Sunbeam Institute of Information Technology
Pune and Karad

Data structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Quick sort

1. Select pivot/axis/reference element from array
2. Arrange lesser elements on left side of pivot
3. Arrange greater elements on right side of pivot
4. Sort left and right side of pivot again (by quick sort)

Selection pivot :

1. extreme left or right
2. middle element
3. median
   - random 3 element
   - random 5 element
   - random 7 element
4. dual pivot

Quick sort

Number of elements = n

levels of division = log n

comps per level = n

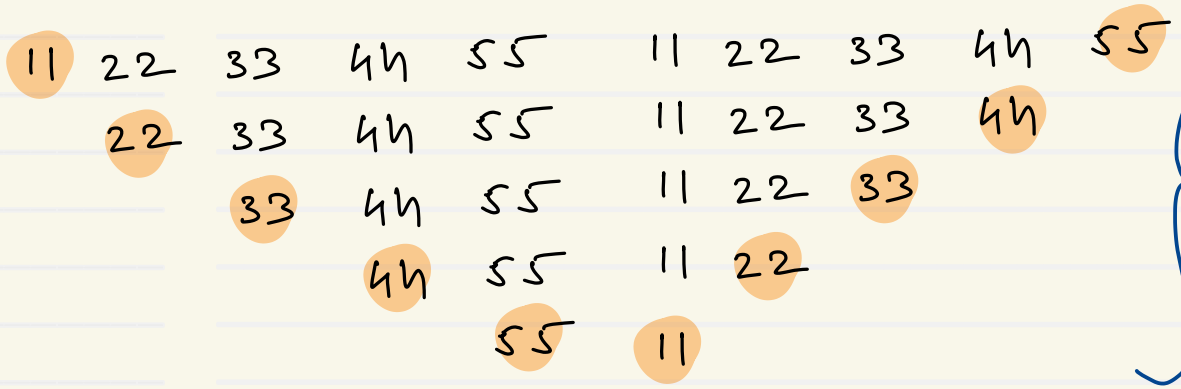
Total comps = n log n

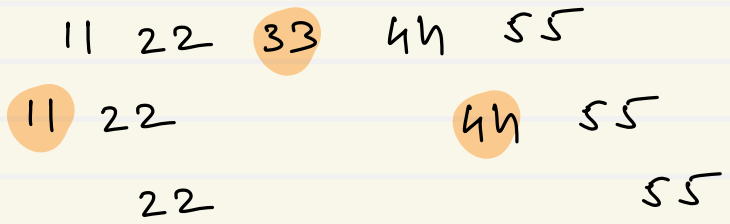
Time ∝ comps

Time ∝ n log n

Best Avg $T(n) = O(n \log n)$

$S(n) = O(1)$





- in quick sort time complexity is dependent on selection pivot.

Graph

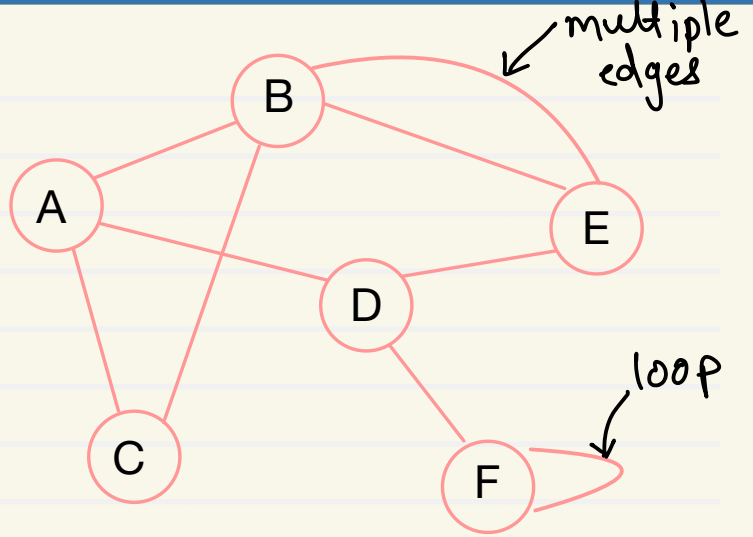
- Graph is a non linear data structure.
- Graph is defined as set of vertices and edges. Vertices (also called as Nodes) hold data, while edges connect vertices and represents relations between them.

  G = { V , E }

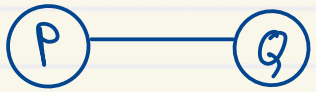
  V = { A , B , C , D , E , F }

  E = { (A-B), (A-C), (A-D), (B-E), (D-E), (D-F) }

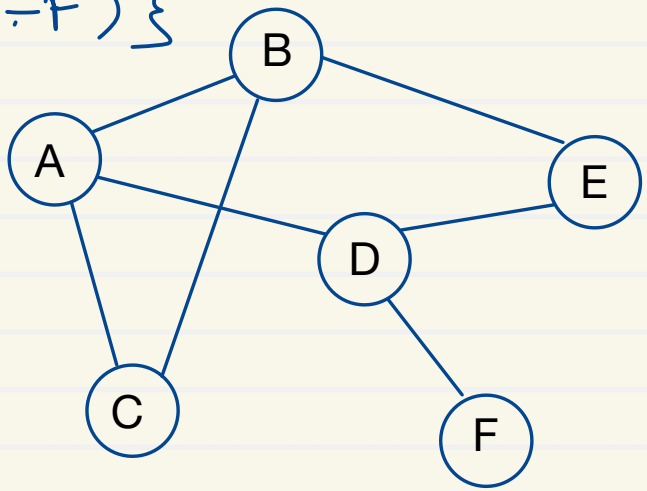
  (B,C)



- When there is an edge from vertex P to vertex Q, P is said to be adjacent to Q.

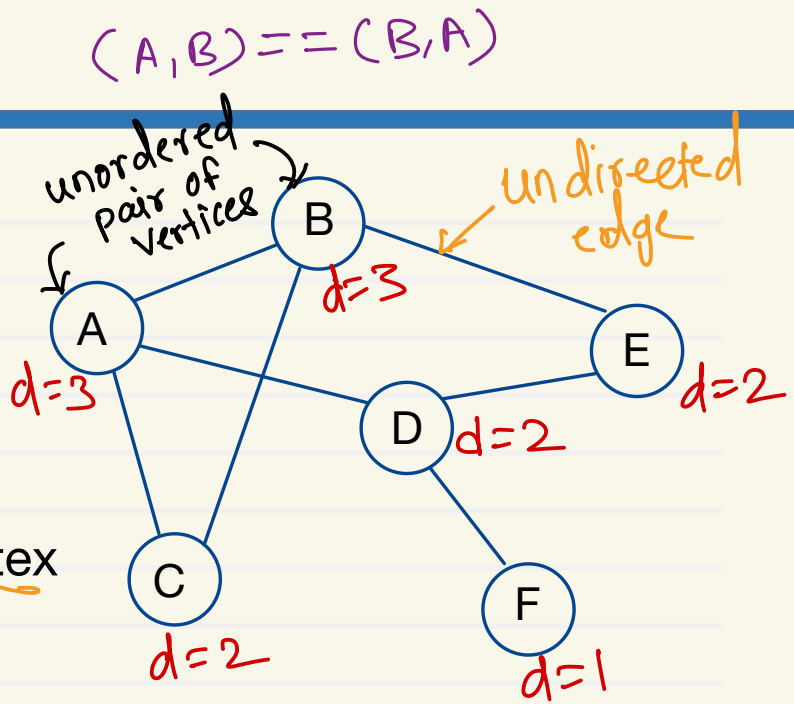


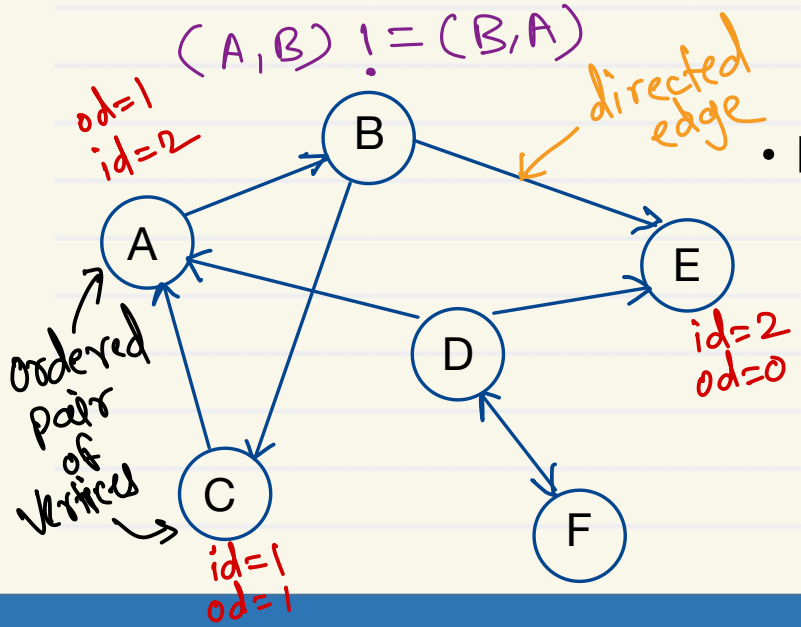
- **Multi graph -** contains multiple edges in adjacent vertices or loops (edge connecting a vertex to it self)
- **Simple graph -** doesn't contains multiple edges in adjacent vertices or loops



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 A is adjacent to B, then B is also adjacent to A.
    - **Degree of vertex :** Number of vertices adjacent to the vertex

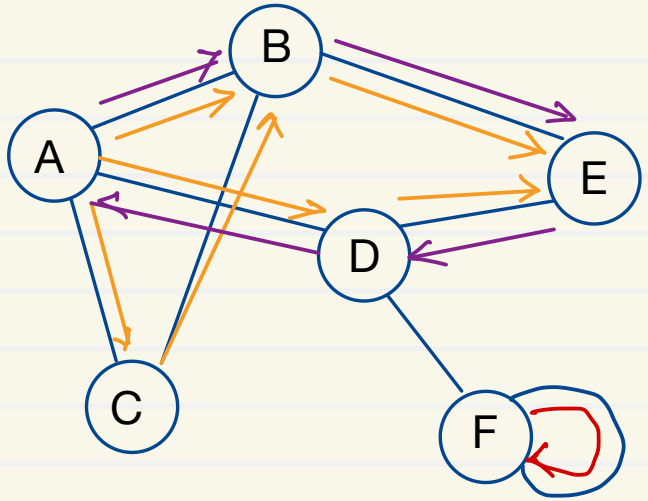




  - **Directed graph : G = { V, E }**
    - V = { A, B, C, D, E, F }
    - E = { <A,B>, <C,A>, <D,A>, <B,C>, <B,E>, <D,E>, <D,F>, <F,D> }
    - If A is adjacent to B, then B is may or may not be adjacent to A.
    - **Out degree :** Number of edges originated from the vertex
    - **In degree :** Number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

Graph

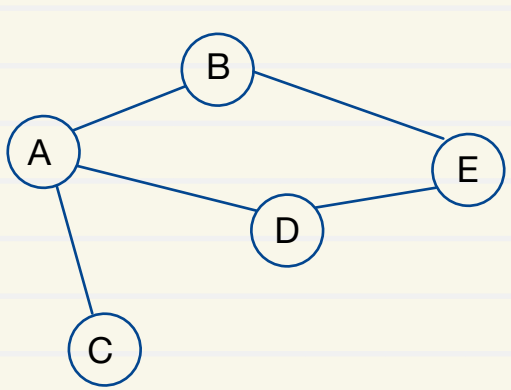
- **Path :** set of edges between two vertices. There can be multiple paths between two vertices.
  - A - D - E
  - A - B - E
  - A - C - B - E

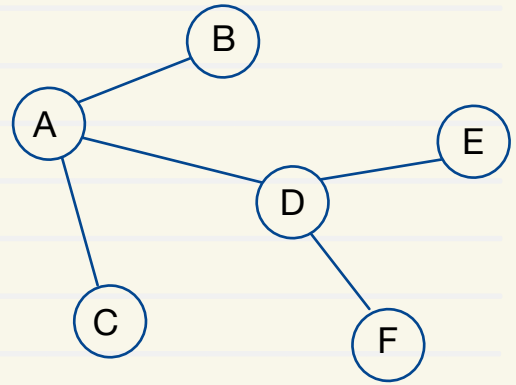


- **Cycle :** path whose start and end vertex is same.
  - A - B - C - A
  - A - B - E - D - A

- **Loop :** edge connecting vertex to itself. It is smallest cycle.
  - F - F

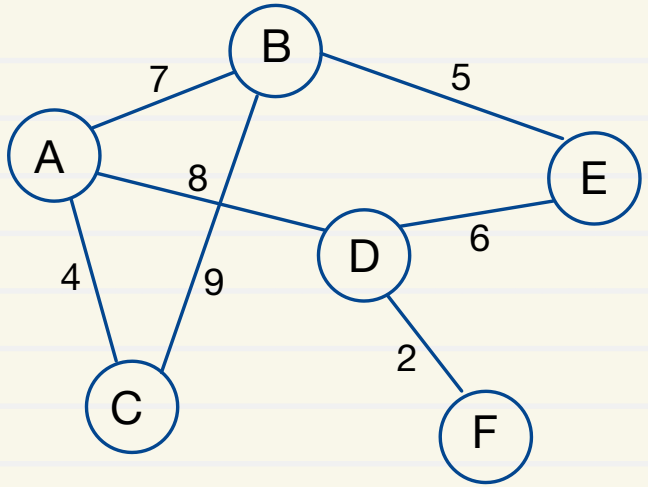
- **Sub graph :** a graph having few vertices and few edges in given graph, is said to be sub graph.



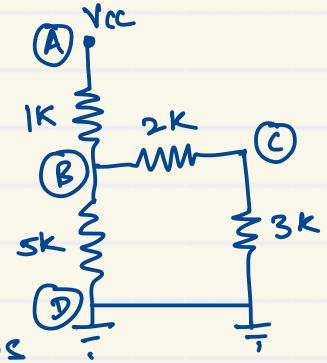


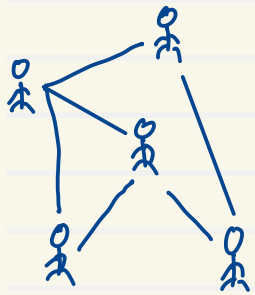
Graph

- **Weighted graph**
  - graph edges have weight associated with them
  - Weight represents some value eg distance, resistance, cost

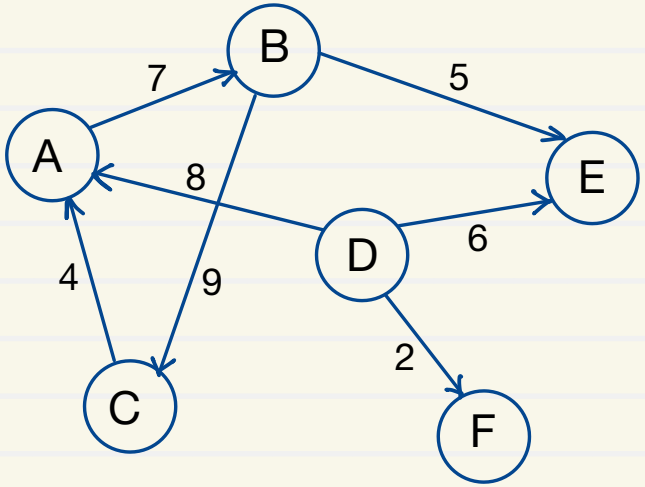


- **Directed weighted graph (Network)**
  - Graph edges have directions as well as weights





- **Graph applications**
  - Electronic circuits
  - Social media
  - Communication network
  - Road network → google maps
  - Flight/Train/Bus services ← online reservation systems
  - Bio-logical and chemical experiments
  - Deep learning (Neural network, Tensor flow)
  - Graph databases (Neo4j)

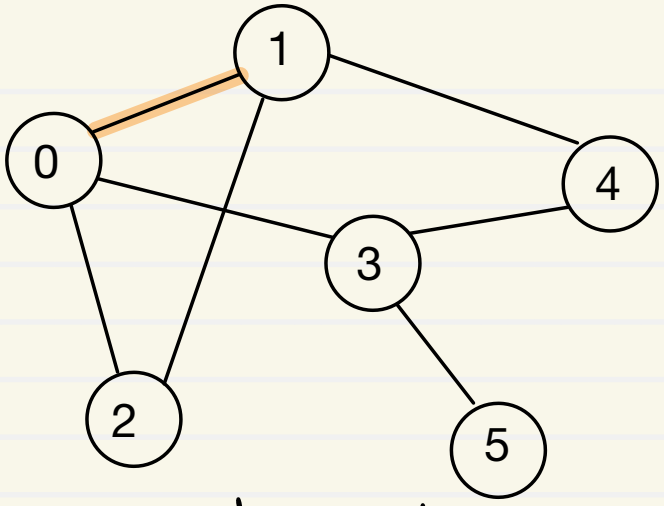


Graph implementation - Adjacency matrix

- If graph is having V vertices, a V x V matrix can be formed to store edges of the graph.
- Each matrix element represents presence or absence of the edge between vertices.
- For non weighted graph, 1 indicates edge and 0 indicates no edge.
- For un directed graph, adjacency matrix is always symmetric across the diagonal.

space complexity of implementation = $O(V^2)$

```
boolean adjMat[V][V];
```



destination

source

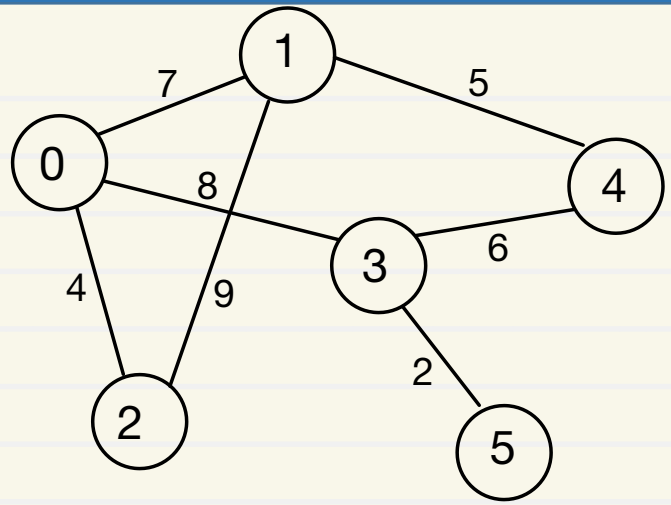
	0	1	2	3	4	5
0	0	1	1	1	0	0
1	1	0	1	0	1	0
2	1	1	0	0	0	0
3	1	0	0	0	1	1
4	0	1	0	1	0	0
5	0	0	0	1	0	0

Graph implementation - Adjacency matrix

- If graph is having V vertices, a V x V matrix can be formed to store edges of the graph.
- Each matrix element represents presence or absence of the edge between vertices.
- For weighted graph, weight value indicates edge and infinity sign indicates no edge.
- For un directed graph, adjacency matrix is always symmetric across the diagonal.

space complexity of implementation = $O(V^2)$

```
int adjMat[V][V];
```



destination

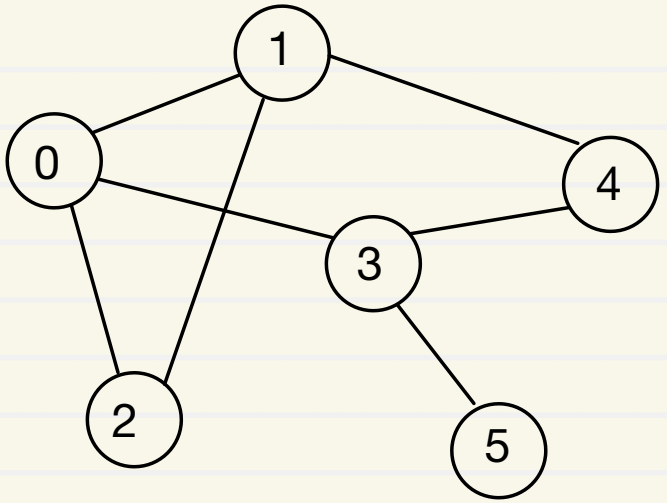
source

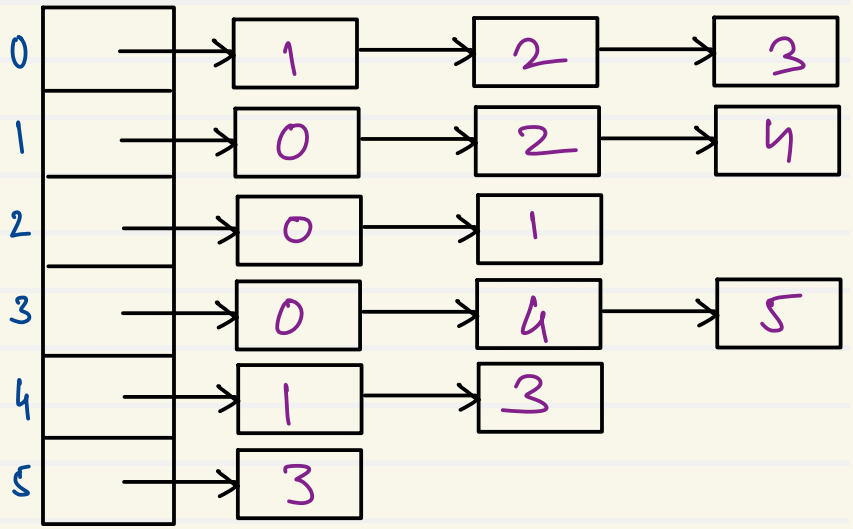
	0	1	2	3	4	5
0	∞	7	4	8	∞	∞
1	7	∞	9	∞	5	∞
2	4	9	∞	∞	∞	∞
3	8	∞	∞	∞	6	2
4	∞	5	∞	6	∞	∞
5	∞	∞	∞	2	∞	∞

Graph implementation - Adjacency list

- Each vertex holds list of it's adjacent vertices.
- For non weighted graph, only neighbour vertices are stored.
- For weighted graph, neighbour vertices and weights of connecting edges are stored.
- If graph is sparse graph (with fewer number of edges), this implementation is more efficient (as compared to adjacency matrix method)

space complexity of implementation = O(V+E)

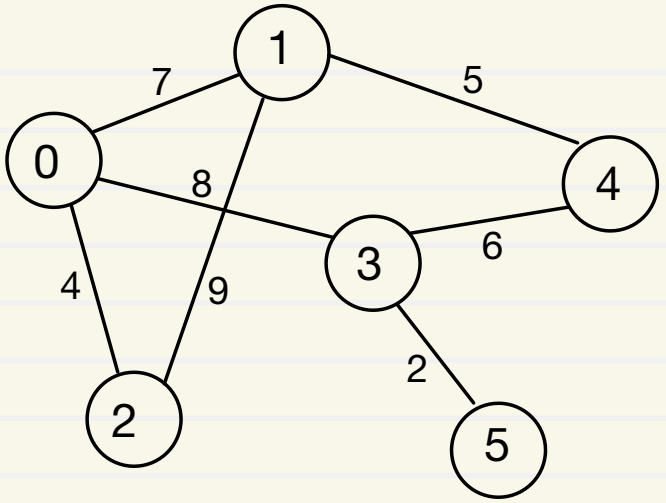


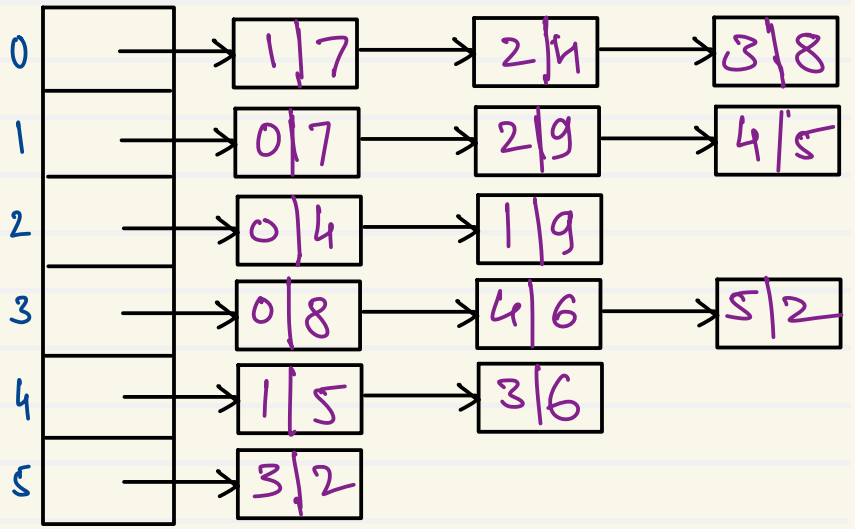


Graph implementation - Adjacency list

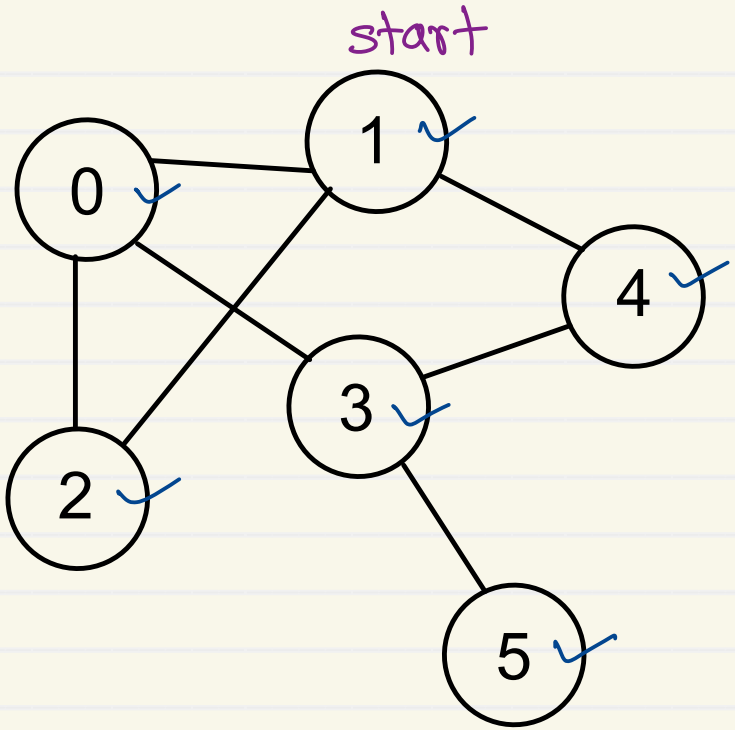
- Each vertex holds list of it's adjacent vertices.
- For non weighted graph, only neighbour vertices are stored.
- For weighted graph, neighbour vertices and weights of connecting edges are stored.
- If graph is sparse graph (with fewer number of edges), this implementation is more efficient (as compared to adjacency matrix method)

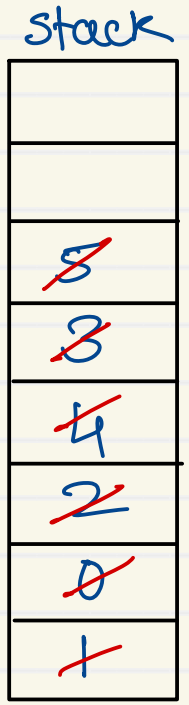
space complexity of implementation = O(V+E)





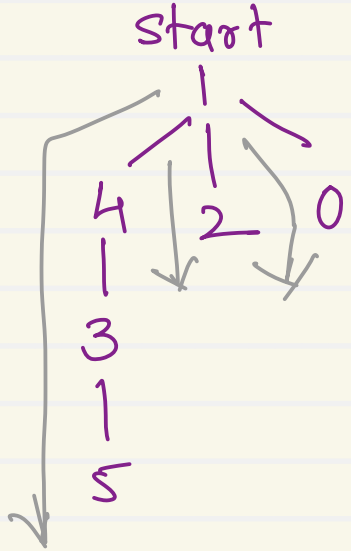
DFS traversal



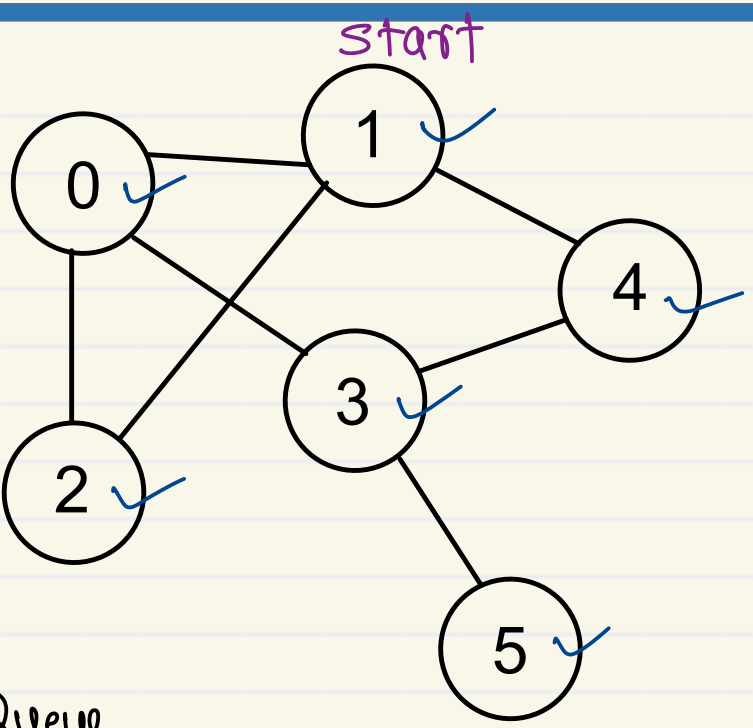


Traversal : 1, 4, 3, 5, 2, 0

1. Choose a vertex as start vertex.
2. Push start vertex on stack and mark it.
3. Pop vertex from stack.
4. Visit/print popped vertex.
5. Push all non marked neighbours of the vertex on the stack and mark them.
6. Repeat step 3 - 5 until stack is empty.



BFS traversal



Queue

1	0	2	4	3	5	

Traversal : 1, 0, 2, 4, 3, 5

1. Choose a vertex as start vertex.
2. Push start vertex on queue and mark it.
3. Pop vertex from queue .
4. Visit/print popped vertex.
5. Push all non marked neighbours of the vertex on the queue and mark them.
6. Repeat step 3 - 5 until queue is empty.

- BFS is also referred as level wise search algorithm