"You don't have to be a genius to code, you just have to be persistent."

Hello Everyone
Very special Good Evening
to All of you ☺
We will Start
from 9:06 pm

Graphs 1

~~~~~~~~~~

Agenda :

~~~~~~~~~~

Introduction {
1. Introduction to Graph
2. Types of Graphs

creation {
3. How to store data in Graph

traversal {
4. BFS (Breadth First Search)

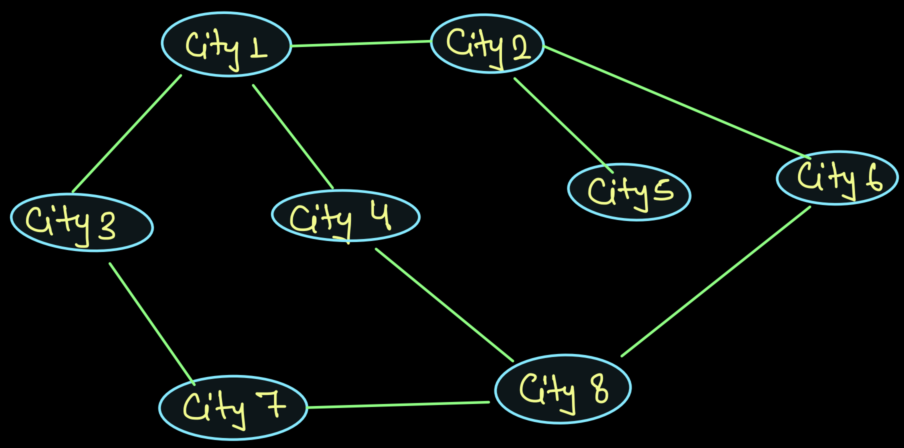
Searching in Graph {
5. Is Path Available from Source to Destination

~~~~~~~~~~~~~~~~~~~~~~~~~~~

Introduction to Graph

~~~~~~~~~~~~~~~~~~~~~~~~~~~

Want to store information of cities and their connectivity.



With the help of graph data structure we can implement the storage of connection.

Cities ≡ vertex

Links ≡ Edge

No. of Vertex ? → 8

No. of Edges ? → 9

Collection of vertex and edges are known as Graph.

neighbour of City 1 → City 2, City 3, & City 4

neighbour of city 8 → City 6, City 4, City 7

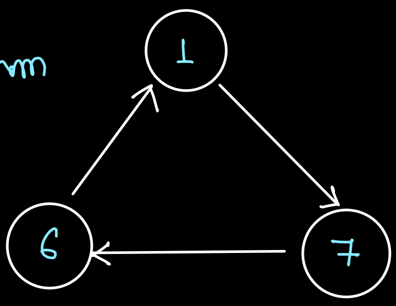
nbr(city 5) = City 2

Direct connect vertex are know as neighbour.

Types of Graphs

1. Based on type of edges:

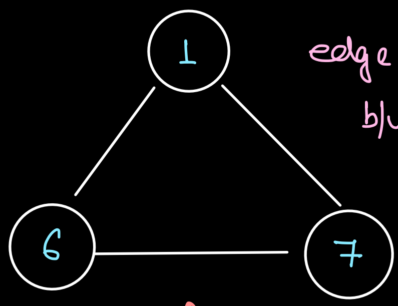
edge from 6 to 1



directed graph

[instagram, youtube scriber]

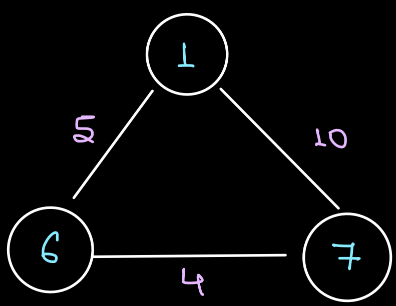
edge is available b/w 1 to 7



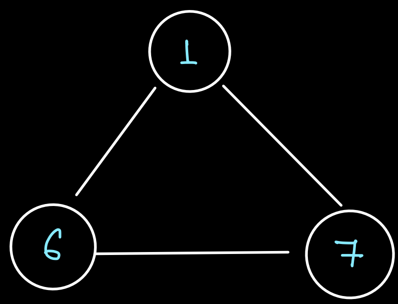
undirected graph

[facebook, LinkedIn connection]

2. Based on Edge wt. present or not:

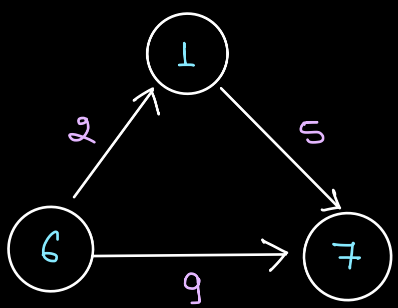


weighted graph



unweighted graph.

3. Combination of above types are also possible:



directed weighted graph

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

How to store data in Graph

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

There is too famous implementation available for graph.

1. Adjacency matrix
2. Adjacency list

1. Adjacency matrix:

Vertex = 7, Edges = 8

```
int[][] graph = new int[v+x][v+x];
```

	0	1	2	3	4	5	6
0	0	1	0	1	0	0	0
1	1	0	1	0	0	0	0
2	0	1	0	1	0	0	0
3	1	0	1	0	1	0	0
4	0	0	0	1	0	1	1
5	0	0	0	0	1	0	1
6	0	0	0	0	1	1	0

edges[][]:

0	0	3
1	0	1
2	2	3
i → 3	3	4
4	1	2
5	4	5
6	4	6
7	5	6

```
int u = edge[i][0];
int v = edge[i][1];
// make a connection
b/w u & v
    u → v
    v → u
grab[u][v] = 1
graph[v][u] = 1
```

undirected graph.

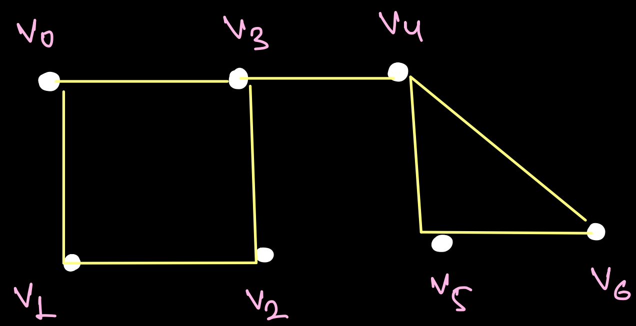
```
int u = edge[i][0];  ,  u = 2
int v = edge[i][1];  ,  v = 3
graph[u][v] = 1;        graph[2][3] = 1
graph[v][u] = 1;        graph[3][2] = 1
```

	0	1	2	3	4	5	6
0	0	1	0	1	0	0	0
1	1	0	1	0	0	0	0
2	0	1	0	1	0	0	0
3	1	0	1	0	1	0	0
4	0	0	0	1	0	1	1
5	0	0	0	0	1	0	1
6	0	0	0	0	1	1	0

undirected + unweighted graph

No. of vertex → Row length OR Column length

= 7



Weighted + Directed graph:

Vtx = 7, Edge = 8

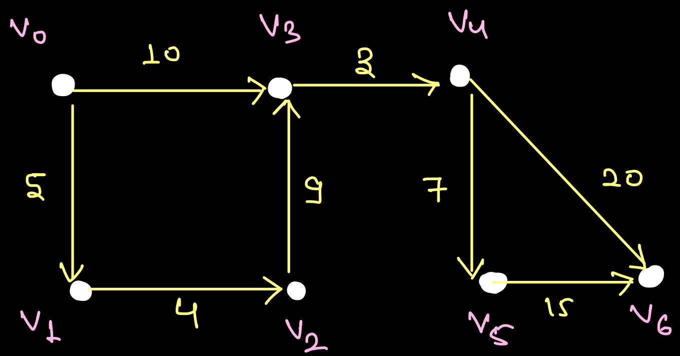
```
int[][] graph = new int[vtx][vtx];
```

	0	1	2	3	4	5	6
0	0	5	0	10	0	0	0
1	0	0	4	0	0	0	0
2	0	0	0	9	0	0	0
3	0	0	0	0	2	0	0
4	0	0	0	0	0	7	20
5	0	0	0	0	0	0	15
6	0	0	0	0	0	0	0

	u	v	wt
0	0	3	→ 10 ✓
1	0	1	→ 5 ✓
2	2	3	→ 9 ✓
3	3	4	→ 3 ✓
4	1	2	→ 4 ✓
5	4	5	→ 7 ✓
6	4	6	→ 20 ✓
7	5	6	→ 15 ✓

```
int u = edge[i][0];
int v = edge[i][1];
int wt = edge[i][2];
// Edge avl. from
   u to v with
   weight wt.

graph[u][v] = wt;
```

	0	1	2	3	4	5	6
0	0	5	0	10	0	0	0
1	0	0	4	0	0	0	0
2	0	0	0	9	0	0	0
3	0	0	0	0	3	0	0
4	0	0	0	0	0	7	20
5	0	0	0	0	0	0	15
6	0	0	0	0	0	0	0



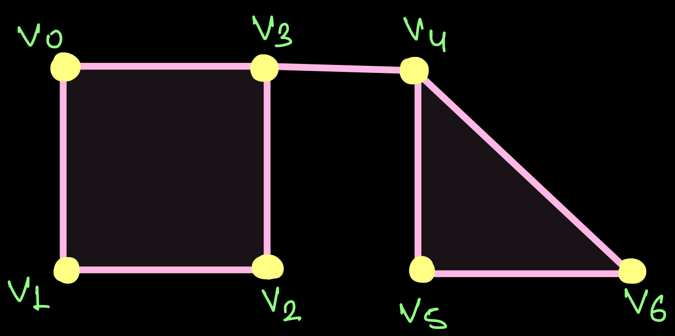
Major disadvantage of Adjacency matrix:

Major disadvantage is wastage of memory, that's why most of the time we will consider Adjacency list instead of Adjacency matrix

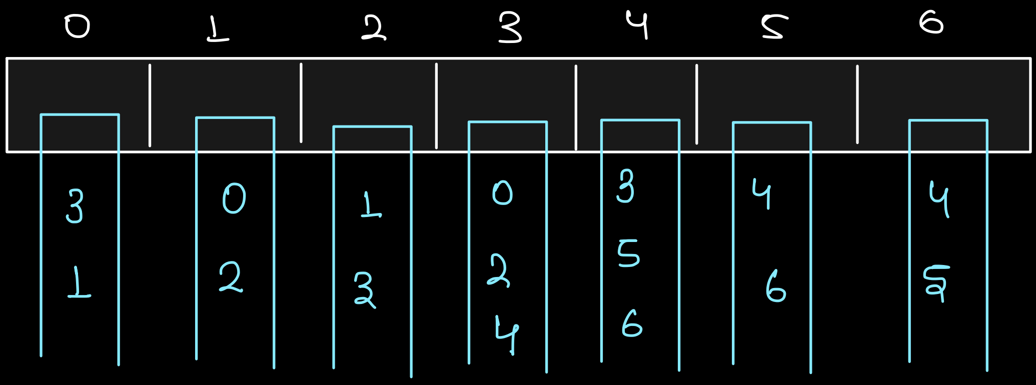
2. Adjacency List:

Undirected graph.

Vtx = 7, Edge = 8

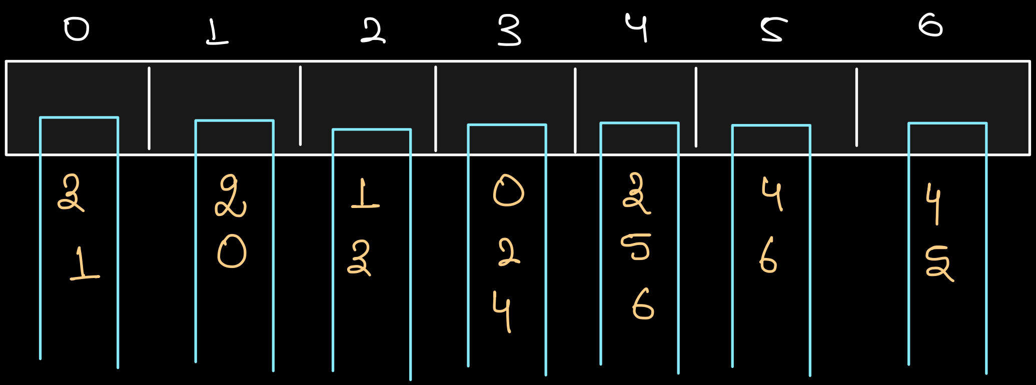


0	0	3
1	0	1
2	2	3
3	3	4
4	1	2
5	4	5
6	4	6
7	5	6



ArrayList < ArrayList < Integer> > graph

Understanding of Implementation P1:



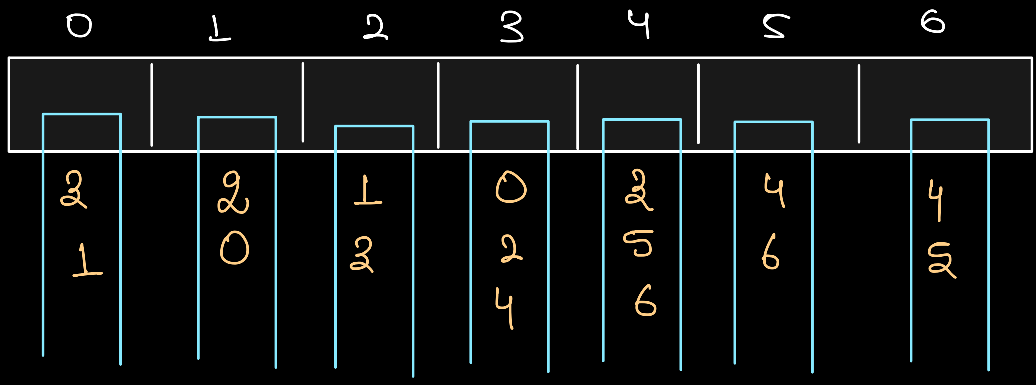
Vtx = 7, Edge = 8

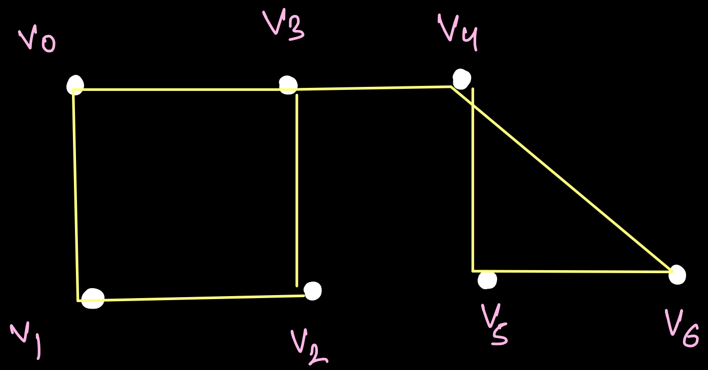
0	0	3
1	0	1
2	2	3
3	3	4
4	1	2
5	4	5
6	4	6
7	5	6

```
int u = edge[i][0]; → 2
int v = edge[i][1]; → 1

graph.get(2).add(1);

graph.get(1).add(2);
```

```
int u = edge[i][0];
int v = edge[i][1];

graph.get(u).add(v);
graph.get(v).add(u);
```

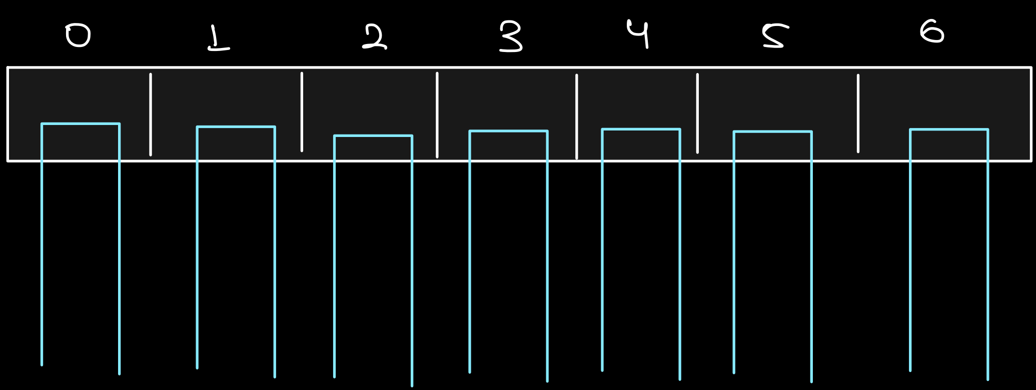
u: 2,
v: 1

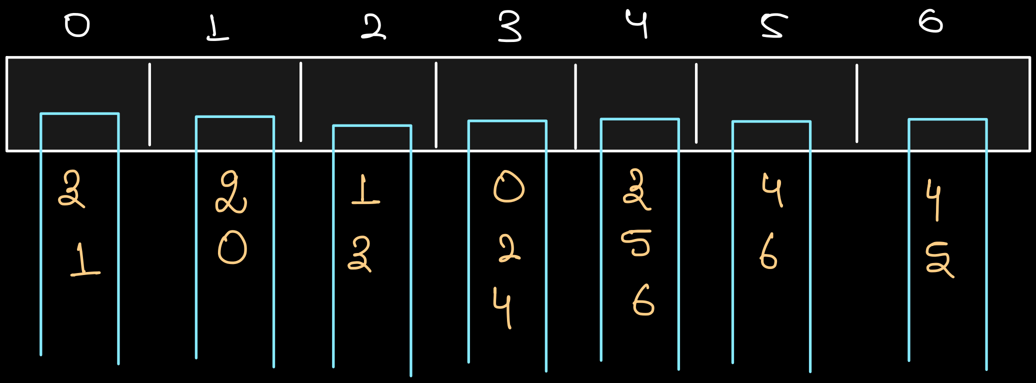




$\underbrace{10:30 - 10:40}_{\text{Break time}}$ → implementation graph + Display.

```
ArrayList<ArrayList<Integer>> graph = new AL<>();

for(int v=0; v<vtx; v++){
    graph.add(new AL< >());
}
```





[0] → 3,1
[1] → 2,0
[2] → 1,3
[3] → 0, 2, 4
[4] → 3, 4, 5
[5] → 4,6
[6] → 4,5

o/p for display.

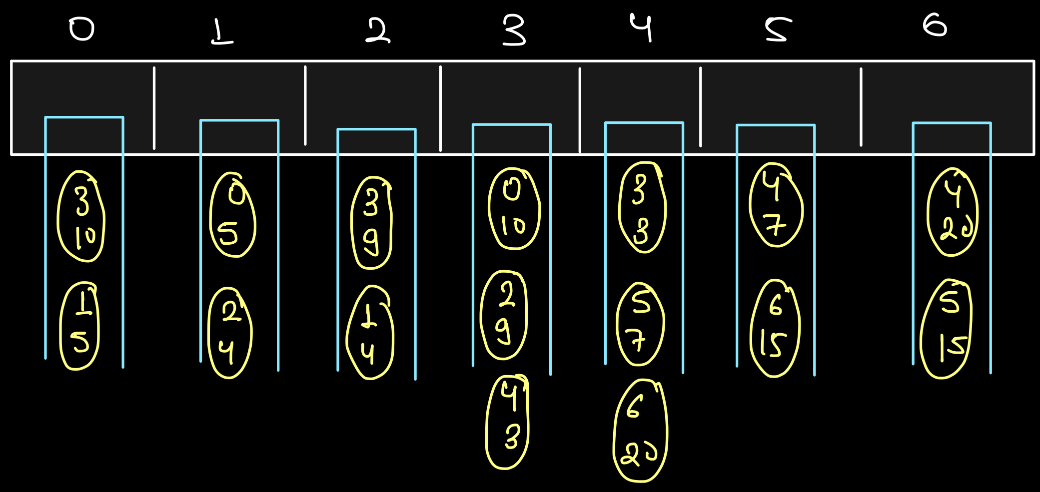
Screen shot of implementation.

```java
import java.util.*;

class Main {

    public static void display(ArrayList<ArrayList<Integer>> graph) {
        int n = graph.size(); // number of vertex

        for(int v = 0; v < n; v++) {
            System.out.print("[" + v + "] -> ");
            // if want to acces vth AL -> graph.get(v) -> this is AL
            // want to iterate on this AL to print nbrs of V

            for(int nbr : graph.get(v)) {
                System.out.print(nbr + " ");
            }

            System.out.println();

        }

        // for-each loop???
        // Al : [10 ,20, 30, 40, 50]
        // for(int ele : list) {
        //  System.out.print(ele + " ")
        // }
    public static void main(String args[]) {
        int vtx = 7;
        int[][] edges = {
            {0, 3}, {0, 1}, {2, 3}, {3, 4}, {1, 2}, {4, 5}, {4, 6}, {5, 6}
        };
        // creation of Graph
        ArrayList<ArrayList<Integer>> graph = new ArrayList<>();
        // prepare container to add data in it
        for(int v = 0; v < vtx; v++) {
            graph.add(new ArrayList<>());
        }
        // container of graph is ready,
        // add all the information of edges
        for(int e = 0; e < edges.length; e++) {
            int u = edges[e][0];
            int v = edges[e][1];
            graph.get(u).add(v);
            graph.get(v).add(u);
        }
        int src = 5;
        bfs(graph, src);
        // display(graph);
    }
}
```

How to make weighted graph :-

Vtx=7, Edge=8

AL<AL<Pair>> graph = new AL<>();



	u	v	wt
0	0	3	→ 10 ✓
1	0	1	→ 5 ✓
2	2	3	→ 9 ✓
3	3	4	→ 3 ✓
4	1	2	→ 4 ✓
5	4	5	→ 7 ✓
6	4	6	→ 20 ✓
7	5	6	→ 15 ✓

```
public class Pair {
    int nbr;
    int wt;
    Pair(int nbr, int wt){
        this.nbr = nbr;
        this.wt = wt;
    }
}

for(int v=0; v< Vtx; v++){
    graph.add(new AL<>());
}

for(int e=0; e< edge.length; e++){
    int u = edge[e][0];
    int v = edge[e][1];
    int wt = edge[e][2];
    graph.get(u).add(new Pair(v, wt));
    graph.get(v).add(new Pair(u, wt));
}
```

Expected o/p :

Example

[0] → 1-1, 2-3

[1] → 0-1, 7-3

⋮ etc.

TODO → display.

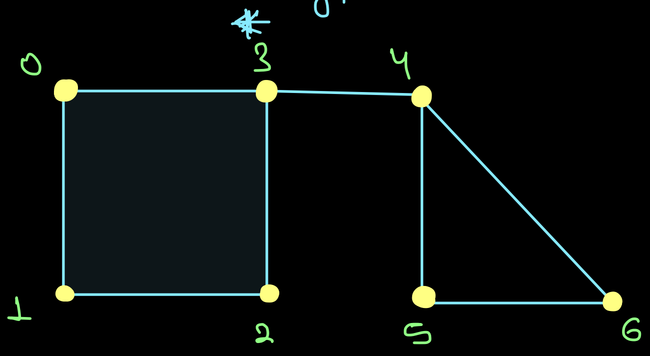
BFS (Breadth First Search) [Traversal of graph]

Exactly same as $\underbrace{\text{level-Order traversal}}$ of tree

d.s. → queue.

undirected graph

starting point



Steps of BFS:

- → Remove
- → work → Print
- → add unvisited neighbour.
  - ↳ mark
  - ↳ Add.

~~3~~ , ~~0~~ , ~~2~~ , ~~4~~ , ~~1~~ , ~~5~~ , ~~6~~

T	T	T	T	T	T	T
~~f~~	~~f~~	~~f~~	~~f~~	~~f~~	~~f~~	~~f~~
0	1	2	3	4	5	6

$\underbrace{3}_{0}$ $\underbrace{0 \quad 2 \quad 4}_{\text{1 dist from 3}}$ $\underbrace{1 \quad 5 \quad 6}_{\text{2 unit dist from 3.}}$

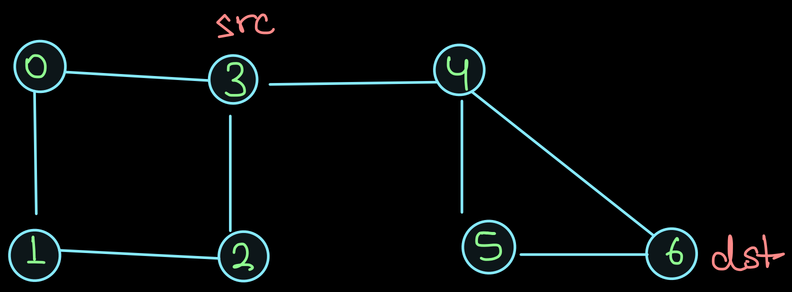
NOTE: Either starting point is given in problem OR we can start traversal from any point.

```java
public static void bfs(ArrayList<ArrayList<Integer>> graph, int src) {
    // number of vertex in graph
    int n = graph.size();
    // Make a visited array i.e. boolean array to mark visit of vertex
    boolean[] vis = new boolean[n];
    // Make a que to add vertex and starting of BFS
    Queue<Integer> qu = new ArrayDeque<>();
    // add source in queue and mark it true i.e. visited
    qu.add(src);
    vis[src] = true;

    while(qu.size() > 0) {
        // remove
        int rem = qu.remove();
        // work -> printing of vertex
        System.out.print(rem + " ");
        // add univisted neighbour
        for(int nbr : graph.get(rem)) {
            // if neighbour is univisted, mark it and add it
            if(vis[nbr] == false) {
                vis[nbr] = true;
                qu.add(nbr);
            }
        }
    }
}
```

Is Path Available from Source to Destination

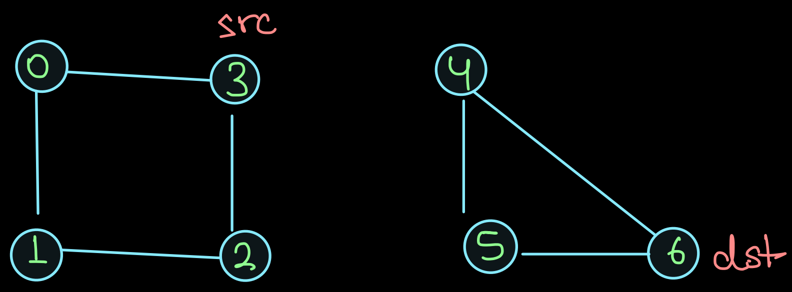
Given an undirected graph, source node and destination node.
Cheek if there is a path Available from source to destination or not.

Ex 1:



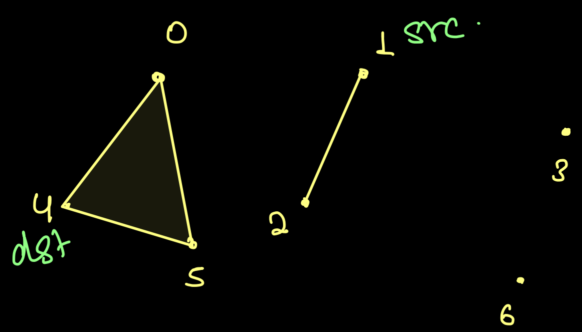
src = 3, dst = 6
ans: true

Ex 2:



src = 3, dst : 6
ans: false.

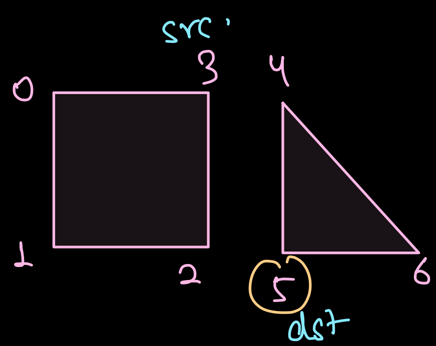
Ex 3:

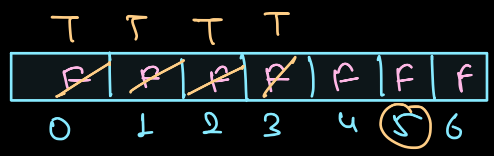


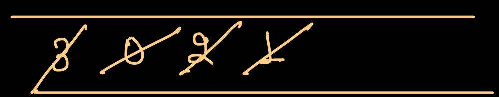
src = 1, dst = 4

ans: false.

Solution: Start BFS Algo from source point, & in End finally check status of vis[dst];







Remove
Add unvisited
nbr

→ return vis[dst];