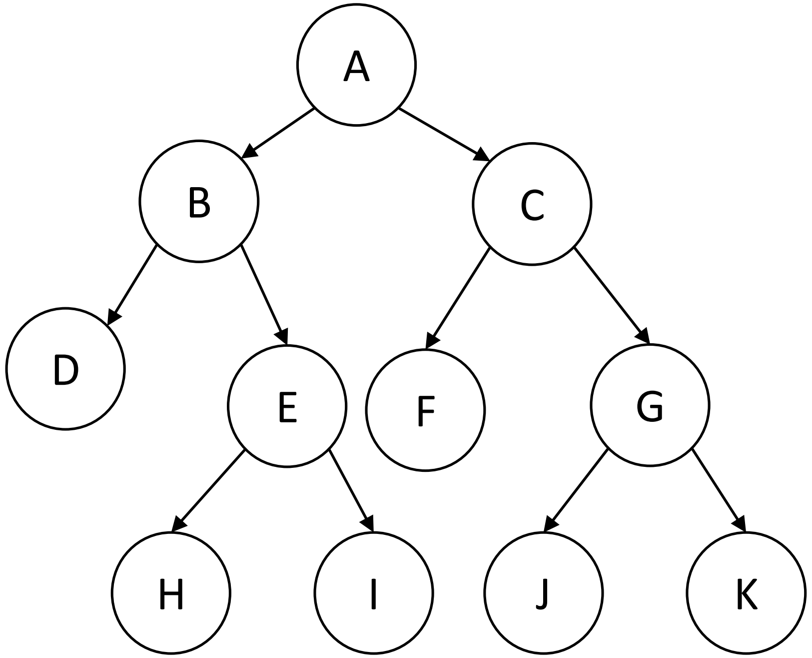
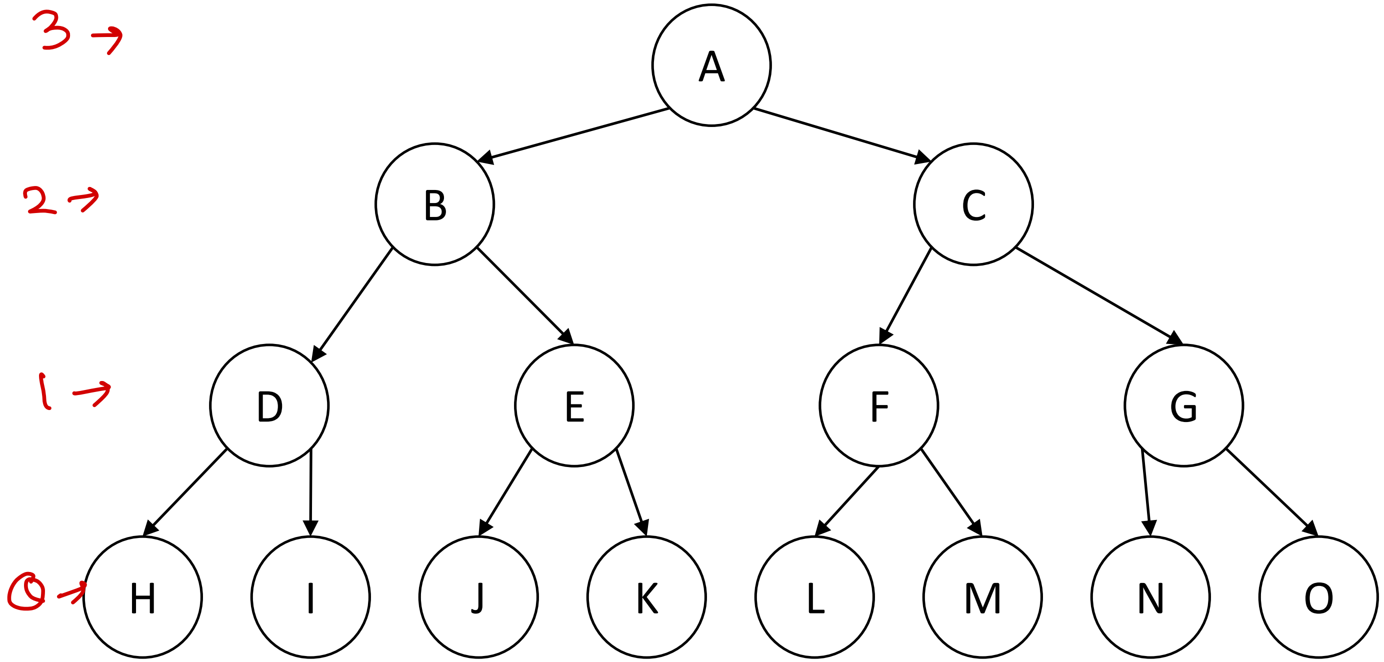
Data Structure & Algorithms

Nilesh Ghule

Strict/Full Binary Tree

Perfect Binary Tree





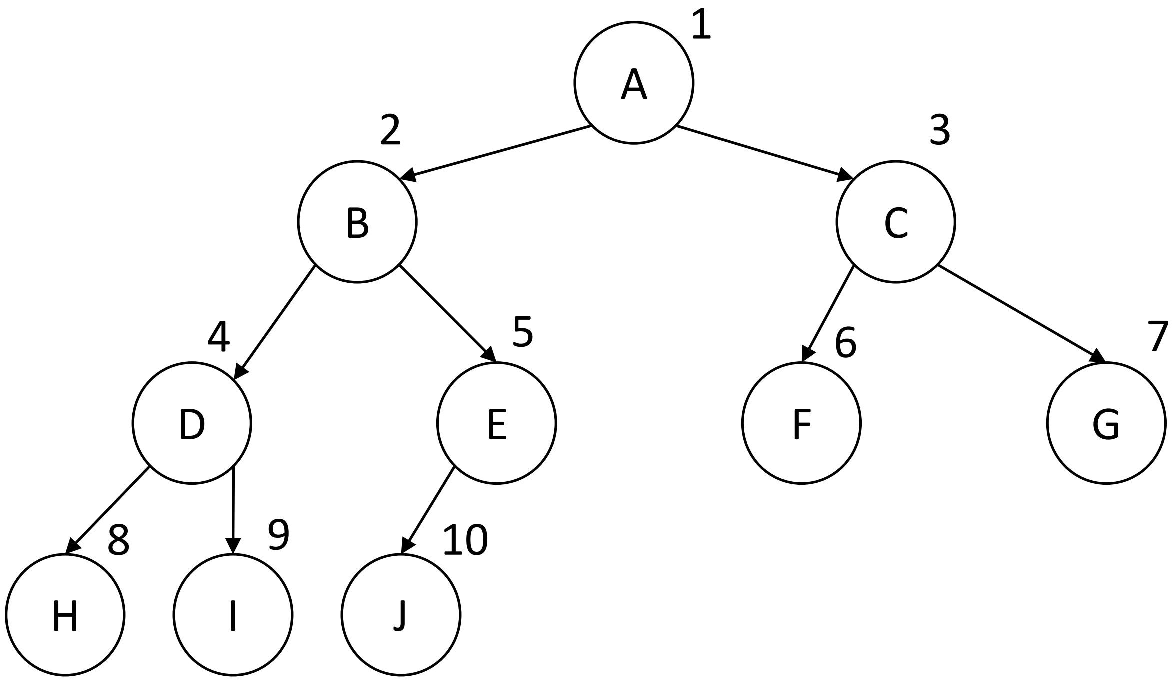
- Binary tree in which each non-leaf node has exactly two child nodes.

- Binary tree which is full for the given height i.e. contains maximum possible nodes.
- Number of nodes = ~~$2^h – 1$~~ $2^{(h+1)} - 1$ (15)

leaf nodes = 2^h (8)

non-leaf nodes = $2^h - 1$ (7)

Complete Binary Tree and Heap



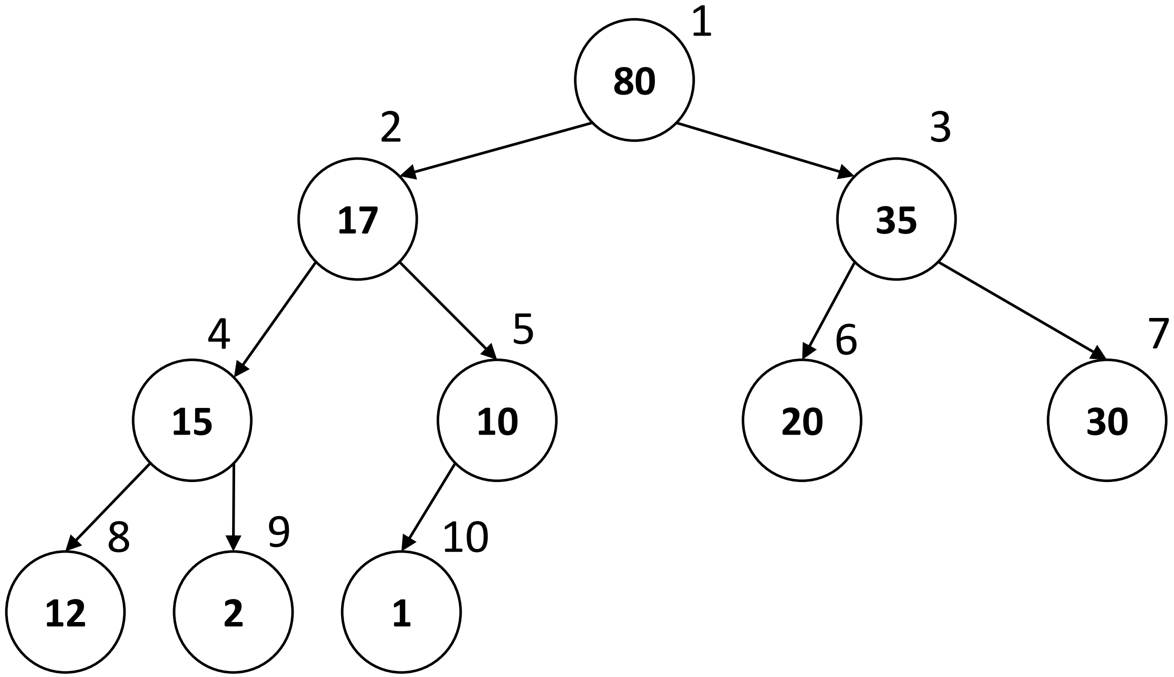
A	B	C	D	E	F	G	H	I	J
1	2	3	4	5	6	7	8	9	10
		P	P=C/2		L=2*P	R=2*P+1		C	

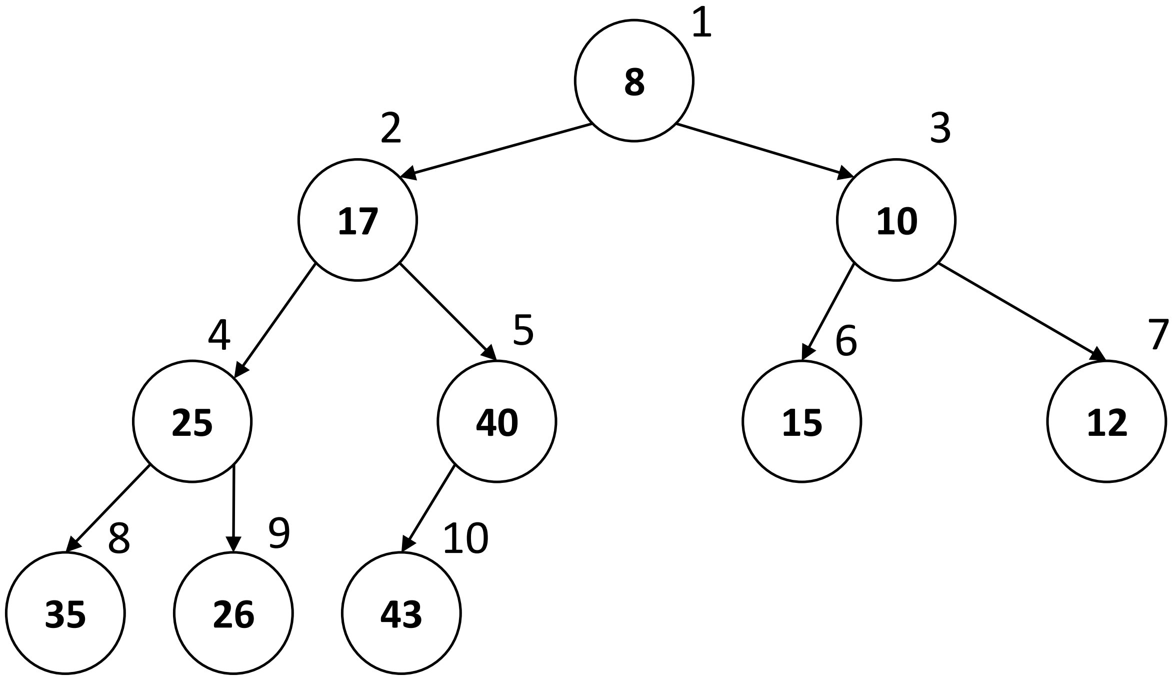
- A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible. in last level.

- Heap is array implementation of complete binary tree.
- Parent child relation is maintained through index calculations
  - parent index = child index / 2
  - left child index = parent index * 2
  - right child index = parent index * 2 + 1

Max Heap & Min Heap

- used to implement priority queue.
push/pop → O(log n).



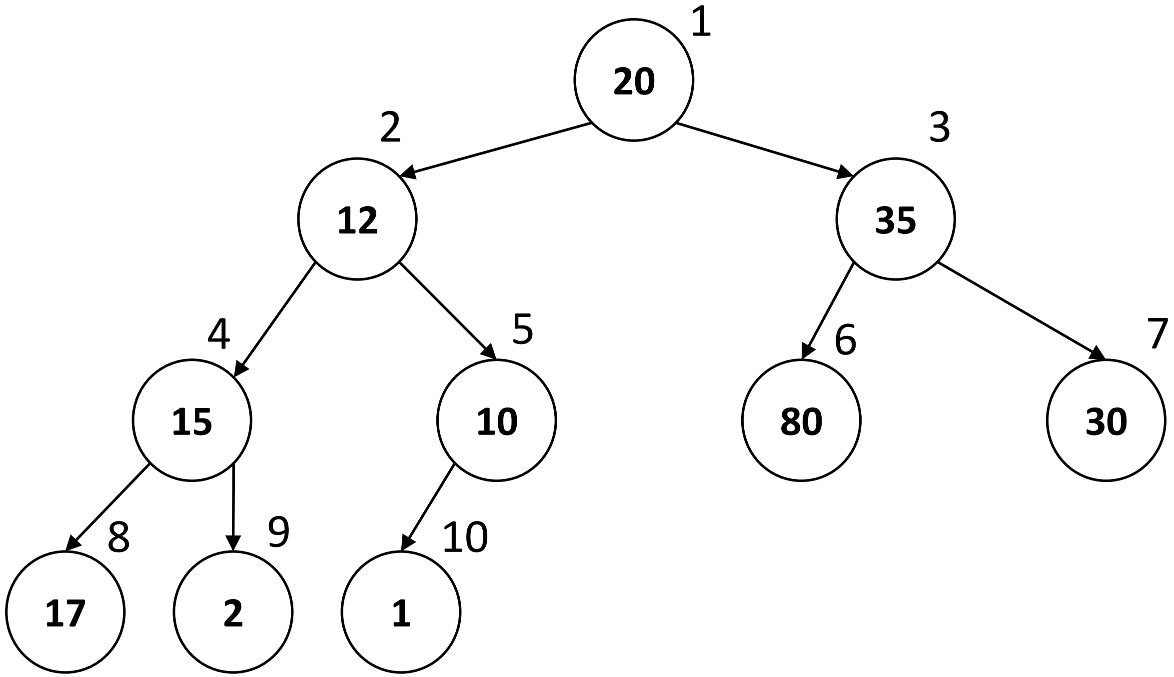


- Max heap is a heap data structure in which each node is greater than both of its child nodes.

Min

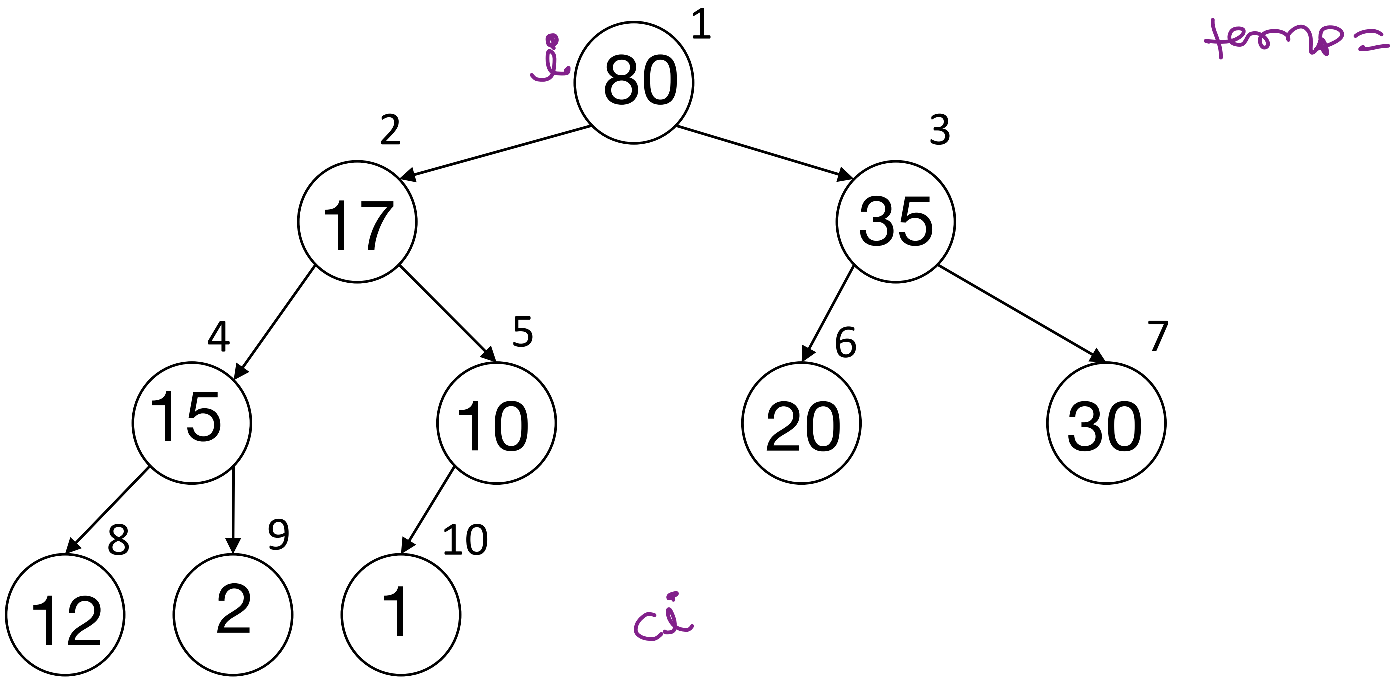
- ~~Max~~ heap is a heap data structure in which each node is smaller than both of its child nodes.

Make Heap



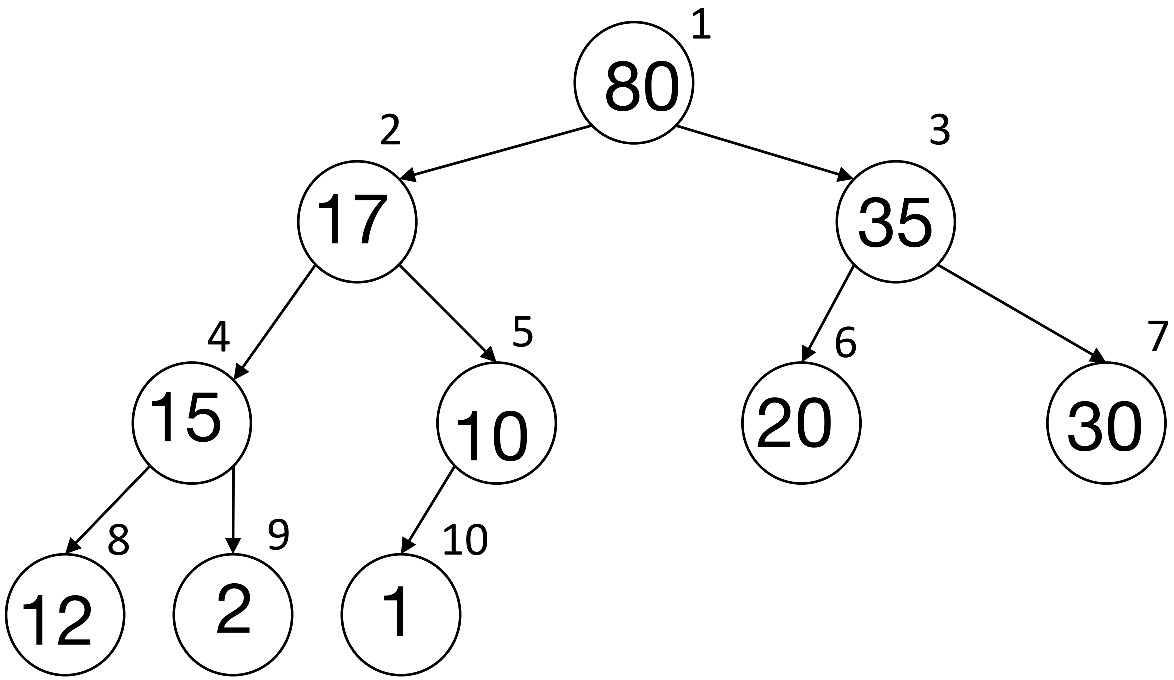
20	12	35	15	10	80	30	17	2	1
1	2	3	4	5	6	7	8	9	10

Max Heap – Initialize



80	17	35	15	10	20	30	12	2	1
1	2	3	4	5	6	7	8	9	10

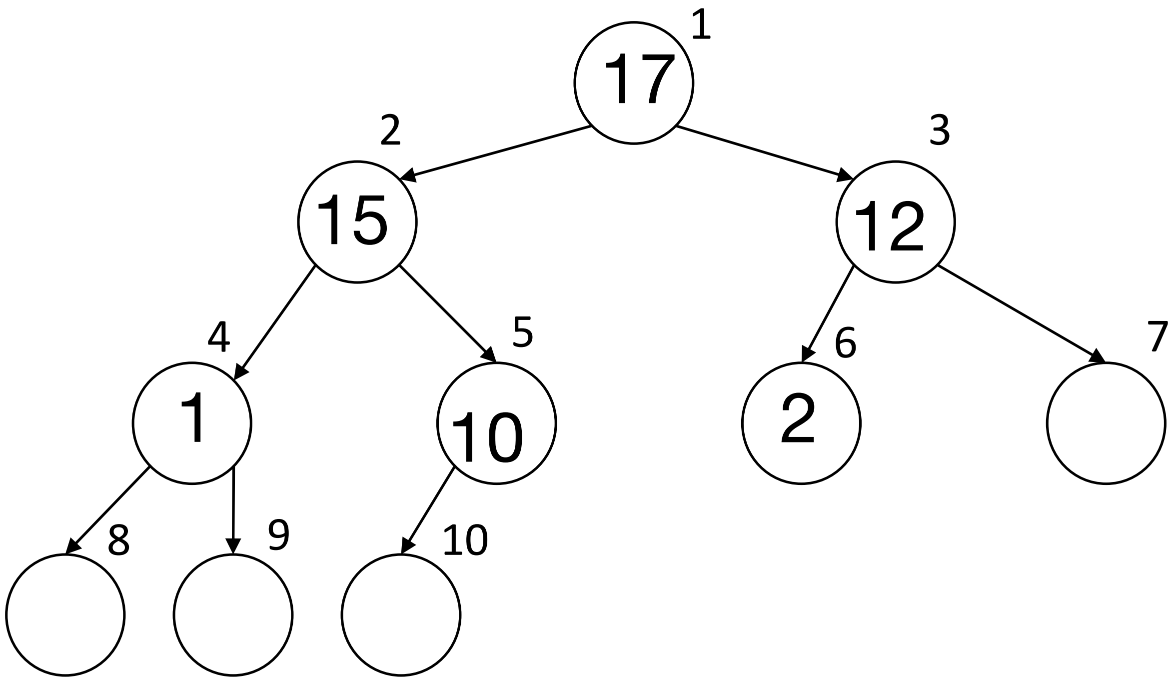
Max Heap – Initialize



80	17	35	15	10	20	30	12	2	1
1	2	3	4	5	6	7	8	9	10

~~Max Heap – Initialize~~

Find k^{th} highest ele from array.



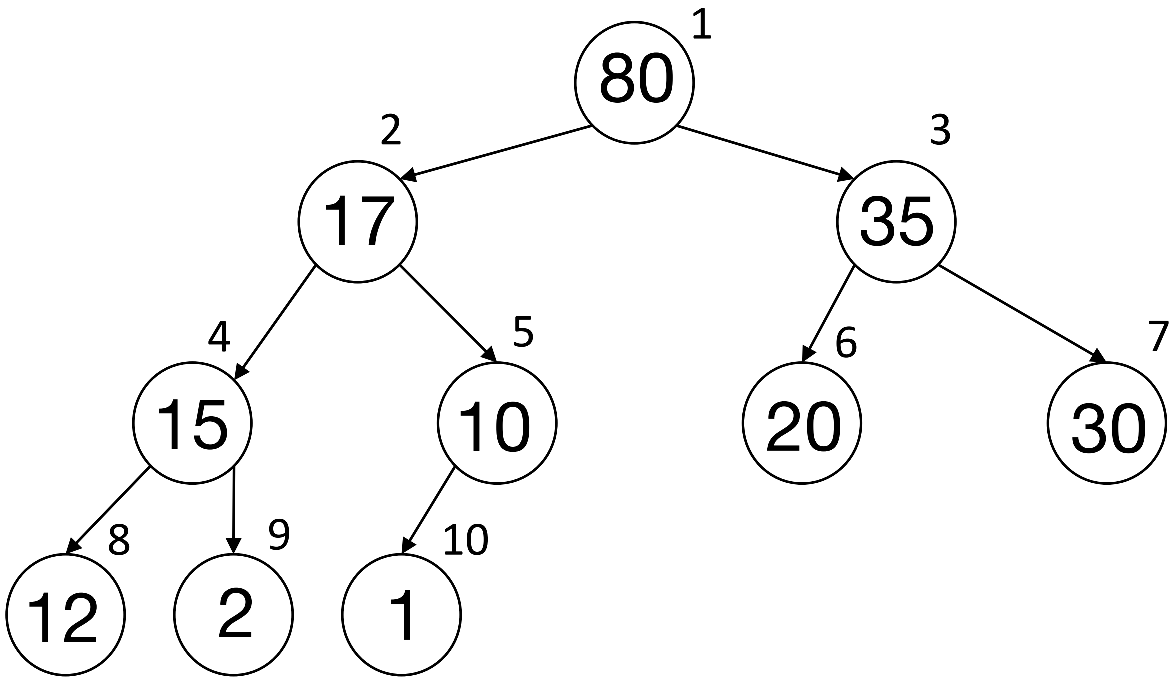
① Convert array into maxheap.

② delete max ele one by one for k times → $O(k \log n)$

80 35 30 20

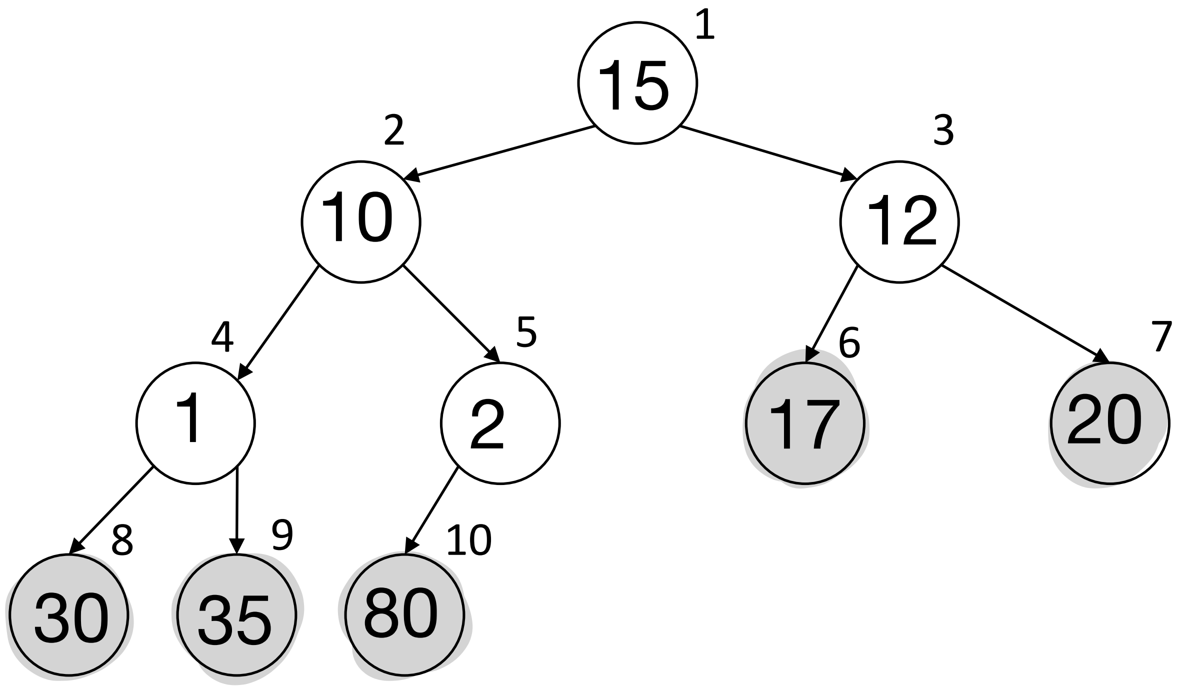
80	17	35	15	10	20	30	12	2	1
1	2	3	4	5	6	7	8	9	10

Max Heap – Delete Element



80	17	35	15	10	20	30	12	2	1
1	2	3	4	5	6	7	8	9	10

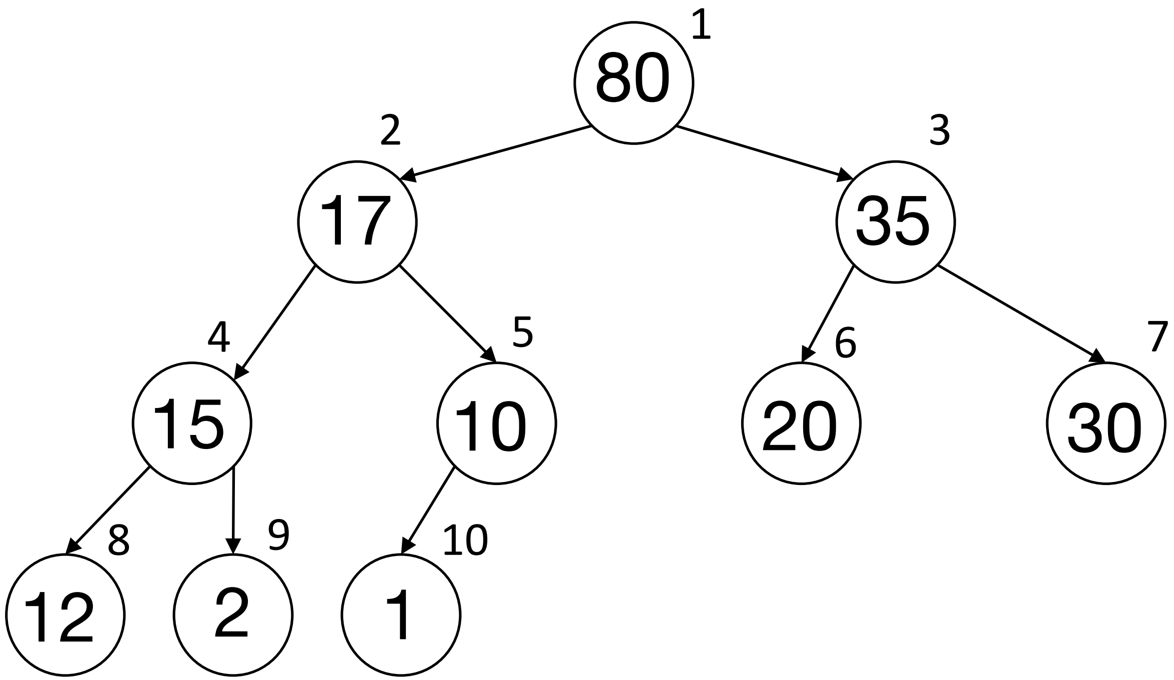
Heap Sort



15	10	12	1	2	17	20	30	35	80
1	2	3	4	5	6	7	8	9	10

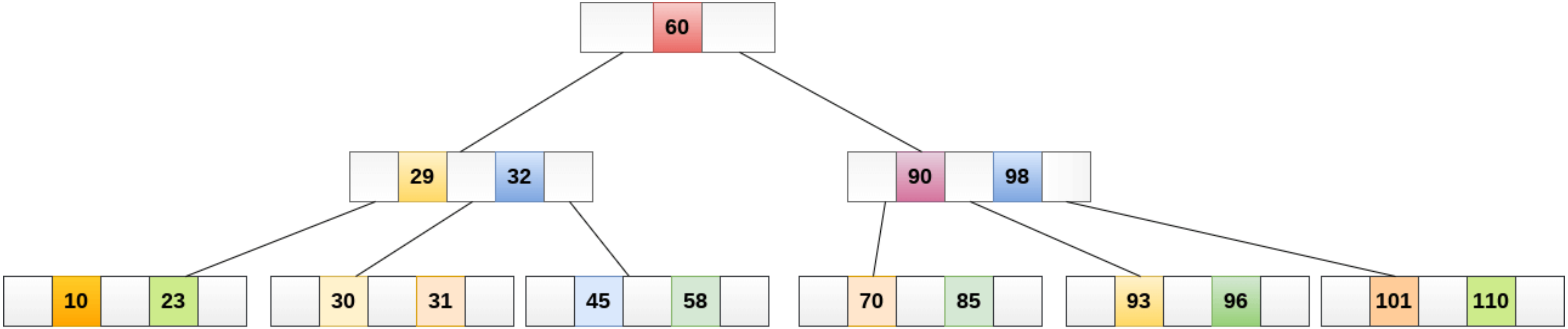
1. make max heap
2. delete max from heap & add to end of array
3. repeate step 2 until heap is empty.

Heap Sort



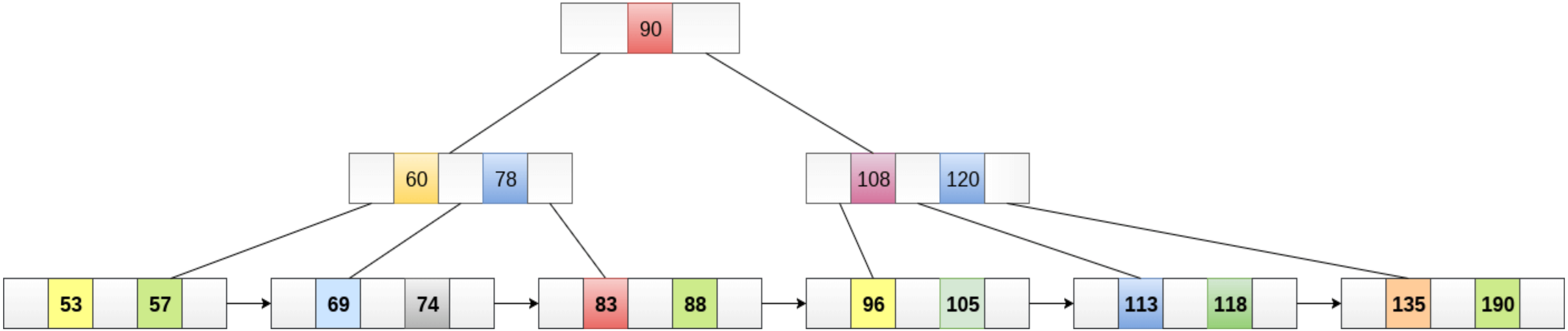
80	17	35	15	10	20	30	12	2	1
1	2	3	4	5	6	7	8	9	10

B 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.

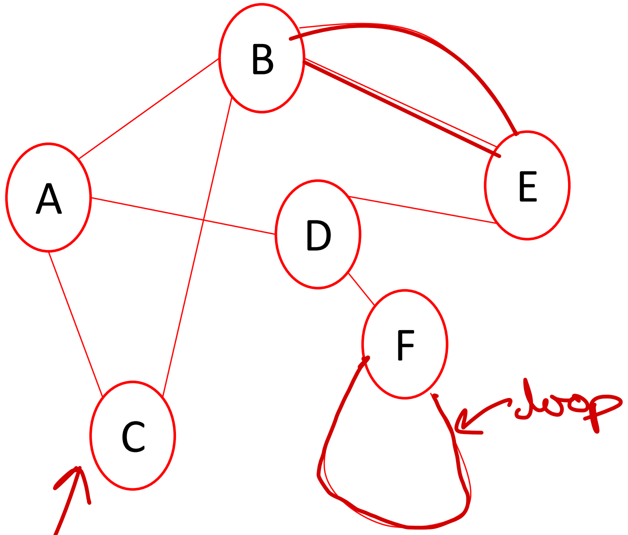
B+ Tree

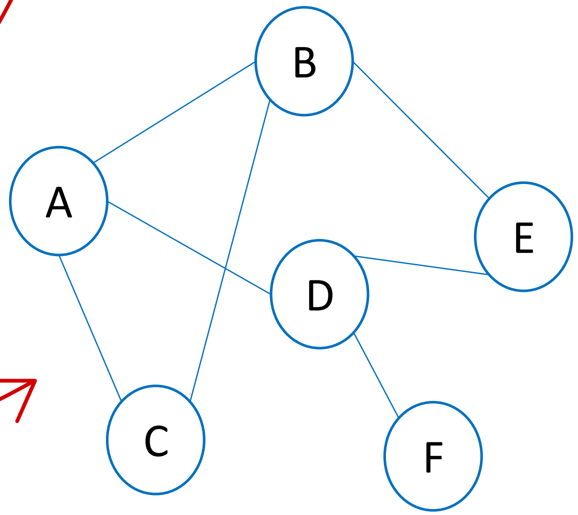


- Extension of B-Tree for efficient insert, delete and search operation.
- Data is stored in leaf nodes only and all leaf nodes are linked together for sequential access.
- Search keys may be redundant.
- Faster searching, simplified deletion (as only from leaf nodes).
- B+Tree is commonly used for indexing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

Graph

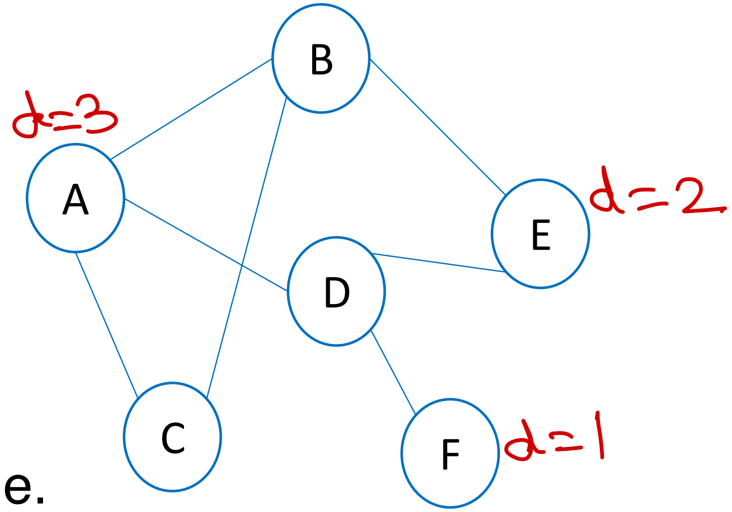
- Graph is a non-linear data structure.
- Graph is defined as set of vertices and edges. Vertices (also called as nodes) hold data, while edges connect vertices and represent relations between them.
  - G = { V, E }
- Vertices hold the data and Edges represents relation between vertices.
- When there is an edge from vertex P to vertex Q, P is said to be adjacent to Q.
- Multi-graph
  - Contains multiple edges in adjacent vertices or loops (edge connecting a vertex to it-self).
- Simple graph
  - Doesn't contain multiple edges in adjacent vertices or loops.



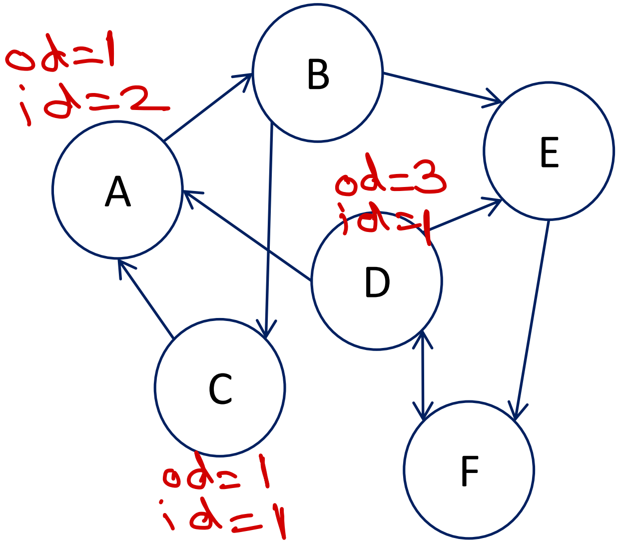


Graph

- Graph edges may or may not have directions.



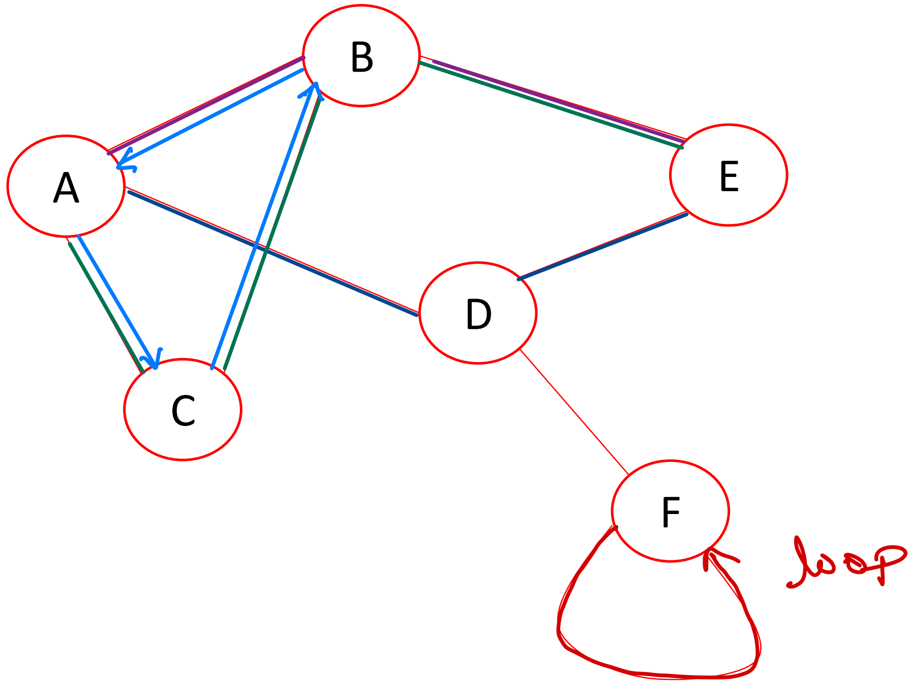
- Undirected Graph: G = { V, E }
  - V = { A, B, C, D, E, F}
  - E = { (A,B), (A,C), (A,D), (B,C), (B,E), (D,E), (D,F) }
  - If P is adjacent to Q, then Q is also adjacent to P.
  - Degree of node: Number of nodes adjacent to the node.
  - Degree of graph: Maximum degree of any node in graph.



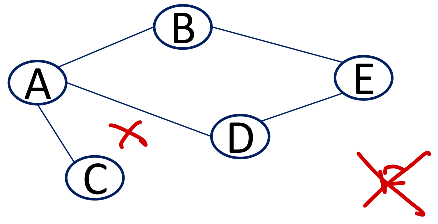
- Directed Graph: G = { V, E }
  - V = { A, B, C, D, E, F}
  - E = {<A,B>, <B,C>, <B,E>, <C,A>, <D,A>, <D,E>, <D,F>, <E,F>, <F,D>}
  - If P is adjacent to Q, then Q is may or may not be adjacent to P.
  - Out-degree: Number of edges originated from the node
  - In-degree: Number of edges terminated on the node

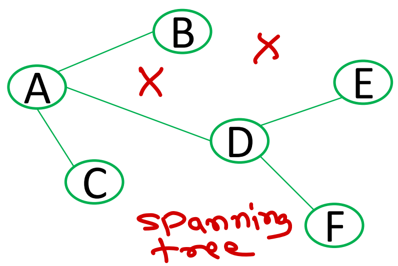
Graph

- 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



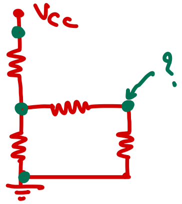
- Sub-Graph: A graph having few vertices and few edges in the given graph, is said to be sub-graph of given graph.

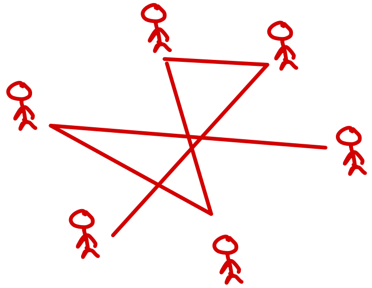


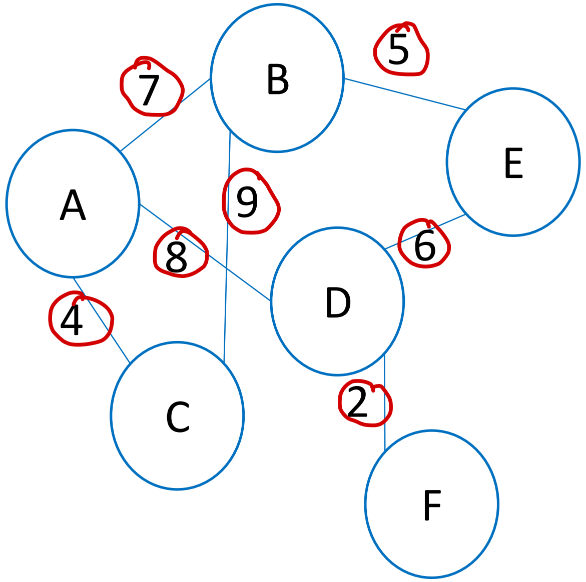


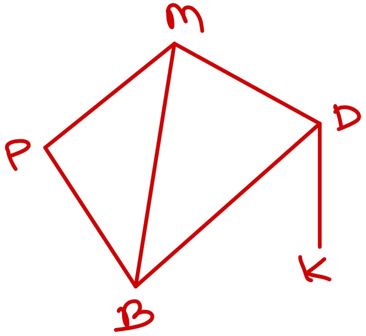
Graph

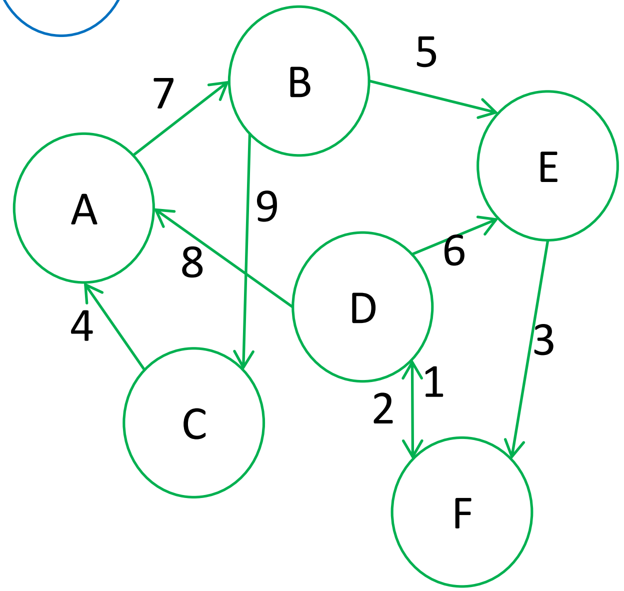
- Weighted graph
  - Graph edges have weight associated with them.
  - Weight represent some value e.g. distance, resistance.
- Directed Weighted graph (Network)
  - Graph edges have directions as well as weights.
- Applications of graph
  - Electronic circuits
  - Social media
  - Communication network
  - Road network
  - Flight/Train/Bus services
  - Bio-logical & Chemical experiments
  - Deep learning (Neural network, Tensor flow)
  - Graph databases (Neo4j)





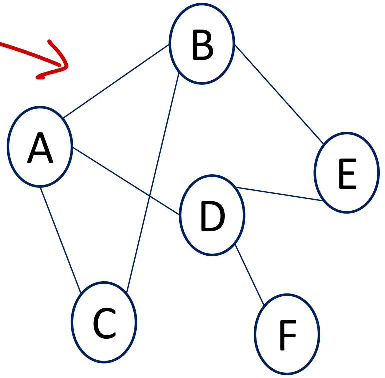


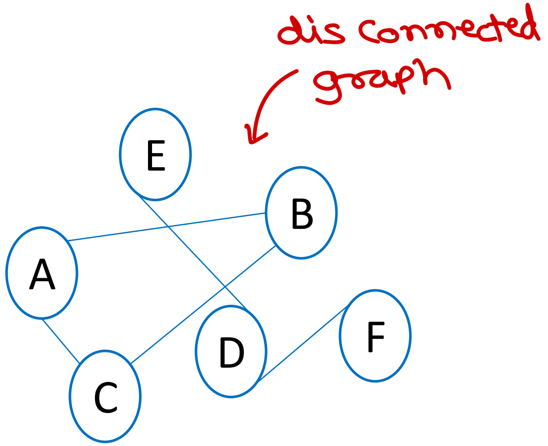




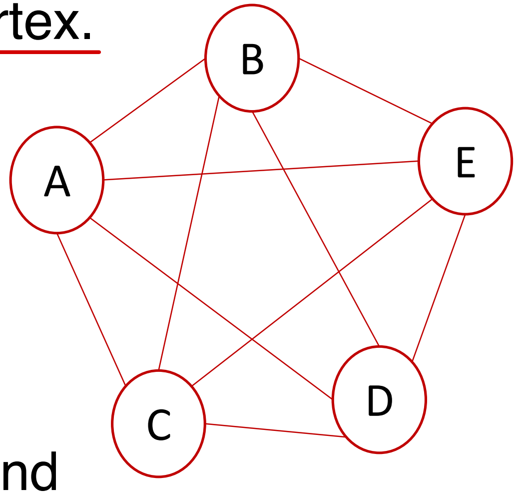
Graph

- Connected graph
  - From each vertex some path exists for every other vertex.
  - Can traverse the entire graph starting from any vertex.

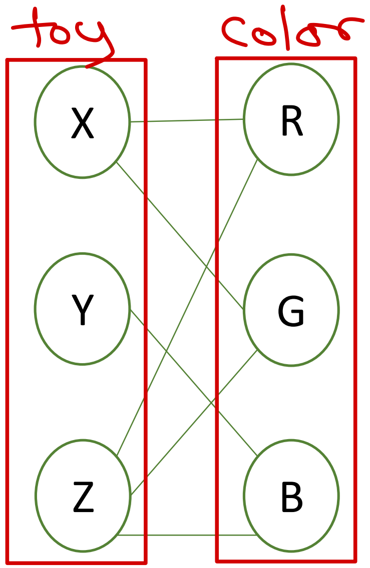




- Complete graph
  - Each vertex of a graph is adjacent to every other vertex.
  - Un-directed graph: Number of edges = n (n-1) / 2
  - Directed graph: Number of edges = n (n-1)



- 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.



Spanning Tree

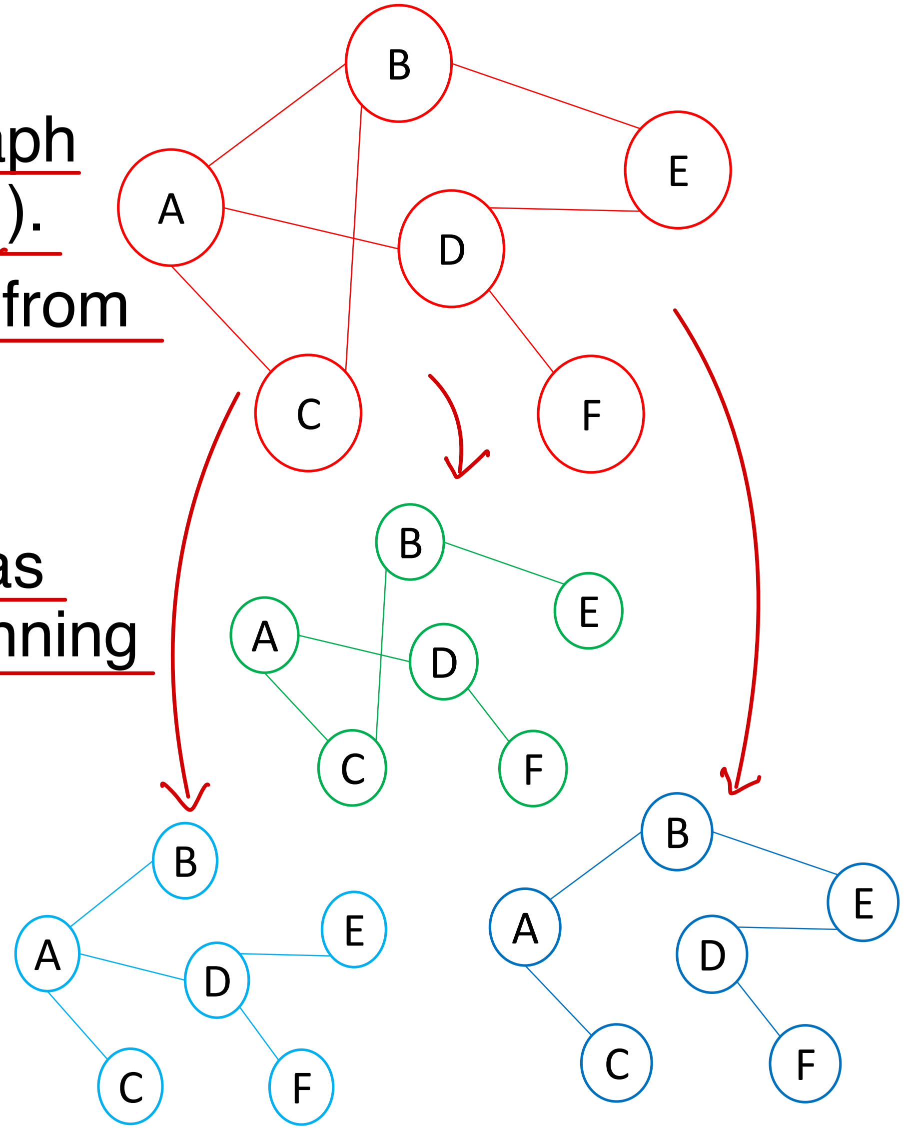
- Tree is a graph without cycles.
- Spanning tree is connected sub-graph of the given graph that contains all the vertices and sub-set of edges (V-1).
- Spanning tree can be created by removing few edges from the graph which are causing cycles to form.
- One graph can have multiple different spanning 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. (mst)
- Spanning tree can be made by various algorithms.
  - BFS Spanning tree
  - DFS Spanning tree

  spanning tree

  - Prim's MST
  - Kruskal's MST

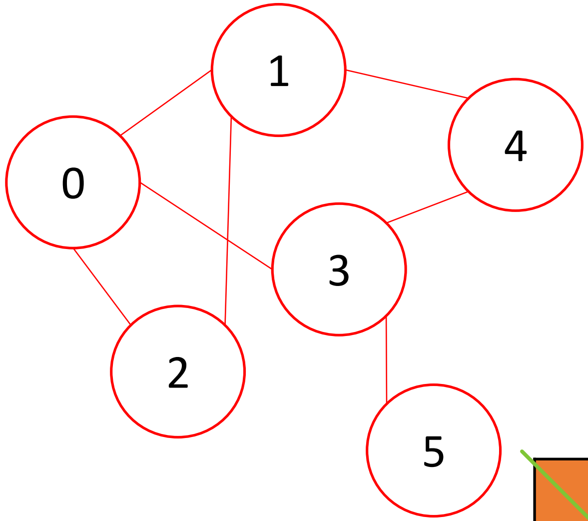
  min spanning tree

  greedy also



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 un-directed graph, adjacency matrix is always symmetric across the diagonal.
- Space complexity of this implementation is $O(V^2)$.



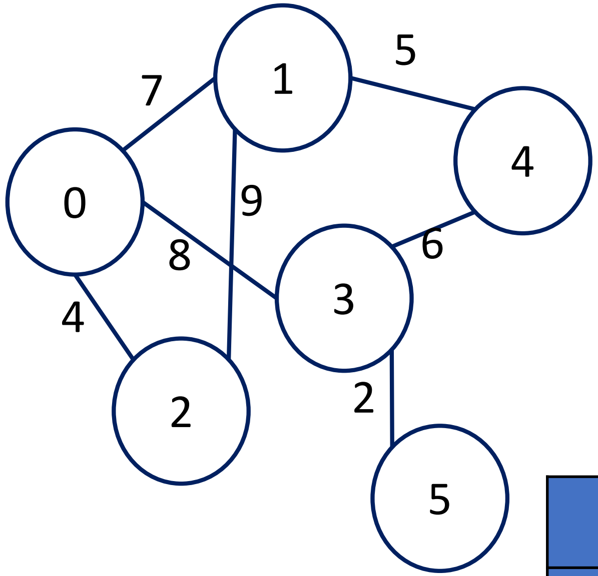
← dest

	0	1	2	3	4	5
0	0	1	1	1	0	0
1	1	0	1	0	1	0
2	1	1	0	0	0	0
3	1	0	0	0	1	1
4	0	1	0	1	0	0
5	0	0	0	1	0	0

↑ src

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 *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 $O(V^2)$.



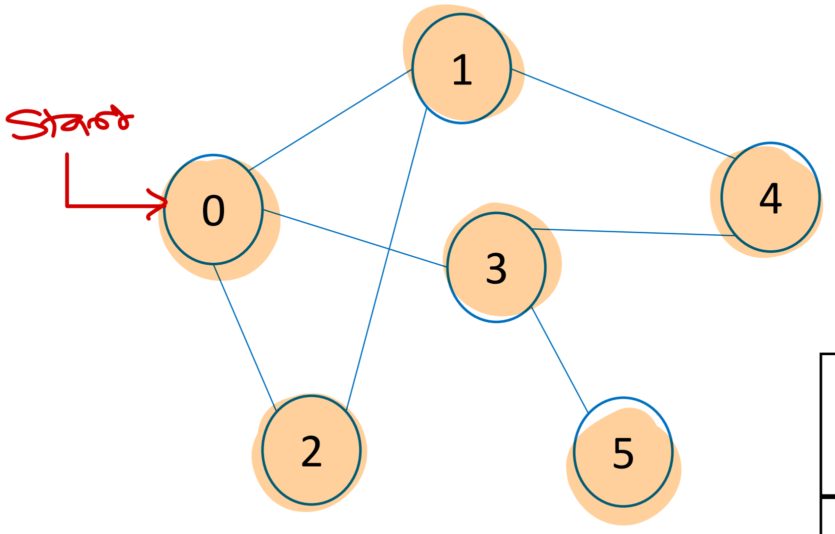
← d

	0	1	2	3	4	5
0	∞	7	4	8	∞	∞
1	7	∞	9	∞	5	∞
2	4	9	∞	∞	∞	∞
3	8	∞	∞	∞	6	2
4	∞	5	∞	6	∞	∞
5	∞	∞	∞	2	∞	∞

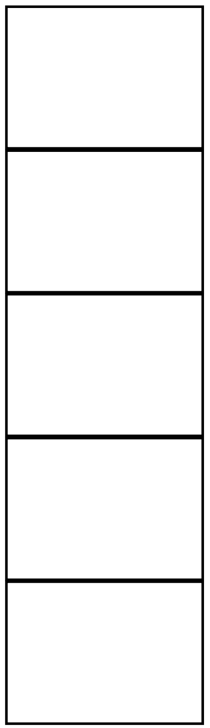
S ↑

Graph Traversal – DFS Algorithm

1. Choose a vertex as start vertex.
2. Push start vertex on stack & mark it.
3. Pop vertex from stack.
4. Visit (Print) the vertex.
5. Put all non-~~visited~~ marked neighbours of the vertex on the stack and mark them.
6. Repeat 3-5 until stack is empty.



0 3 5 4 2 1

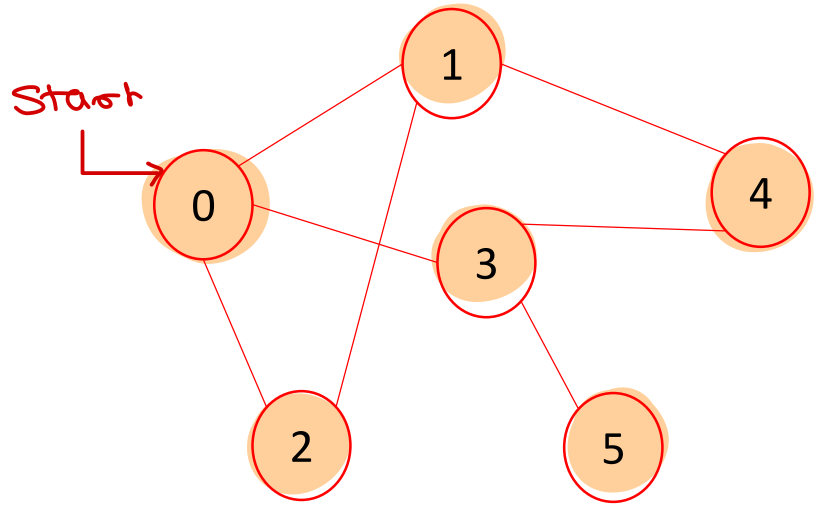


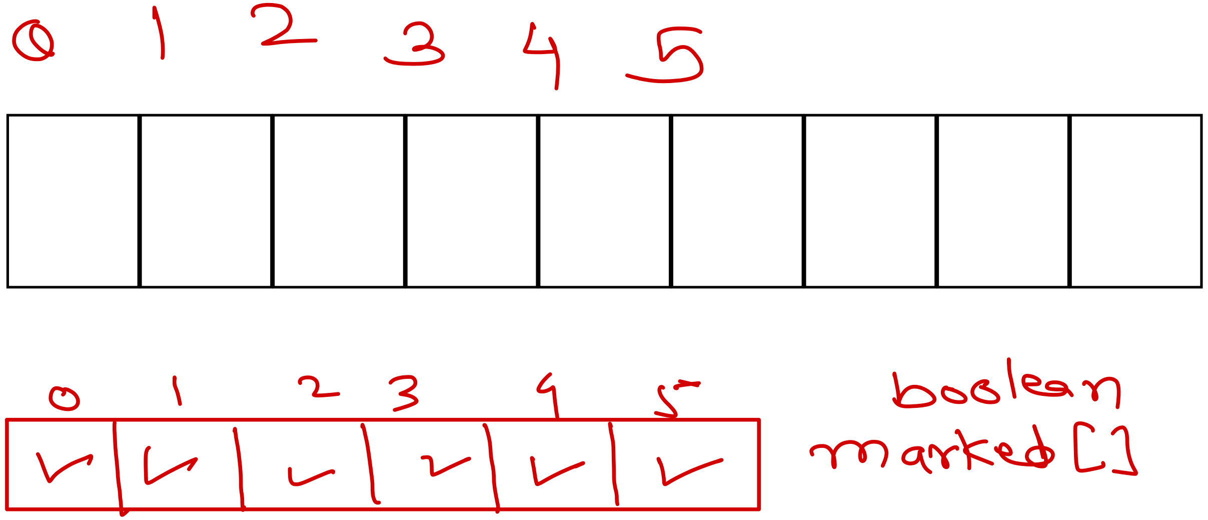
Graph Traversal – BFS Algorithm

BFS trav.

1. Choose a vertex as start vertex.
2. Push start vertex on queue & mark it.
3. Pop vertex from queue.
4. Visit (Print) the vertex.
5. Put all non-~~visited~~ marked neighbours of the vertex on the queue and mark them.
6. Repeat 3-5 until queue is empty.

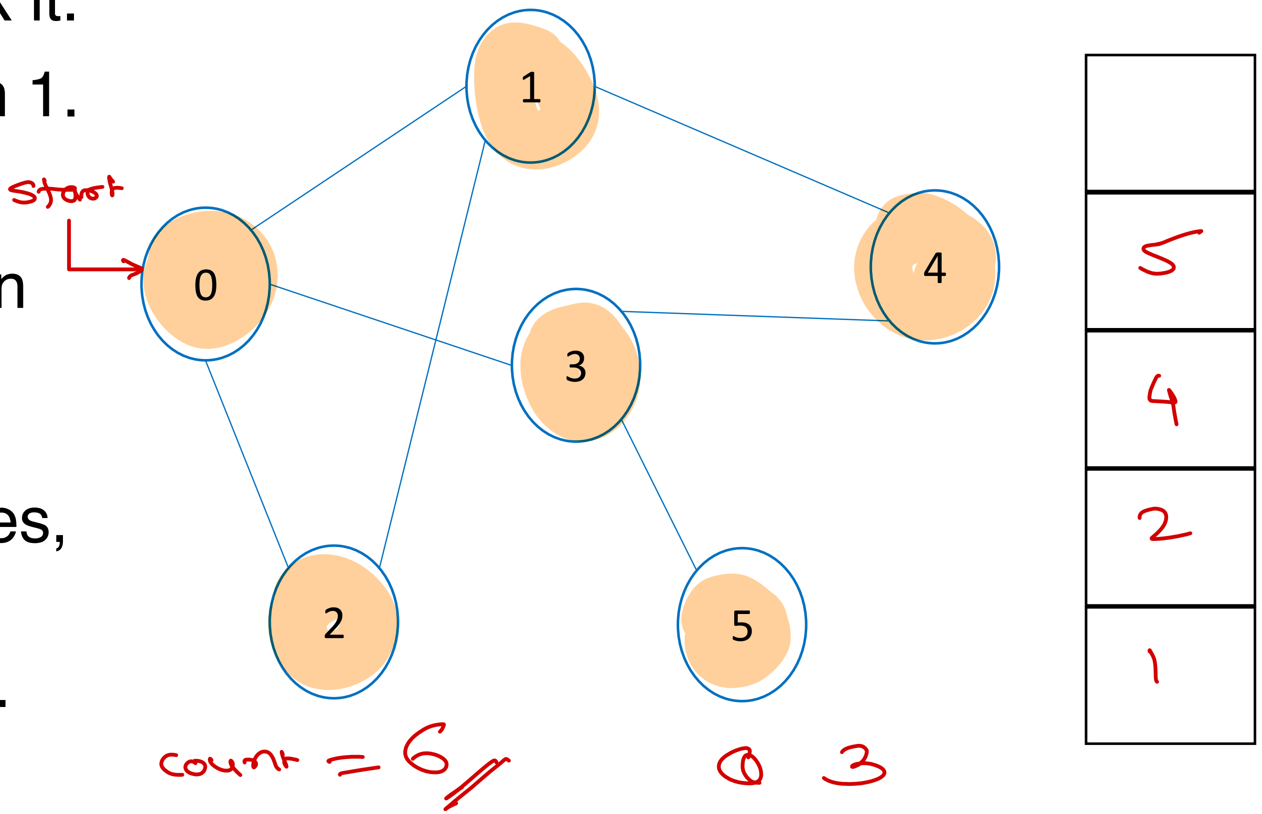
- BFS is also referred as level-wise search algorithm.





Check Connected-ness

1. push starting vertex on stack & mark it.
2. begin counting marked vertices from 1.
3. pop a vertex from stack.
4. push all its non-marked neighbors on the stack, mark them and increment count.
5. if count is same as number of vertices, graph is connected (return).
6. repeat steps 3-5 until stack is empty.
7. graph is not connected (return)



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>