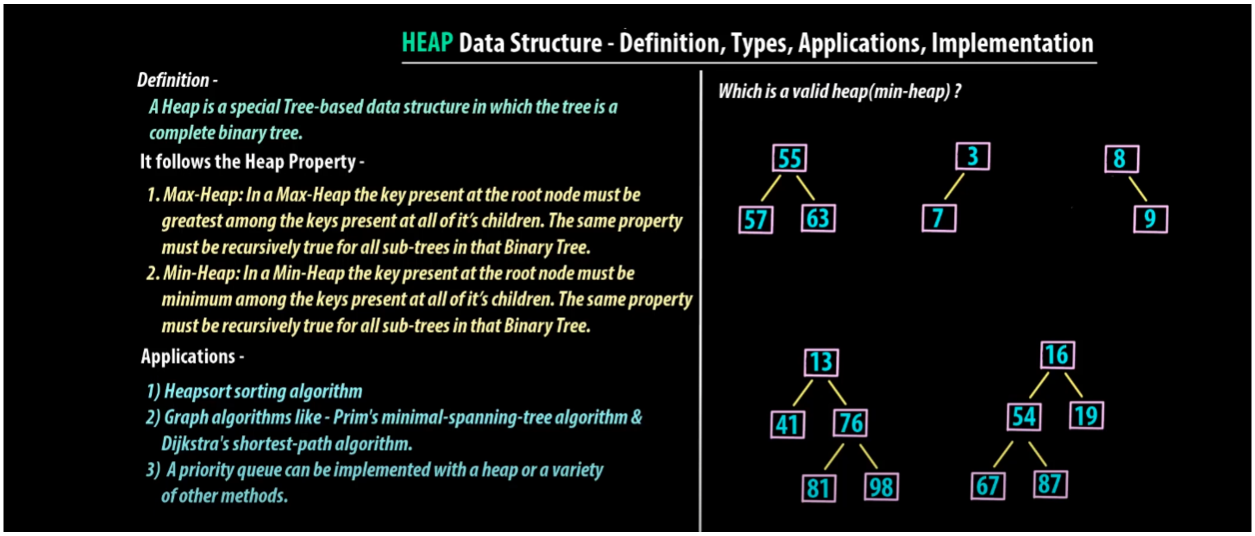
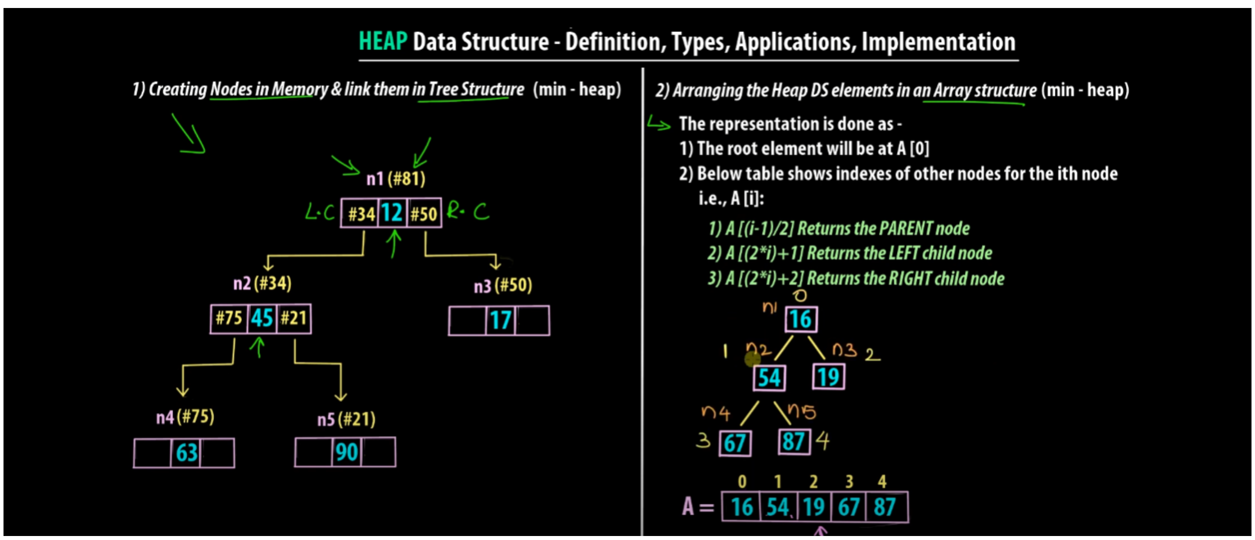
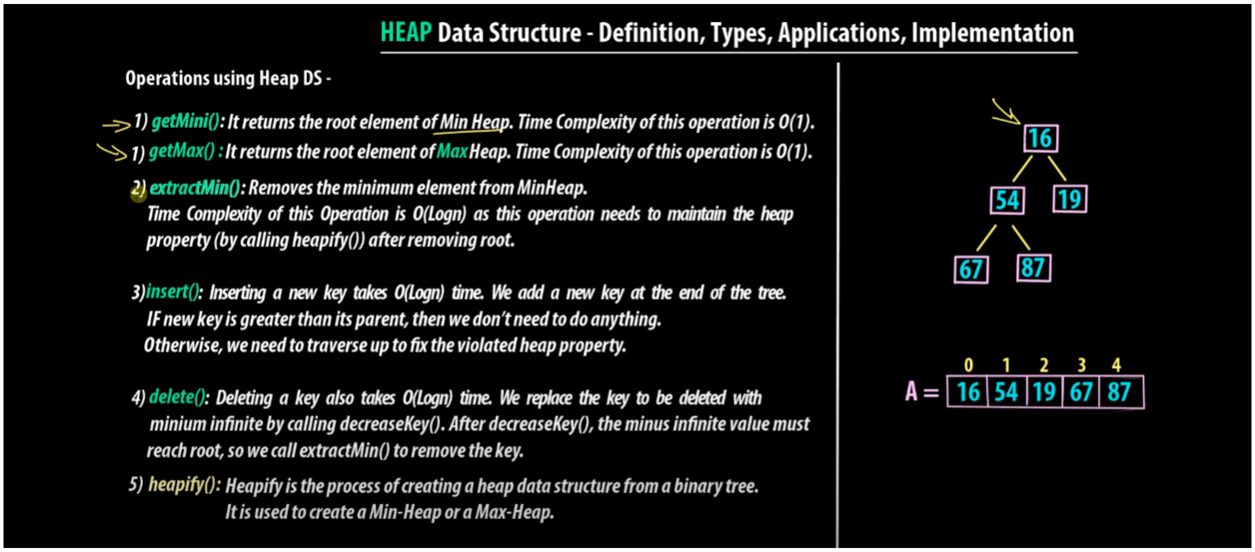
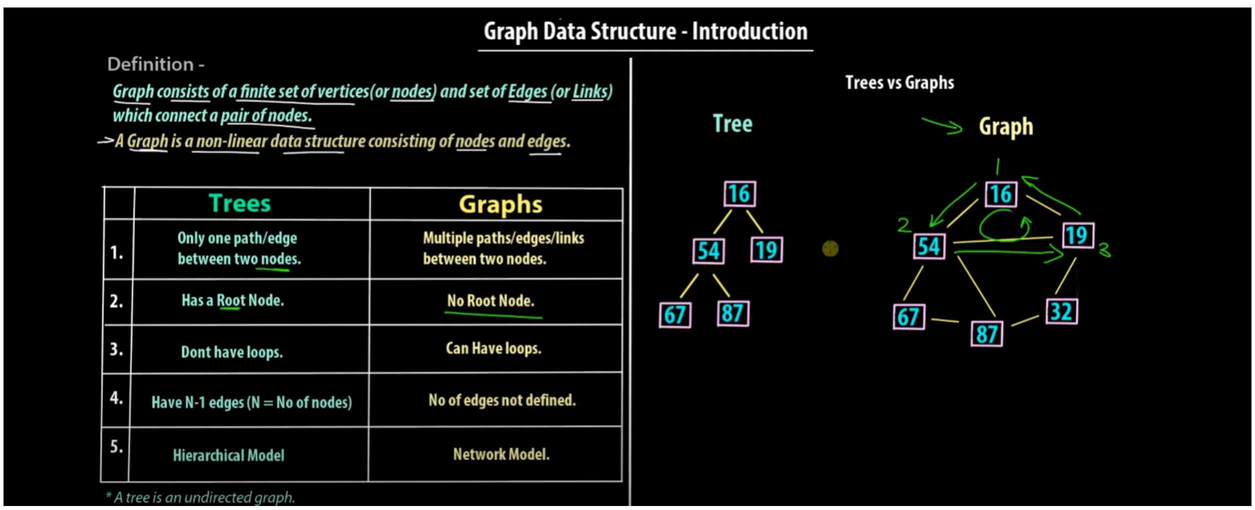
HEAP

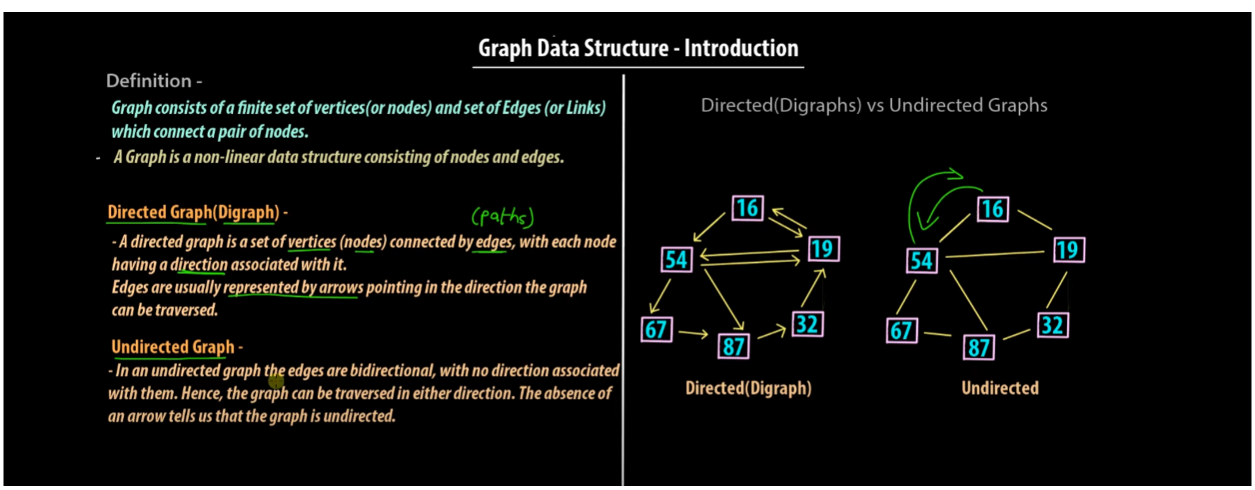

HEAP Data Structure - Definition, Types, Applications, Implementation
Definition -
A Heap is a special Tree-based data structure in which the tree is a complete binary tree.
It follows the Heap Property -
1. Max-Heap: In a Max-Heap the key present at the root node must be greatest among the keys present 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.
Applications -
1) Heapsort sorting algorithm
2) Graph algorithms like - Prim's minimal-spanning-tree algorithm & Dijkstra's shortest-path algorithm.
3) A priority queue can be implemented with a heap or a variety of other methods.
Which is a valid heap(min-heap) ?
55
57
63
3
7
8
9
13
41
76
81
98
16
54
19
67
87


HEAP Data Structure - Definition, Types, Applications, Implementation
1) Creating Nodes in Memory & link them in Tree Structure (min - heap)
n1 (#81)
L.C
#34
12
#50
R.C
n2 (#34)
#75
45
#21
n3 (#50)
17
n4 (#75)
63
n5 (#21)
90
2) Arranging the Heap DS elements in an Array structure (min - heap)
The representation is done as -
1) The root element will be at A [0]
2) Below table shows indexes of other nodes for the ith node i.e., A [i]:
1) A [(i-1)/2] Returns the PARENT node
2) A [(2*i)+1] Returns the LEFT child node
3) A [(2*i)+2] Returns the RIGHT child node
0
n1
16
1
n2
n3
2
54
19
n4
n5
3
67
87
4
0 1 2 3 4
A = 16 54 19 67 87


HEAP Data Structure - Definition, Types, Applications, Implementation
Operations using Heap DS -
1) getMini(): It returns the root element of Min Heap. Time Complexity of this operation is O(1).
1) getMax() : It returns the root element of Max Heap. Time Complexity of this operation is O(1).
2) extractMin(): Removes the minimum element from MinHeap.
Time Complexity of this Operation is O(Logn) as this operation needs to maintain the heap property (by calling heapify()) after removing root.
3) insert(): Inserting a new key takes O(Logn) time. We add a new key at the end of the tree.
IF new key is greater than its parent, then we don't need to do anything.
Otherwise, we need to traverse up to fix the violated heap property.
4) delete(): Deleting a key also takes O(Logn) time. We replace the key to be deleted with minium infinite by calling decreaseKey(). After decreaseKey(), the minus infinite value must reach root, so we call extractMin() to remove the key.
5) heapify(): Heapify is the process of creating a heap data structure from a binary tree.
It is used to create a Min-Heap or a Max-Heap.
16
54
19
67
87
0 1 2 3 4
A = 16 54 19 67 87

GRAPH


Graph Data Structure - Introduction
Definition -
Graph consists of a finite set of vertices(or nodes) and set of Edges (or Links) which connect a pair of nodes.
A Graph is a non-linear data structure consisting of nodes and edges.
Trees | Graphs
1. Only one path/edge between two nodes. | Multiple paths/edges/links between two nodes.
2. Has a Root Node. | No Root Node.
3. Dont have loops. | Can Have loops.
4. Have N-1 edges (N = No of nodes) | No of edges not defined.
5. Hierarchical Model | Network Model.
* A tree is an undirected graph.
Trees vs Graphs
Tree
Graph
16
54
19
67
87
16
54
19
67
87
32
2
3


Graph Data Structure - Introduction
Definition -
Graph consists of a finite set of vertices(or nodes) and set of Edges (or Links) which connect a pair of nodes.
- A Graph is a non-linear data structure consisting of nodes and edges.
Directed Graph(Digraph) -
(paths)
- A directed graph is a set of vertices (nodes) connected by edges, with each node having a direction associated with it.
Edges are usually represented by arrows pointing in the direction the graph can be traversed.
Undirected Graph -
- In an undirected graph the edges are bidirectional, with no direction associated with them. Hence, the graph can be traversed in either direction. The absence of an arrow tells us that the graph is undirected.
Directed(Digraphs) vs Undirected Graphs
16
54
19
67
87
32
Directed(Digraph)
16
54
19
67
87
32
Undirected

Graph Data Structure - Introduction

Definition -

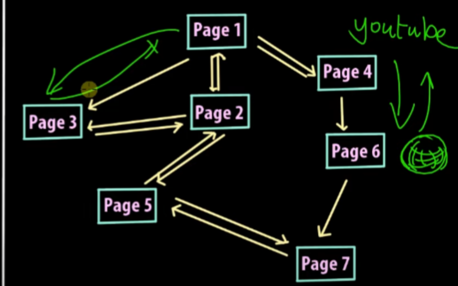
Graph consists of a finite set of vertices(or nodes) and set of Edges (or Links) which connect a pair of nodes.

A Graph is a non-linear data structure consisting of nodes and edges.

Example of Graphs -

1. *Social Network*
2. *Maps (Google Maps) / GPS / Navigation / Flight*
3. *World Wide Web (WWW)*

Unweighted, Directed



Graph Data Structure - *Adjacency Matrix*

Adjacency Matrix -

An adjacency matrix is a way of representing a graph as a matrix of booleans (0's and 1's). A finite graph can be represented in the form of a square matrix on a computer,where the boolean value of the matrix indicates if there is a direct path between two vertices.

Let's assume the n x n matrix as adj[n][n].

- if there is an edge from vertex i to j, mark adj[i][j] as 1. i.e. adj[i][j] == 1
- if there is no edge from vertex i to j, mark adj[i][j] as 0. i.e. adj[i][j] == 0

list/hash table

a = 0
b = 1
c = 2
d = 3
e = 4
f = 5

h(b) → 1 (b, e)
h(e) → 4 (1, 4)

adj =

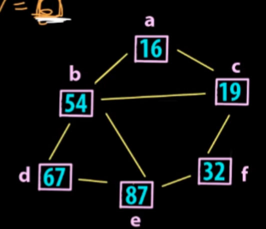
i \ j	a 0	b 1	c 2	d 3	e 4	f 5
a 0	0	1	1	0	0	0
b 1	1	0	1	1	1	0
c 2	1	1	0	0	0	1
d 3	0	1	0	0	1	0
e 4	0	1	0	1	0	1
f 5	0	0	1	0	1	0

adj[1][4] = ?1

= 36

n = V = 6

Graph



n = number of nodes in graph

Time Complexity -

1. Find Node in graph - O(1) / O(n)
2. Find all adjacent nodes of a node - O(n)
3. Check if 2 nodes are connected - O(1)

Space Complexity -

$O(n^2)$ = $(100)^2$

Graph Data Structure - *Adjacency List*

a=0, b=1, c=2, d=3, e=4, f=5

Adjacency List -

In Adjacency List, we use an array of a list to represent the graph. The size of the array is equal to the number of vertices. Let the array be an array[]. An entry array[i] represents the list of vertices adjacent to the ith vertex.

adj =

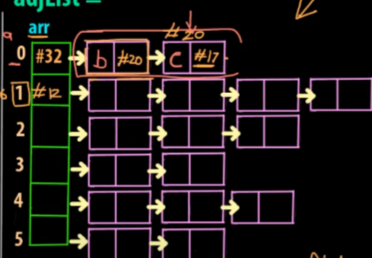
i \ j	a 0	b 1	c 2	d 3	e 4	f 5
a 0	0	1	1	0	0	0
b 1	1	0	1	1	1	0
c 2	1	1	0	0	0	1
d 3	0	1	0	0	1	0
e 4	0	1	0	1	0	1
f 5	0	0	1	0	1	0

Time Complexity -

1. Find all adjacent nodes - O(n)
2. Check if 2 nodes are connected - O(1)

Space Complexity - $O(n^2)$

adjList =




```
class Node
{
  int data;
  Node* next;

}

Node *arr[6];
```

Node arr[0] →
Node arr[1] →

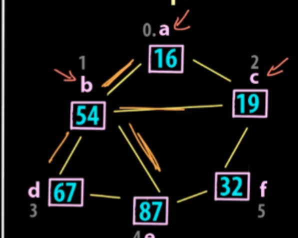
Time Complexity -

1. Find all adjacent nodes - O(n)
2. Check if 2 nodes connected - O(n)

Space Complexity - O(e)

e --> no of edges

Graph



n = number of nodes in graph