

# Pupils Plan S25



Graph Basics (Representation + Traversal)

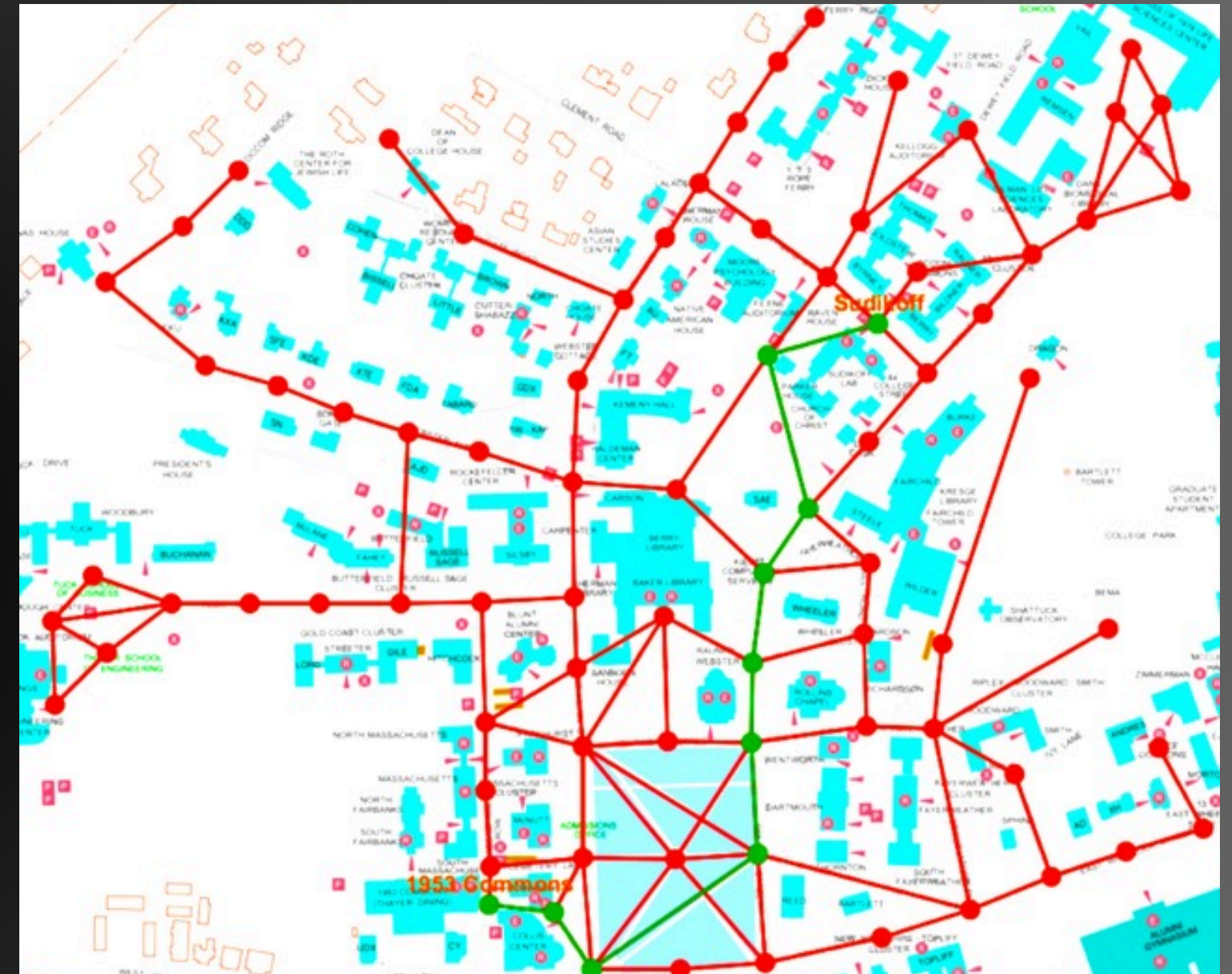
◆ Real-world applications:

📶 Networks (Wi-Fi routers, Internet routing)

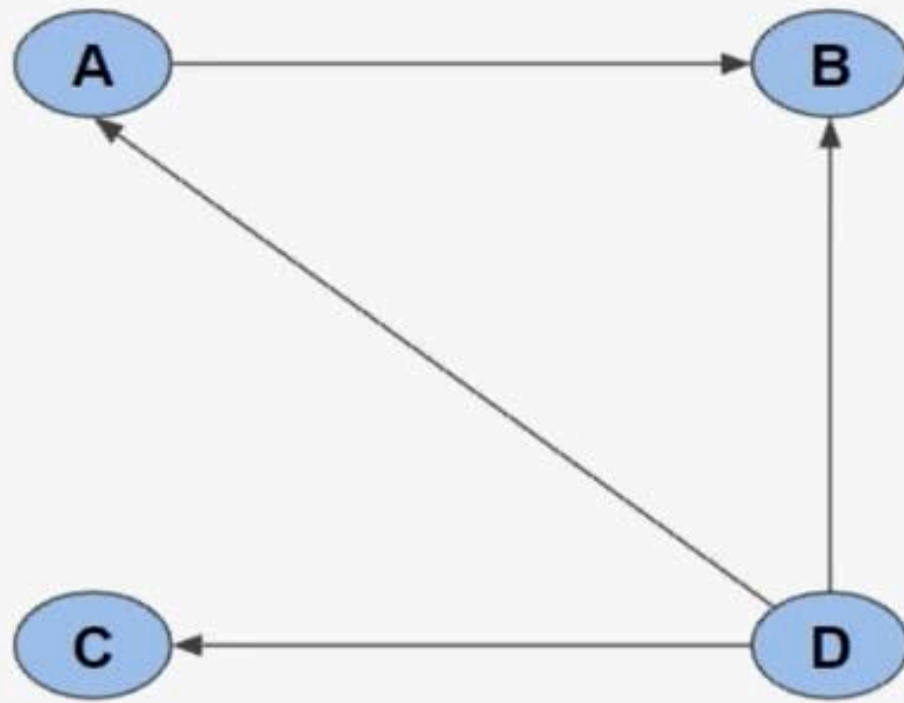
🗺️ Maps & Navigation (Google Maps, GPS)

📊 Dependencies (Course scheduling, Build order)

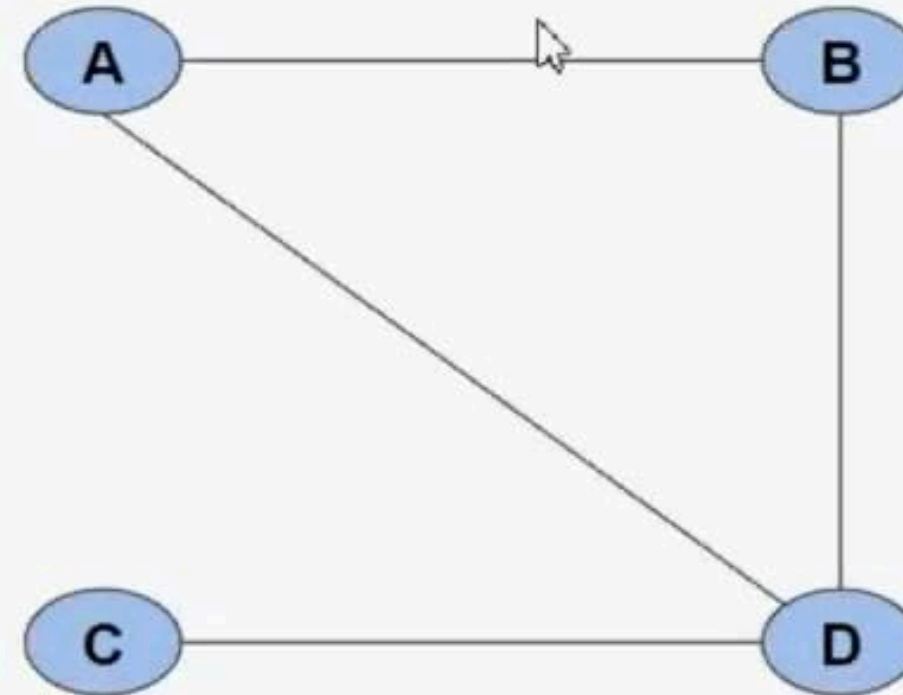
👤 Social Networks (Friend suggestions, Influencers)



**Graph can be :**  
**Directed / Undirected**



**DIRECTED GRAPH**



### Example Input:

$n = 5, m = 5$

Edges:

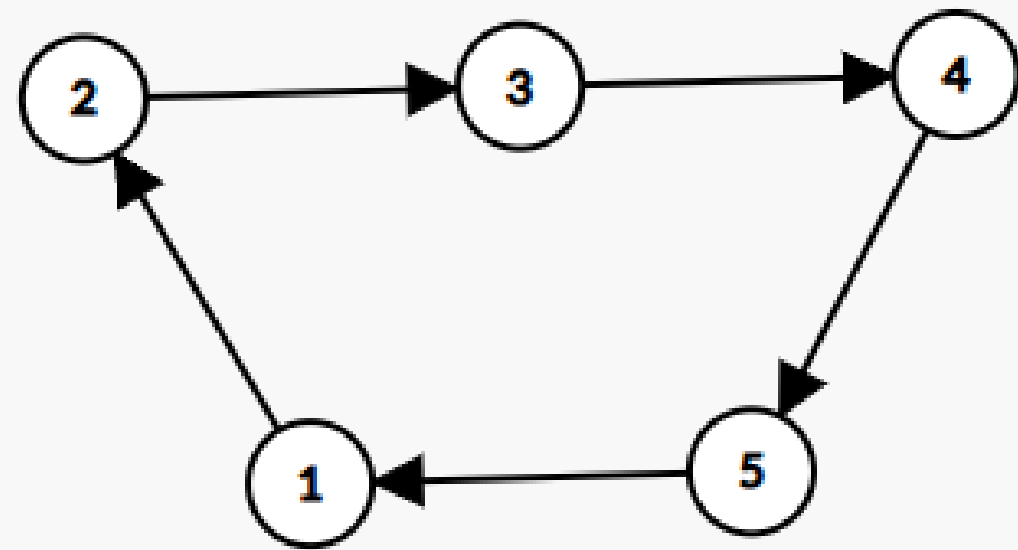
1 2

2 3

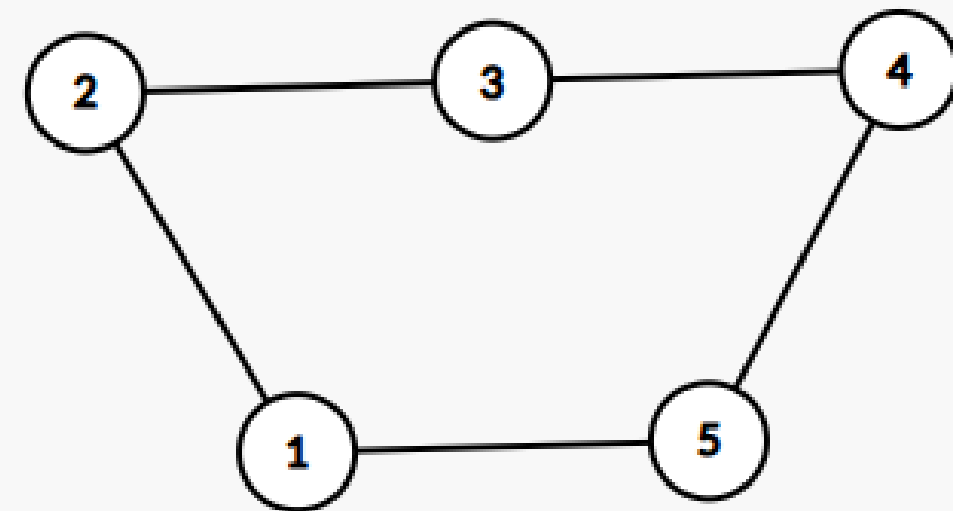
3 4

4 5

5 1



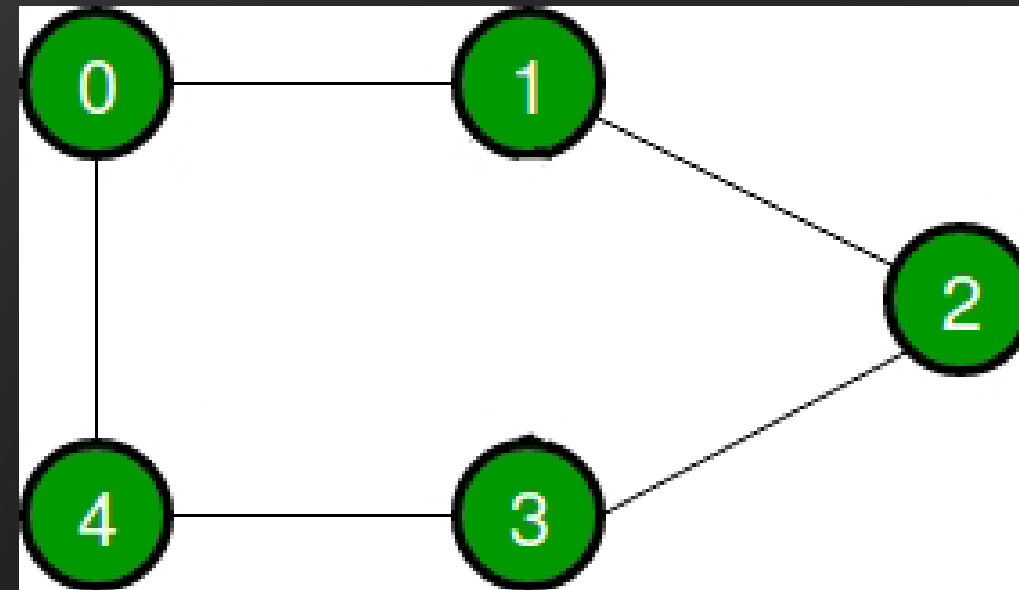
Directed



Undirected




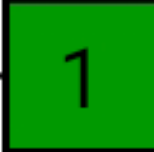
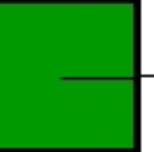



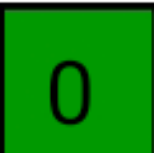
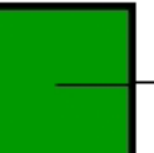
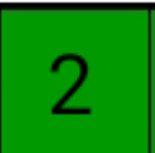
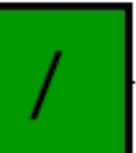


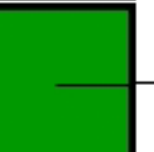



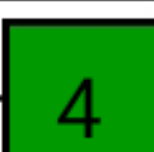




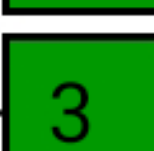

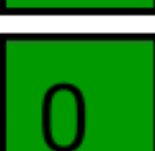

# How Do We Represent Graphs?



Adjacency Matrix:

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

Adjacency List:

0		→		1	→		→		4	/	
1		→		0	→		→		2	/	
2		→		1	→		→		3	/	
3		→		4	→		→		2	/	
4		→		3	→		→		0	/	

## Example Input:

`n = 5, m = 5`

Edges:

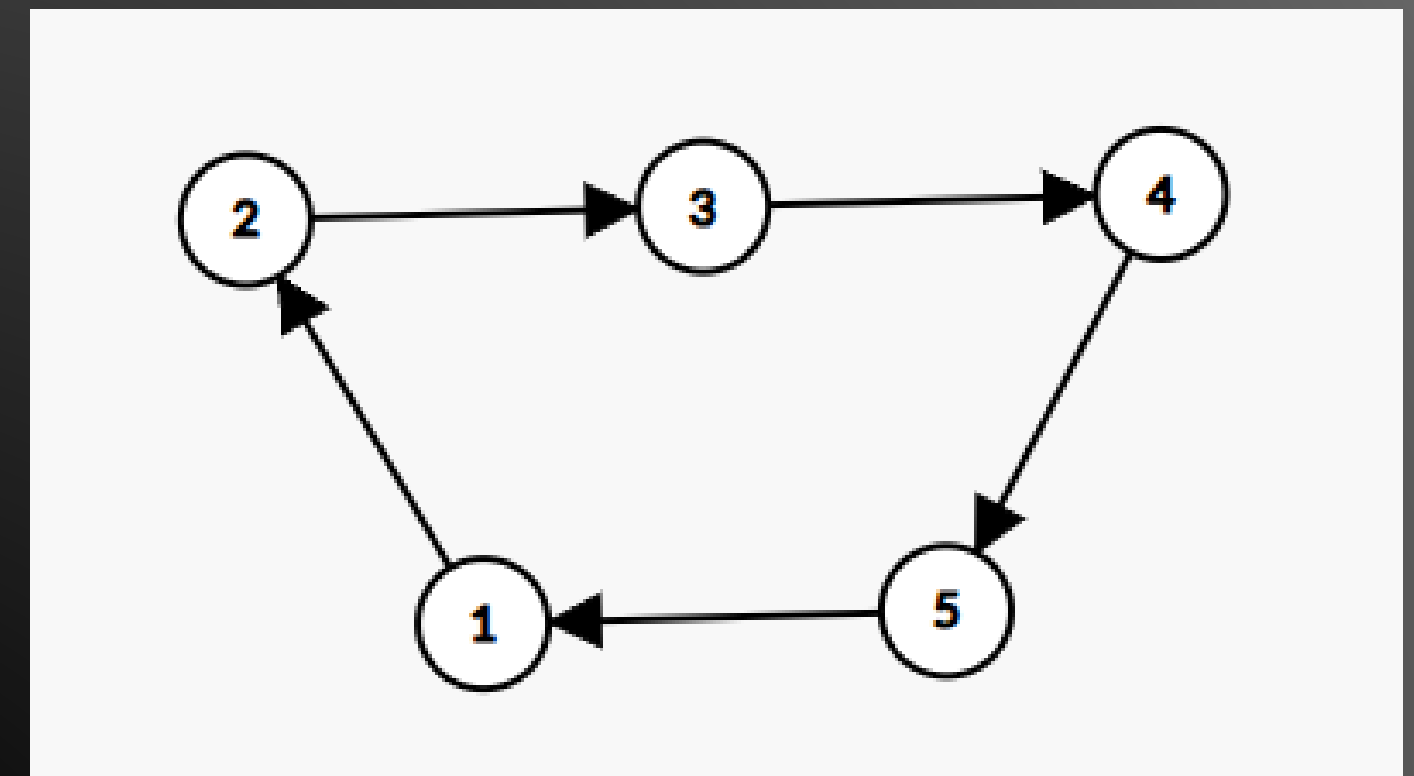
1 2

2 3

3 4

4 5

5 1



Directed

```
int n = sc.nextInt(); // number of nodes
int m = sc.nextInt(); // number of edges
Map<Integer, ArrayList<Integer>> Edges = new HashMap<>();
for (int i = 0; i < m; i++) {
    int u = sc.nextInt();
    int v = sc.nextInt();

    // Insert u → v
    Edges.putIfAbsent(u, new ArrayList<>());
    Edges.get(u).add(v);
}
```

Adjacency List:

```
int n = sc.nextInt(); // number of nodes
int m = sc.nextInt(); // number of edges
int[][] edges = new int[n][n]; // O(n^2)
for (int i = 0; i < m; i++) {
    int u = sc.nextInt();
    int v = sc.nextInt();
    edges[u][v] = 1;
}
```

Adjacency Matrix:

## Example Input:

`n = 5, m = 5`

Edges:

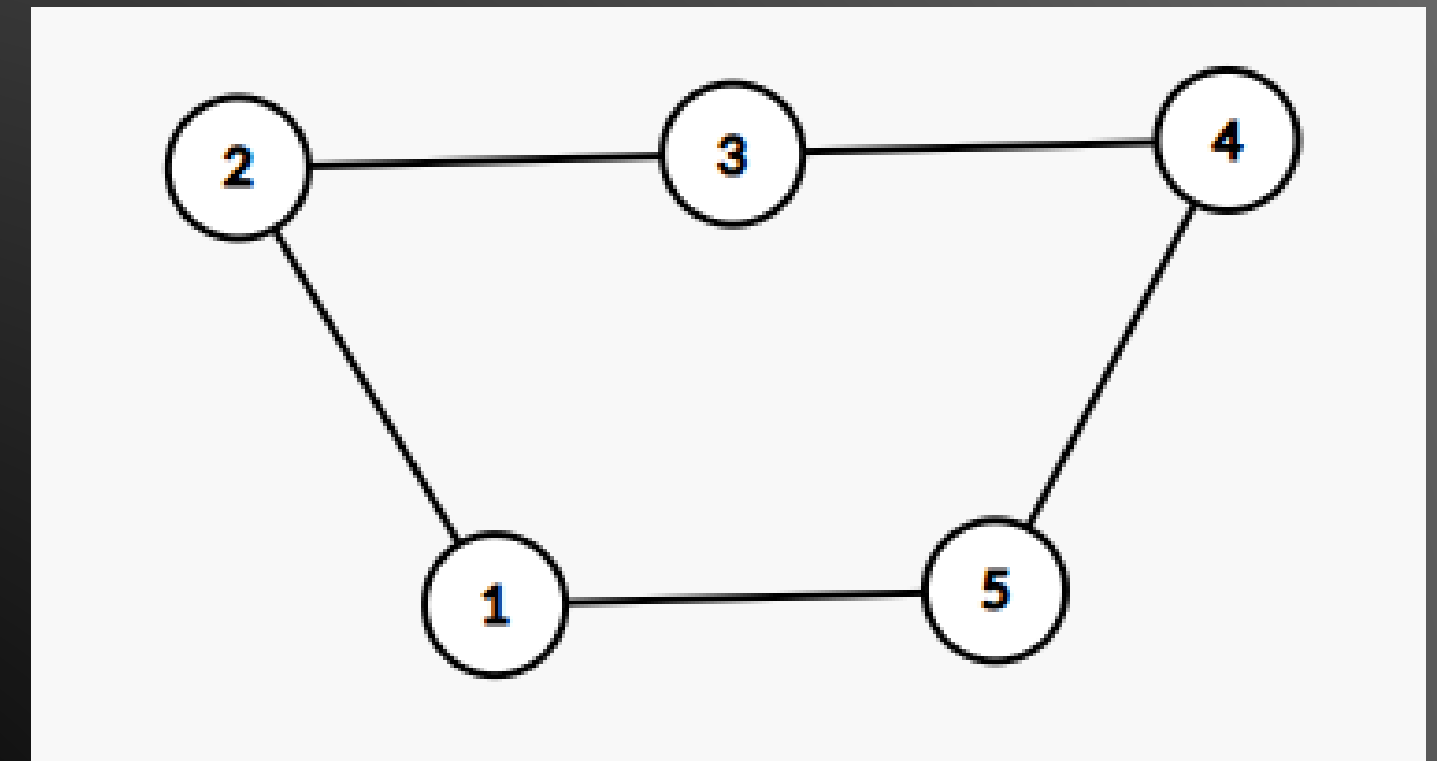
1 2

2 3

3 4

4 5

5 1



UnDirected

```
int n = sc.nextInt(); // number of nodes
int m = sc.nextInt(); // number of edges
Map<Integer, ArrayList<Integer>> Edges = new HashMap<>();

for (int i = 0; i < m; i++) {
    int u = sc.nextInt();
    int v = sc.nextInt();

    Edges.putIfAbsent(u, new ArrayList<>());
    Edges.get(u).add(v);

    Edges.putIfAbsent(v, new ArrayList<>()); //Undirected
    Edges.get(v).add(u);
}
```

Adjacency List:

```
int n = sc.nextInt(); // number of nodes
int m = sc.nextInt(); // number of edges
int[][] edges = new int[n][n]; // O(n^2)
for (int i = 0; i < m; i++) {
    int u = sc.nextInt();
    int v = sc.nextInt();
    edges[u][v] = 1;
    edges[v][u] = 1; // Undirected
}
```

Adjacency Matrix:

Operations	Adjacency Matrix	Adjacency List
Storage Space	This representation makes use of $V \times V$ matrix, so space required in worst case is $O(V^2)$ .	In this representation, for every vertex we store its neighbours. In the worst case, if a graph is connected $O(V)$ is required for a vertex and $O(E)$ is required for storing neighbours corresponding to every vertex. Thus, overall space complexity is $O( V + E )$ .
Adding a vertex	In order to add a new vertex to $V \times V$ matrix the storage must be increased to $( V +1)^2$ . To achieve this we need to copy the whole matrix. Therefore the complexity is $O(V^2)$ .	There are two pointers in adjacency list first points to the front node and the other one points to the rear node. Thus insertion of a vertex can be done directly in $O(1)$ time.
Adding an edge	To add an edge say from $i$ to $j$ , $matrix[i][j] = 1$ which requires $O(1)$ time.	Similar to insertion of vertex here also two pointers are used pointing to the rear and front of the list. Thus, an edge can be inserted in $O(1)$ time.
Removing a vertex	In order to remove a vertex from $V \times V$ matrix the storage must be decreased to $V^2$ from $( V +1)^2$ . To achieve this we need to copy the whole matrix. Therefore the complexity is $O(V^2)$ .	In order to remove a vertex, we need to search for the vertex which will require $O( V )$ time in worst case, after this we need to traverse the edges and in worst case it will require $O( E )$ time. Hence, total time complexity is $O( V + E )$ .
Removing an edge	To remove an edge say from $i$ to $j$ , $matrix[i][j] = 0$ which requires $O(1)$ time.	To remove an edge traversing through the edges is required and in worst case we need to traverse through all the edges. Thus, the time complexity is $O( E )$ .
Querying	In order to find for an existing edge the content of matrix needs to be checked. Given two vertices say $i$ and $j$ $matrix[i][j]$ can be checked in $O(1)$ time.	In an adjacency list every vertex is associated with a list of adjacent vertices. For a given graph, in order to check for an edge we need to check for vertices adjacent to given vertex. A vertex can have at most $O( V )$ neighbours and in worst case we would have to check for every adjacent vertex. Therefore, time complexity is $O( V )$ .



## **Problem - B**

Codeforces. Programming competitions and contests, programming community



## DFS – Depth-First Search (Adjacency List)

```
void dfs(int node, Set<Integer> visited, Map<Integer, ArrayList<Integer>> edges) {  
    visited.add(node);  
    if (!edges.containsKey(node)) return;  
  
    for (int neighbor : edges.get(node)) {  
        if (!visited.contains(neighbor)) {  
            dfs(neighbor, visited, edges);  
        }  
    }  
}
```

Time Complexity:  $O(V + E)$

Space Complexity:  $O(V + E)$

## DFS – Depth-First Search (Adjacency Matrix)

```
int n = sc.nextInt(), m = sc.nextInt();
int[][] adj = new int[n + 1][n + 1];
boolean[] visited = new boolean[n + 1];

for (int i = 0; i < m; i++) {
    int u = sc.nextInt(), v = sc.nextInt();
    adj[u][v] = 1;
    adj[v][u] = 1; // remove for directed
}

dfs(1);

void dfs(int node) {
    visited[node] = true;
    for (int i = 1; i <= n; i++) {
        if (adj[node][i] == 1 && !visited[i]) dfs(i);
    }
}
```

Time Complexity:  $O(V^2)$

Space Complexity:  $O(V^2)$

## BFS— Breadth -First Search (Adjacency List)

```
int n = sc.nextInt(), m = sc.nextInt();
ArrayList<ArrayList<Integer>> adj = new ArrayList<>();
for (int i = 0; i <= n; i++) adj.add(new ArrayList<>());
boolean[] visited = new boolean[n + 1];

for (int i = 0; i < m; i++) {
    int u = sc.nextInt(), v = sc.nextInt();
    adj.get(u).add(v);
    adj.get(v).add(u); // remove for directed
}

Queue<Integer> q = new LinkedList<>();
q.add(1);
visited[1] = true;

while (!q.isEmpty()) {
    int node = q.poll();
    for (int nei : adj.get(node))
        if (!visited[nei]) {
            visited[nei] = true;
            q.add(nei);
        }
}
```

Time:  $O(V + E)$

Space:  $O(V + E)$

## BFS– Breadth -First Search (Adjacency Matrix)

```
int n = sc.nextInt(), m = sc.nextInt();
int[][] adj = new int[n + 1][n + 1];
boolean[] visited = new boolean[n + 1];

for (int i = 0; i < m; i++) {
    int u = sc.nextInt(), v = sc.nextInt();
    adj[u][v] = 1;
    adj[v][u] = 1; // remove for directed
}

Queue<Integer> q = new LinkedList<>();
q.add(1);
visited[1] = true;

while (!q.isEmpty()) {
    int node = q.poll();
    for (int i = 1; i <= n; i++) {
        if (adj[node][i] == 1 && !visited[i]) {
            visited[i] = true;
            q.add(i);
        }
    }
}
```

Time Complexity:  $O(V^2)$

Space Complexity:  $O(V^2)$





# LeetCode

## Number of Provinces

Can you solve this real interview question? Number of Provinces - There are  $n$  cities. Some of them are connected, while some are not. If city  $a$  is connected directly with city  $b$ , and city  $b$  is connected directly with city ...

 LeetCode