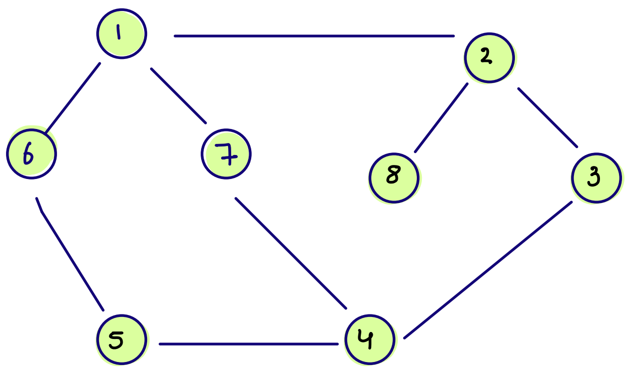
Agenda

1) Introduction
2) Types of graph
3) How to store
4) BFS (Breadth First traversal)
5) Is path from source to destination

Intro



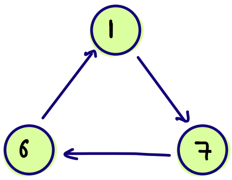
graph contains vertices and edges ,

v = 8

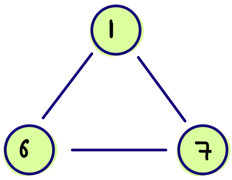
e = 9

Types / classification of graphs

i) Based on type of edges.

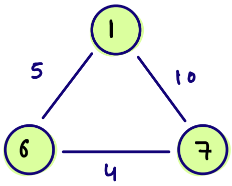


directed graph
(instagram follow)

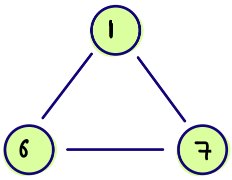


undirected graph
(Facebook, LinkedIn)

ii) Based on edge wt present or not

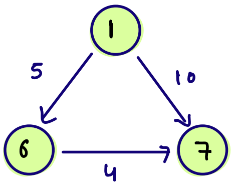


weighted graph



unweighted graph

*) combination of above two types are also possible.



directed weighted graph

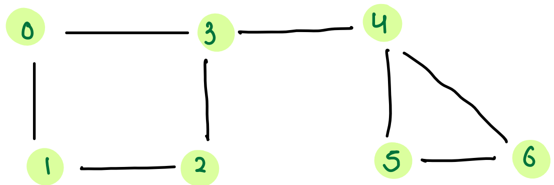
Storing a graph

Two famous ways are:

i) Adjacency matrix

ii) Adjacency List

1) Adjacency matrix representation



vtx = 7 e = 8

edges →

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

undirected graph

matrix → v x v

$$\begin{array}{c|ccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

graph

```
u = edges[i][0]

v = edges[i][1]

// edge b/w u and v

graph[u][v] = 1;
graph[v][u] = 1;
```

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

graph

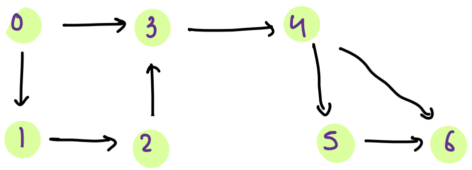
vtx = 7   e = 8

edges →

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

←

directed graph



```
u = edges[i][0]

v = edges[i][1]

// edge from u to v

graph[u][v] = 1;
```

// directed weighted graph

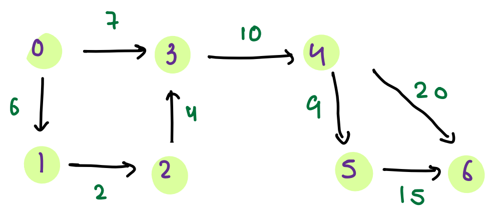
vtx = 7   e = 8

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

graph

edges →

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



```
u = edges[i][0];
v = edges[i][1];
wt = edges[i][2];
//edge from u to v of wt
graph[u][v] = wt;
```

disadvantage of Adjacency matrix representation

→ waste of space

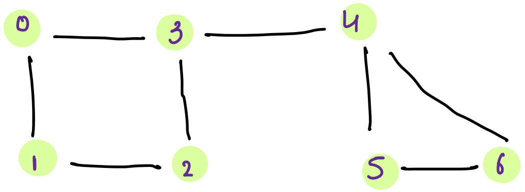
↓

Due to this majorly Adjacency list is used

2) Adjacency List representation

```
AL<AL<Integer>> graph;
```

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



vtx = 7 e = 8

edges →

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

undirected graph

```
u = edges[i][0];
v = edges[i][1];
// edge b/w u and v
graph.get(u).add(v)
graph.get(v).add(u)
```

---

vtx = 7 e = 8

edges →

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

directed graph

```
u = edges[i][0];
v = edges[i][1];
// edge from u to v
graph.get(u).add(v)
```

// directed weighted graph

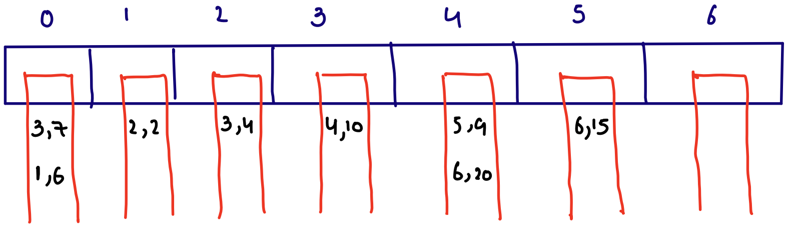
↳ AL < AL < Pair > > graph;

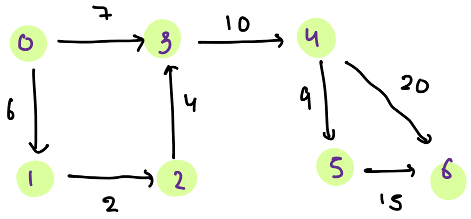
↳ class Pair {
    int v;
    int wt;
}

vtx = 7   e = 8

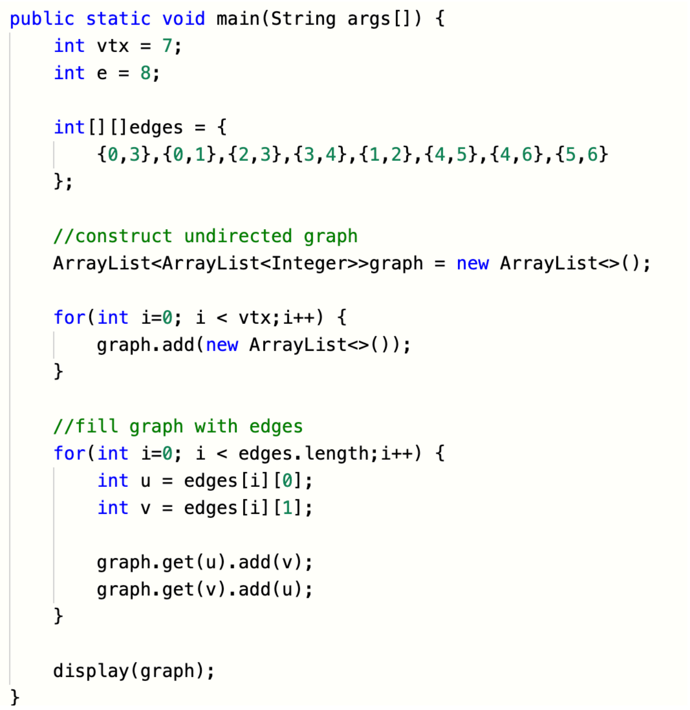
edges →

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





```
u = edges[i][0];
v = edges[i][1];
wt = edges[i][2];
// edge from u to v of wt
Pair np = new Pair(v,wt);
graph.get(u).add(np);
```



```
public static void main(String args[]) {
    int vtx = 7;
    int e = 8;

    int[][]edges = {
        {0,3},{0,1},{2,3},{3,4},{1,2},{4,5},{4,6},{5,6}
    };

    //construct undirected graph
    ArrayList<ArrayList<Integer>>graph = new ArrayList<>();

    for(int i=0; i < vtx;i++) {
        graph.add(new ArrayList<>());
    }

    //fill graph with edges
    for(int i=0; i < edges.length;i++) {
        int u = edges[i][0];
        int v = edges[i][1];

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

    display(graph);
}
```

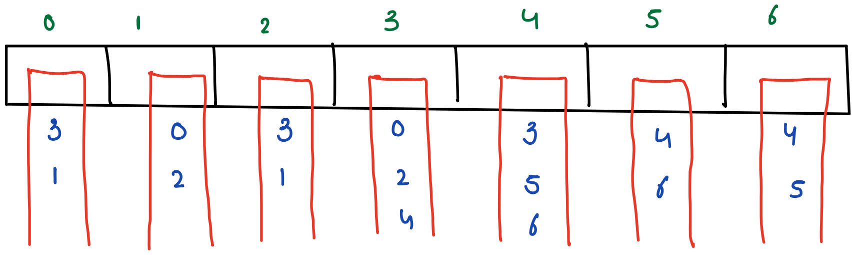
vtx = 7   e = 8

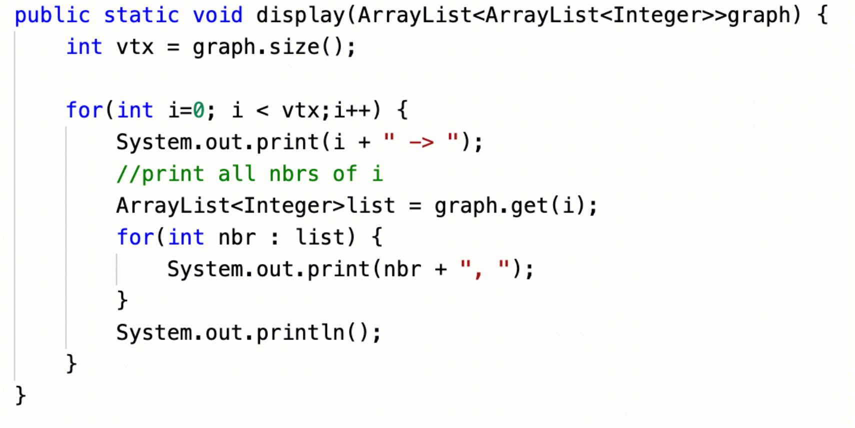
edges →

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

undirected graph

graph :





```
public static void display(ArrayList<ArrayList<Integer>>graph) {
    int vtx = graph.size();

    for(int i=0; i < vtx;i++) {
        System.out.print(i + " -> ");
        //print all nbrs of i
        ArrayList<Integer>list = graph.get(i);
        for(int nbr : list) {
            System.out.print(nbr + ", ");
        }
        System.out.println();
    }
}
```

0 → 3 1

1 → 0 2

2 → 3 1

3 → 0 2 4

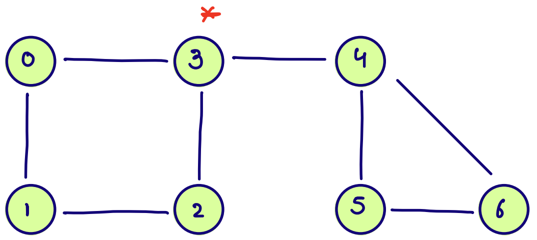
4 → 3 5 6

5 → 4 6

6 → 4 5

Traversal on graph

↳ BFS (Breadth first search)   { levelorder in trees }



start with 3

3 0 2 4 1 5 6

purpose: just to go on every vertex

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

visited:

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

→ remove

→ print (work)

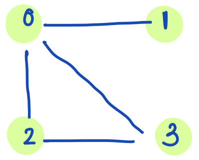
→ add unvisited nbr

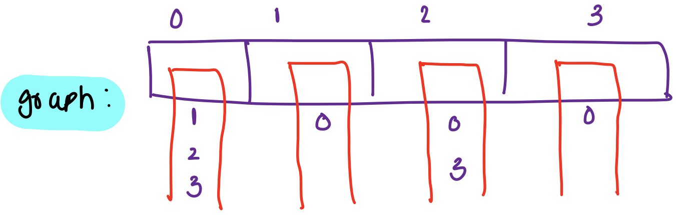
note: Either the src (where to start from) will be given to you or you can start from any valid vertex.

```
void  BFS ( AL <AL < Integer>> graph, int  src)  {

    Queue <Integer> q  = new ArrayDeque<>();
    boolean [] vis =  new  boolean [ graph.size()];

    q.add (src);
    vis [src] = true;

    while  (q.size() > 0)  {
        int  rmv = q.remove();
        sop (rmv);

        // add  unvisited  nbr  of  rmv
        AL <Integer> list = graph.get(rmv);

        for (int nbr : list)  {
            if (vis[nbr] == false)  {
                q. add (nbr);
                vis [nbr] = true;
            }
        }
    }
}
```

```
void  BFS ( AL <AL < Integer>> graph,  int  src)  {

  Queue < Integer> q  =  new  ArrayDeque <>();
  boolean [] vis =  new  boolean [ graph.size() ];

  q.add (src);
  vis [src] = true;

  while  (q.size() > 0)  {
  |  1)  int   rmv = q.remove();
  |
  |  2)  sop  (rmv) ;
  |
  |  3)  // add  unvisited  nbr  of  rmv
  |      AL < Integer > list = graph.get (rmv) ;
  |
  |      for  ( int  nbr : list )  {
  |      |      if  (vis [nbr]  == false)  {
  |      |      |      q. add  (nbr);
  |      |      |      vis [nbr] =  true ;
  |      |      }
  |      }
  }
}
```

src = 1



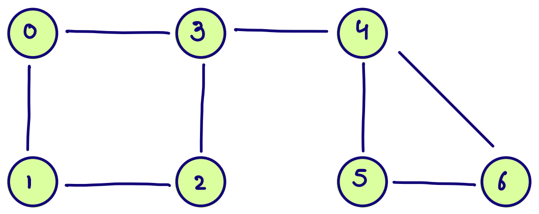


q	~~1~~	~~0~~	~~2~~	~~3~~

vis:	T	T	T	T
	0	1	2	3

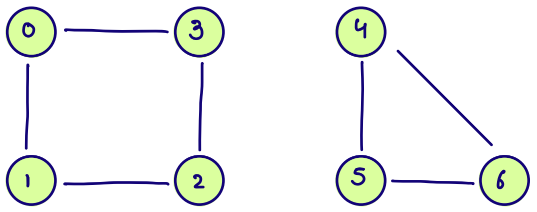
o/p: 1 0 2 3

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



src = 3 dest = 6

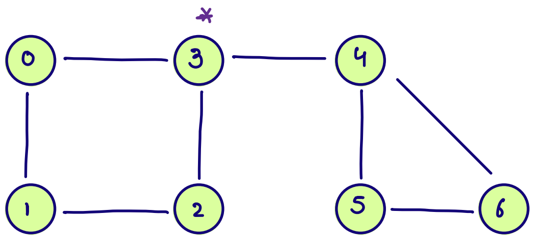
ans = true



src = 3 dest = 6

ans = false

BFS from src and then check the value
↳ vis[dest]



src=3 dest = 6

→ remove

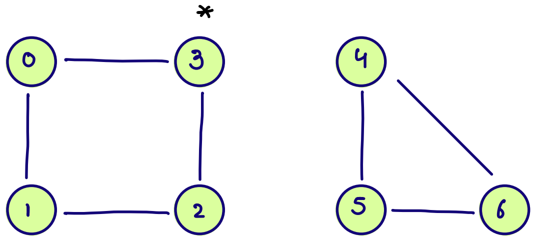
→ add unvisited nbr

q:	~~3~~	~~0~~	~~2~~	~~4~~	~~1~~	~~5~~	~~6~~

ans→ true

vis[6] ⇒ true

vis:	T	T	T	T	T	T	T
	0	1	2	3	4	5	6



src=3 dest = 6

→ remove

→ add unvisited nbr

q:	~~3~~	~~0~~	~~2~~	~~1~~

ans = false

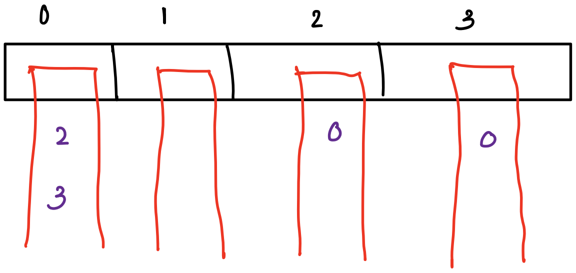
vis[6] ⇒ false

vis:	T	T	T	T	F	F	F
	0	1	2	3	4	5	6

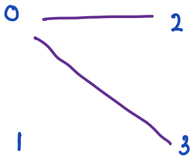
Doubts

vtx = 4   e = 2

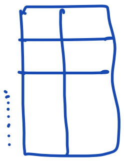
0	2
0	3

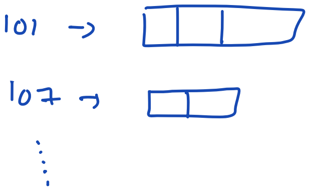


undirected



vtx = 4 , named : 101, 107, 57, 89   and then edges





HashMap < Integer, AL < Integer>> graph