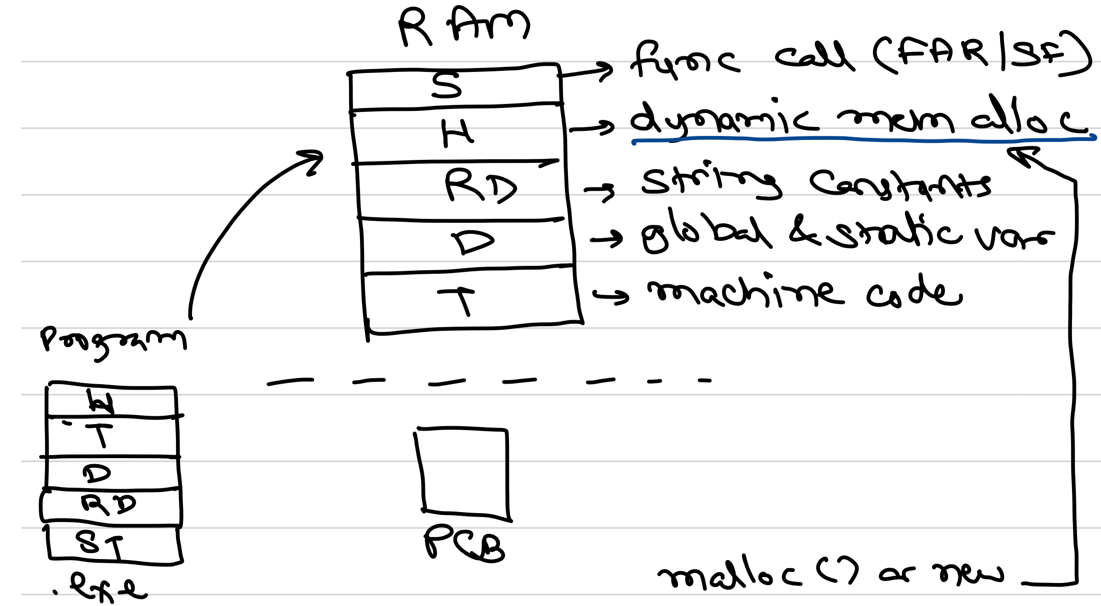
Data Structure & Algorithms

Sunbeam Infotech

process heap vs heap data structure?



Heap: Dictionary meaning
= Collection of untidy objects

Heap data struct

↓

Array impl of Complete Binary tree.

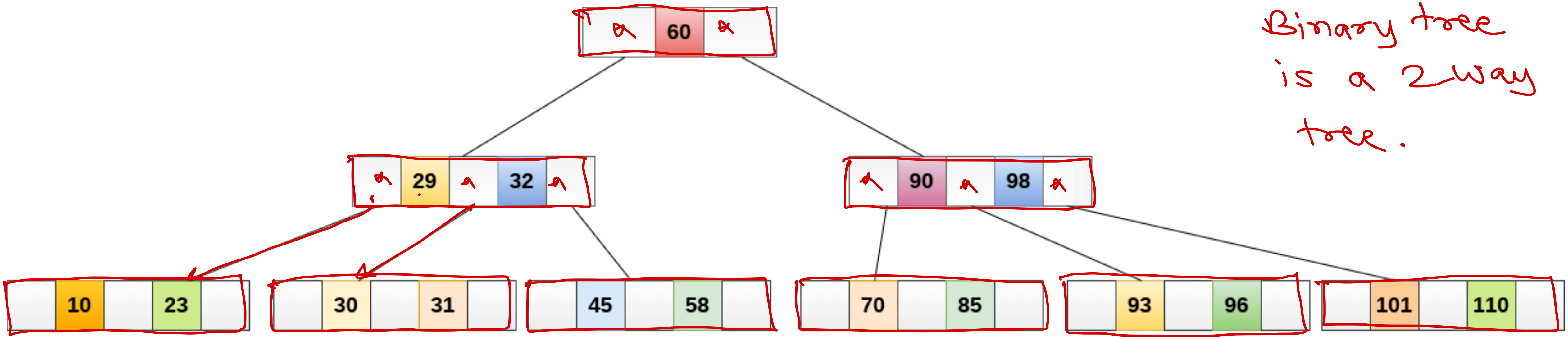
Type:

- Max Heap
- Min Heap

Agenda

- B-Tree vs B+ Tree
- Graph Introduction
- Graph terminologies
- Spanning trees
- Forest
- Graph types
- Graph implementation
  - Adjacency matrix
  - Adjacency list
- BFS & DFS Traversal

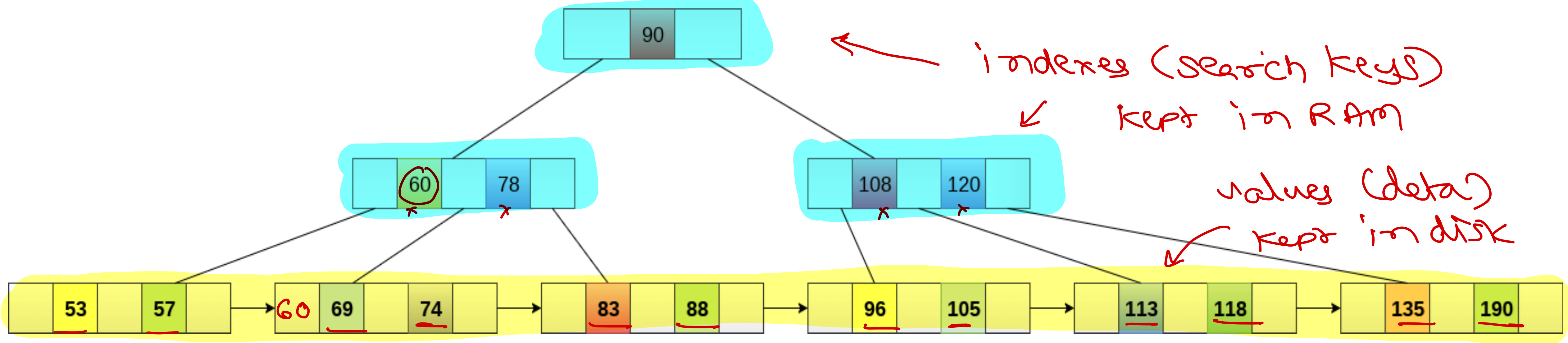
B Tree – specialization of m-way tree (tree with degree m).



Binary tree is a 2-way tree.

- A B-Tree of order m can have at most m-1 keys and m children.
- B tree store large number of keys in a single node. This allows storing number of values keeping height minimal.
- Note that in B-Tree all leaf nodes are at same level.
- B-Tree is commonly used for indexing into file systems and databases. It ensures quick data searching and speed up disk access.

B+ Tree



- Extension of B-Tree for efficient insert, delete and search operation.
- Data is stored in leaf nodes only and all leaf nodes are linked together for sequential access.
- Search keys may be redundant.
- Faster searching, simplified deletion (as only from leaf nodes).
- B+Tree is commonly used for indexing into file systems and databases. It ensures quick data searching and speed up disk access.

NTFS

Dictionary = Hash Map = map = Hash Table

Array (Table).

Open addressing

Chaining

bucket → linked list or BST or AVR tree

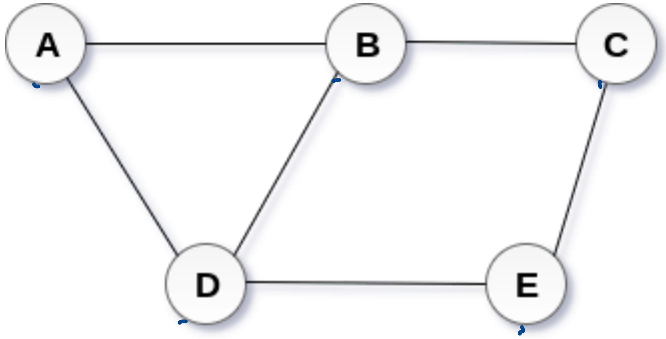
or B Tree

Graph

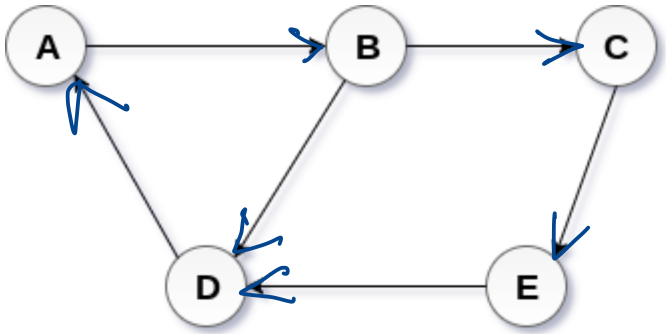
- Graph is a non-linear data structure.
- Graph is defined as set of vertices and edges. Vertices (also called as nodes) hold data, while edges connect vertices and represent relations between them.
  - G = { V, E }
  - V = { A, B, C, D, E }
- Graph edges may or may not have directions.
- Undirected edges
  - E = { (A,B), (A,D), (B,C), (B,D), (C,E), (D,E) }
- Directed edges
  - E = { <A,B>, <B,C>, <B,D>, <C,E>, <D,A>, <E,D> }
- When there is an edge from vertex P to vertex Q, P is said to be adjacent to Q.

P adjacent Q
Q adjacent P

P adjacent to Q



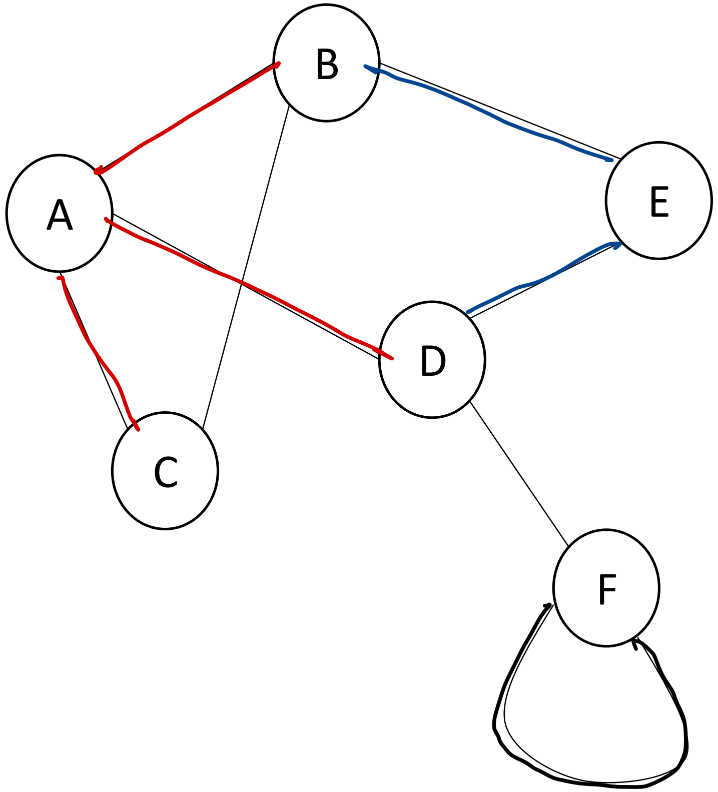
Undirected Graph



Directed Graph

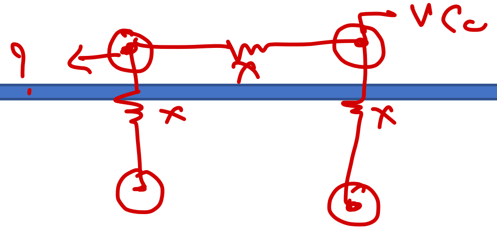
Graph terminologies

- Degree of node: Number of nodes adjacent to the node.
  - Degree of A is 3
  - Degree of E is 2
- Degree of graph: Maximum degree of any node in graph. 3
- Path: Set of edges between two vertices. There can be multiple paths between two vertices.
  - A – D – E
  - A – B – E
  - A – C – B – E
- Cycle: Path whose start and end vertex is same.
  - A – B – C – A
  - A – B – E – D – A
- Loop: Edge connecting vertex to itself. It is smallest cycle.
  - F — F



Graph Types

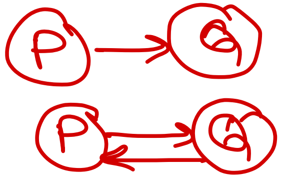
NoSQL db → Neo4J



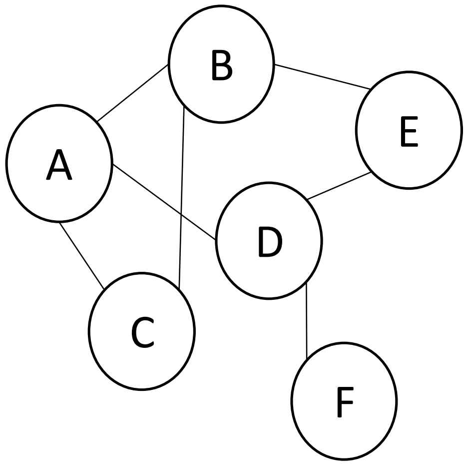
- Un-directed graph
  - Graph edges do not have directions.
  - If P is adjacent to Q, Q is adjacent to P.

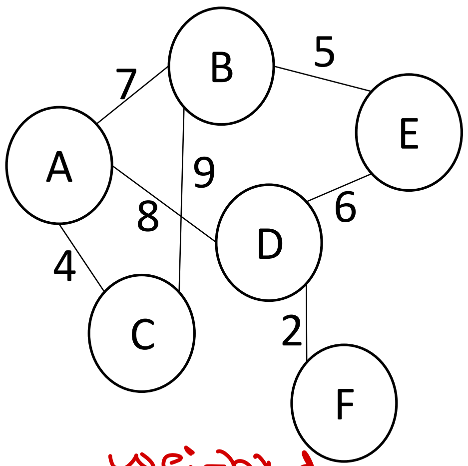
(P, Q)

- Directed graph (Di-graph)
  - Graph edges have direction.
  - If P is adjacent to Q, Q may not be adjacent to P.
  - Out-degree: Number of edges originated from the node
  - In-degree: Number of edges terminated on the node



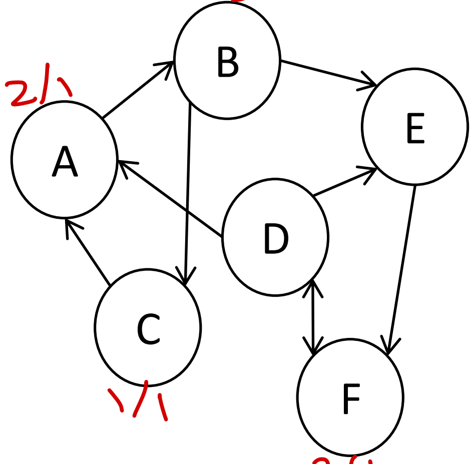
- Weighted graph
  - Graph edges have weight associated with them.
  - Weight represent some value e.g. distance, resistance.
- Directed Weighted graph (Network)
  - Graph edges have directions as well as weights.

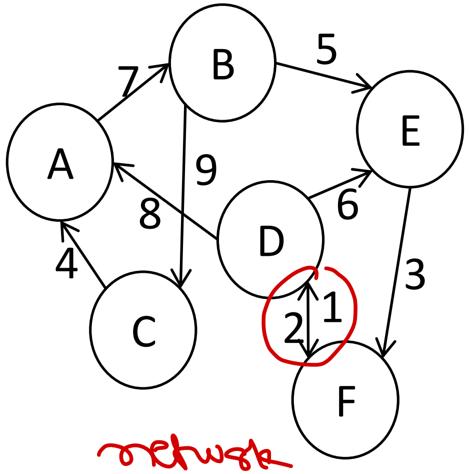




weighted graph

di-graph

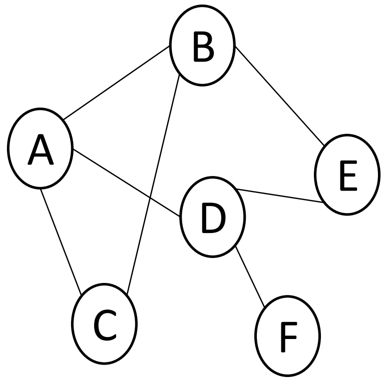


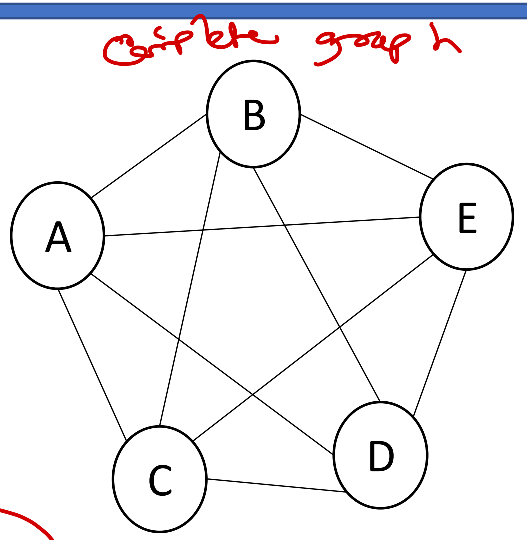


network

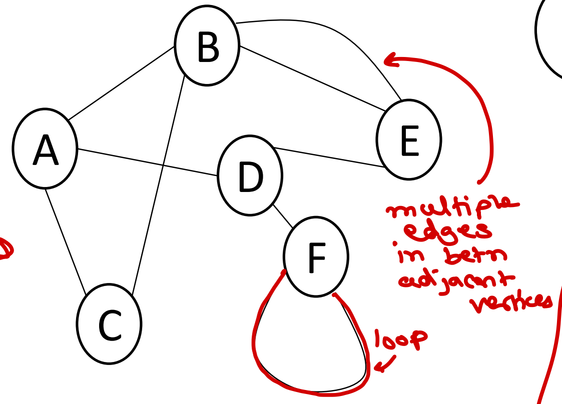
Graph types

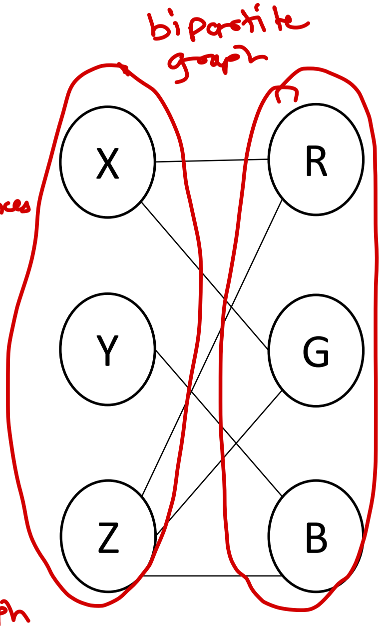
- Simple graph
  - Doesn't have multiple edges in adjacent vertices ~~or~~ & loops.
- Connected graph
  - From each vertex some path exists for every other vertex.
  - Can traverse the entire graph starting from any vertex.
- Complete graph
  - Each vertex of a graph is adjacent to every other vertex. direct edge
  - For un-directed graph
    - Number of edges = n (n-1) / 2 $= 5*4/2 = 10$
  - For directed graph
    - Number of edges = n (n-1)
- Bi-partite graph
  - Vertices can be divided in two disjoint sets.
  - Vertices in first set are connected to vertices in second set.
  - Vertices in a set are not directly connected to each other.

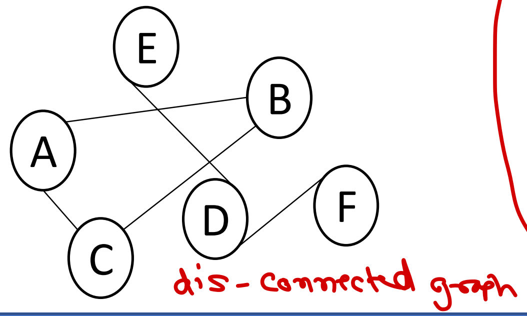




not simple graph



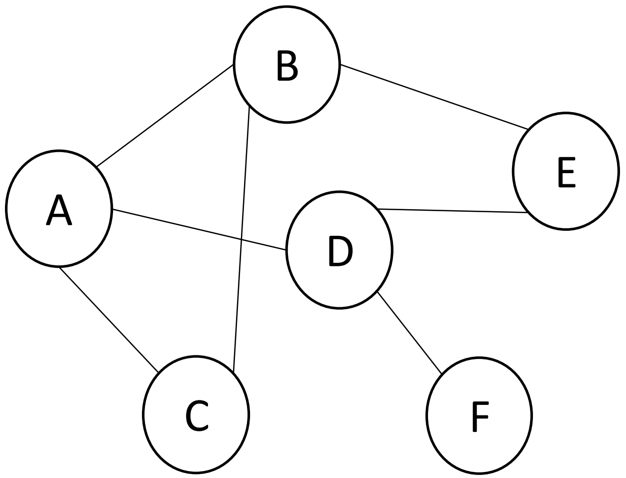


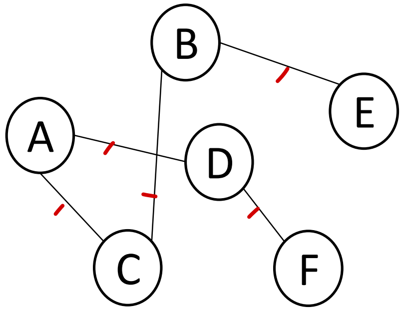


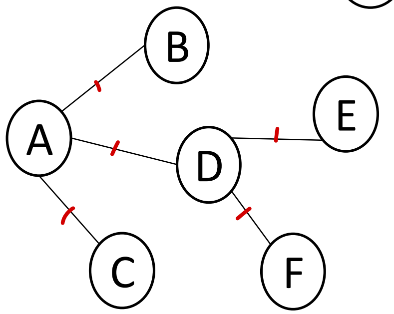
Spanning Tree

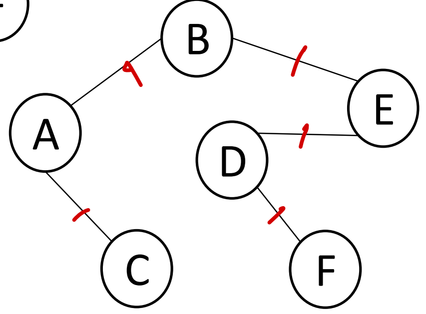
- Tree is a graph without cycles.
- Spanning tree is a connected sub-graph of the given graph that contains all the vertices and sub-set of edges.
- Spanning tree can be created by removing few edges from the graph which are causing cycles ~~to form~~.
- One graph can have multiple different spanning trees.
- In weighted graph, spanning tree can be made who has minimum weight (sum of weights of edges). Such spanning tree is called as Minimum Spanning Tree.
- Spanning tree can be made by various algorithms.
    - BFS Spanning tree
    - DFS Spanning tree
    - Prim's MST
    - Kruskal's MST

* Spanning tree contains V-1 edges.









Graph implementation

- Adjacency

Matrix — List

2-D array — Linked List

```
int mat[V][V];
```

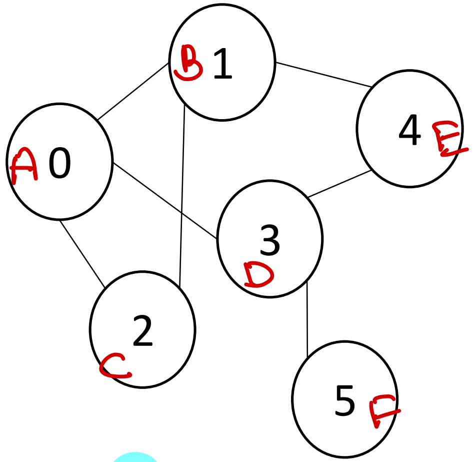
```
list<int> arr[V];
```

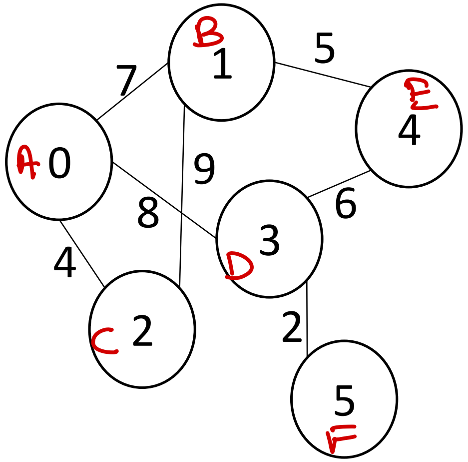
```
vector<vector<int>> mat;
```

```
vector<list<int>> arr;
```

Graph Implementation – Adjacency Matrix

- If graph have V vertices, a V x V matrix can be formed to store edges of the graph.
- Each matrix element represent presence or absence of the edge between vertices.
- For non-weighted graph, 1 indicate edge and 0 indicate no edge.
- For weighted graph, weight value indicate the edge and infinity sign ∞ represent no edge.
- For un-directed graph, adjacency matrix is always symmetric across the diagonal.
- Space complexity of this implementation is $O(V^2)$.





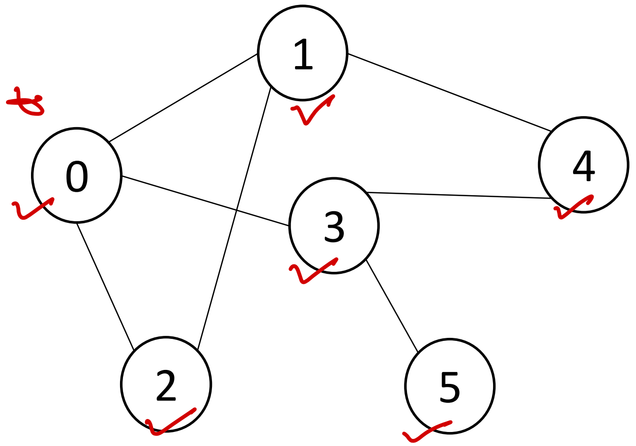
		0	1	2	3	4	5
		A	B	C	D	E	F
0	A	0	1	1	1	0	0
1	B	1	0	1	0	1	0
2	C	1	1	0	0	0	0
3	D	1	0	0	0	1	1
4	E	0	1	0	1	0	0
5	F	0	0	0	1	0	0

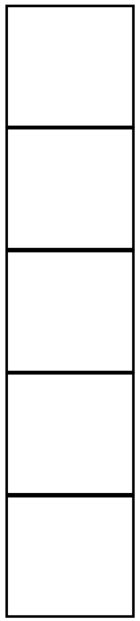
	A	B	C	D	E	F
A	∞	7	4	8	∞	∞
B	7	∞	9	∞	5	∞
C	4	9	∞	∞	∞	∞
D	8	∞	∞	∞	6	2
E	∞	5	∞	6	∞	∞
F	∞	∞	∞	2	∞	∞

Graph Traversal – BFS & DFS

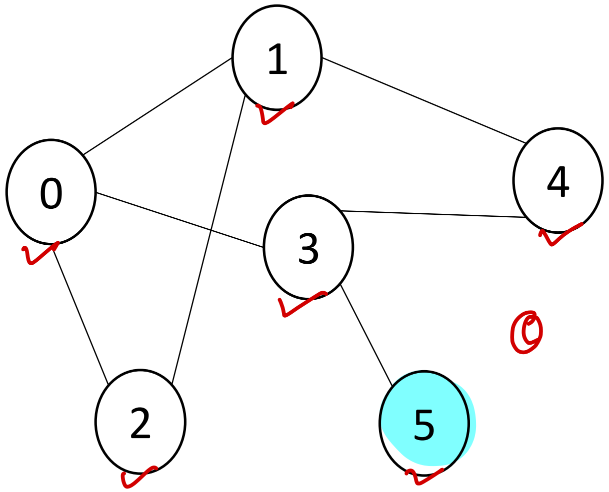
0 3 5 4 2 1

- DFS algorithm.
  1. • Choose a vertex as start vertex. = 0
  2. • Push start vertex on stack. & mark as visited
  3. • Pop vertex from stack.
  4. • Print the vertex.
  5. • Put all non-visited neighbours of the vertex on the stack and mark them as visited.
  6. repeat 3-5 until stack is empty.

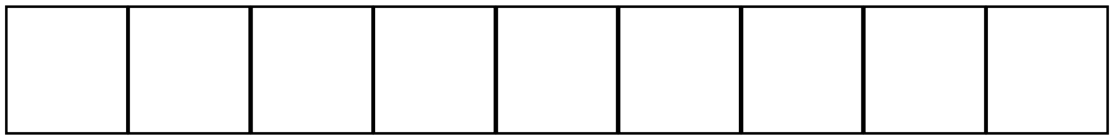




- BFS algorithm.
  1. • Choose a vertex as start vertex.
  2. • Push start vertex on queue. & mark as visited
  3. • Pop vertex from queue.
  4. • Print the vertex.
  5. • Put all non-visited neighbours of the vertex on the ~~stack~~ queue and mark them as visited.
  6. repeat 3-5 until queue is empty



0 1 2 3 4 5



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>