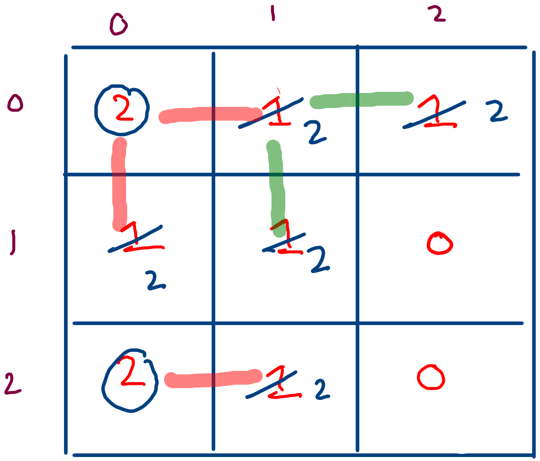
Rotting Oranges

`[[2,1,1],[1,1,0],[0,1,1]]`

ans: 2



	0	1	2
0	2	~~1~~ 2	~~1~~ 2
1	~~1~~ 2	~~1~~ 2	0
2	2	~~1~~ 2	0

(i) multiple src

~~0,0,0~~	~~2,0,0~~	~~0,1,1~~	~~1,0,1~~	~~2,1,1~~	~~0,2,2~~	~~1,1,2~~
0		1			2	

```
Pair {
        int i;
        int j;
}       int t;
```

BFS

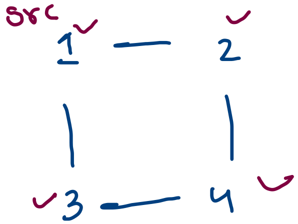
remove

mark*

work

add unvisited nbr

marking on removal



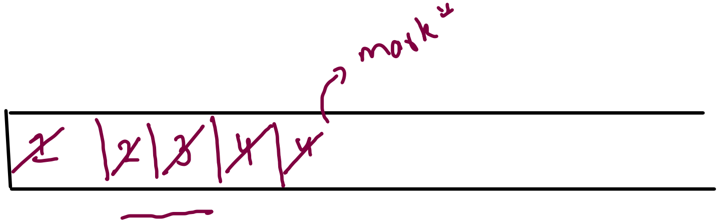
remove

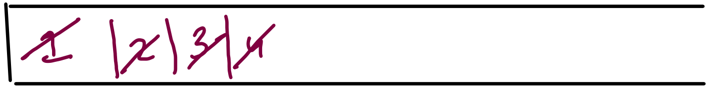
work

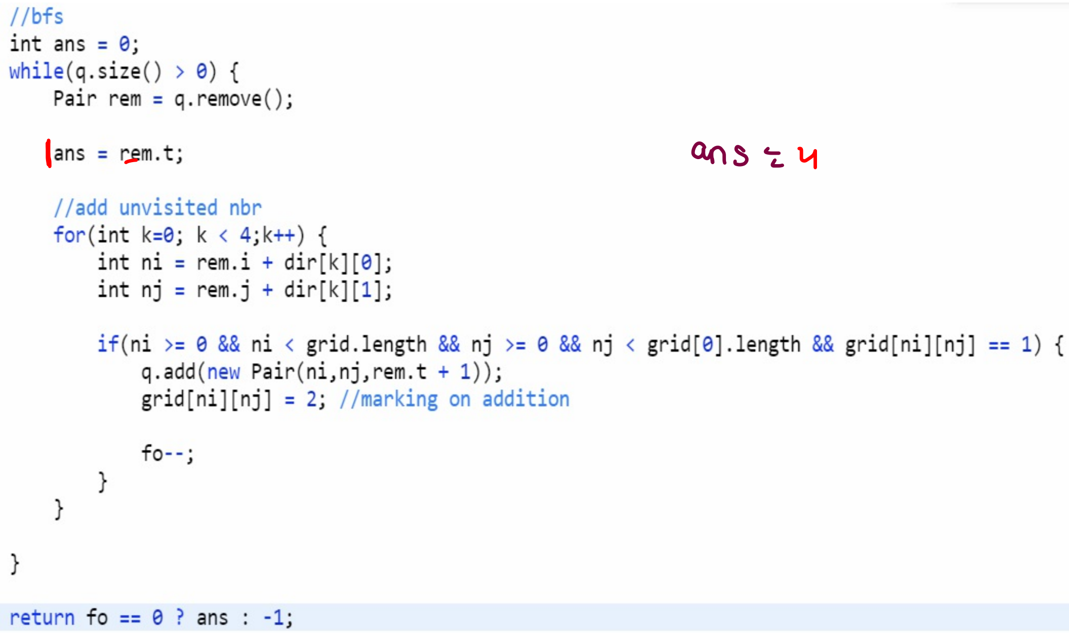
add unvisited

→ mark nbr

marking on addition







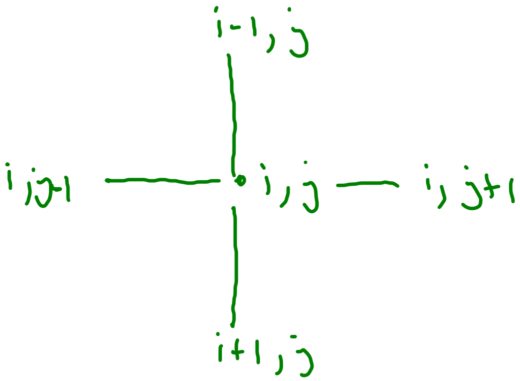
```
//bfs
int ans = 0;
while(q.size() > 0) {
    Pair rem = q.remove();

    ans = rem.t;

    //add unvisited nbr
    for(int k=0; k < 4;k++) {
        int ni = rem.i + dir[k][0];
        int nj = rem.j + dir[k][1];

        if(ni >= 0 && ni < grid.length && nj >= 0 && nj < grid[0].length && grid[ni][nj] == 1) {
            q.add(new Pair(ni,nj,rem.t + 1));
            grid[ni][nj] = 2; //marking on addition

            fo--;
        }
    }

}

return fo == 0 ? ans : -1;
```

ans = 4

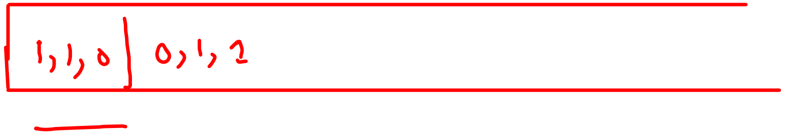
	0	1	2	3
0	0	2	0	~~1~~ 2
1	~~1~~ 2	~~1~~ 2	~~1~~ 2	~~1~~ 2
2	2	~~1~~ 2	0	0
3	0	2 ~~1~~	0	0

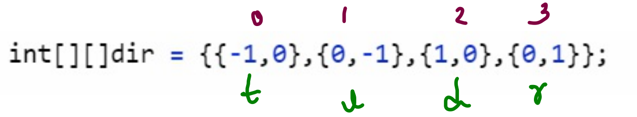
fo = ~~7~~ ~~6~~ ~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ 0

~~0,1,0~~	~~2,0,0~~	~~1,1,1~~	~~1,0,1~~	~~2,1,1~~	~~1,2,2~~	~~3,1,2~~	~~1,3,3~~	~~0,3,4~~
0		1			2		3	4

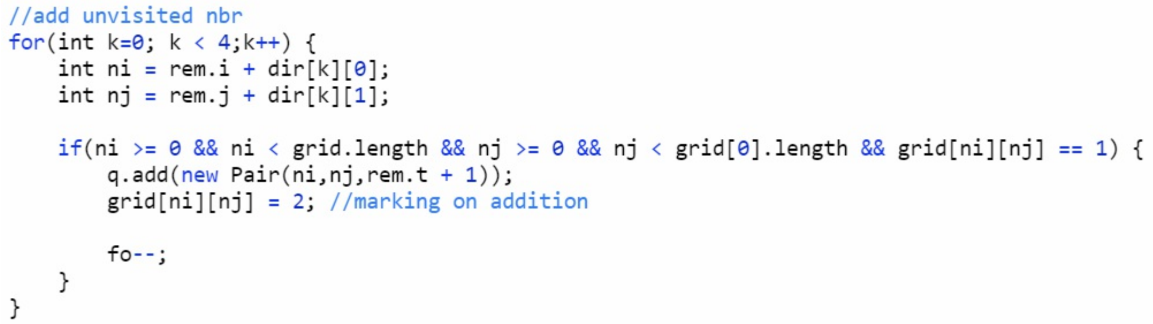


	0	1	2
0	0	1	0
1	1	2	1
2	0	1	0



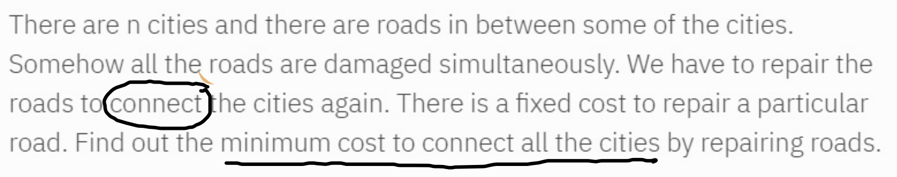


```
int[][]dir = {{-1,0},{0,-1},{1,0},{0,1}};
```



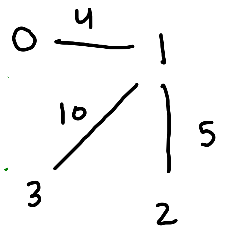
```
//add unvisited nbr
for(int k=0; k < 4;k++) {
    int ni = rem.i + dir[k][0];
    int nj = rem.j + dir[k][1];

    if(ni >= 0 && ni < grid.length && nj >= 0 && nj < grid[0].length && grid[ni][nj] == 1) {
        q.add(new Pair(ni,nj,rem.t + 1));
        grid[ni][nj] = 2; //marking on addition

        fo--;
    }
}
```

Minimum Cost To Connect All Cities



There are n cities and there are roads in between some of the cities. Somehow all the roads are damaged simultaneously. We have to repair the roads to connect the cities again. There is a fixed cost to repair a particular road. Find out the minimum cost to connect all the cities by repairing roads.

0 — 1	4
1 — 2	5
1 — 3	10
0 — 3	15

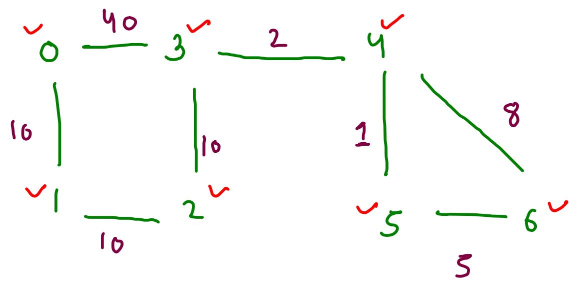


MST

(Prims)

Dijkstra : (i) shortest path (single src all dest)

(ii) failure : fails on -ve wt (greedy)



remove

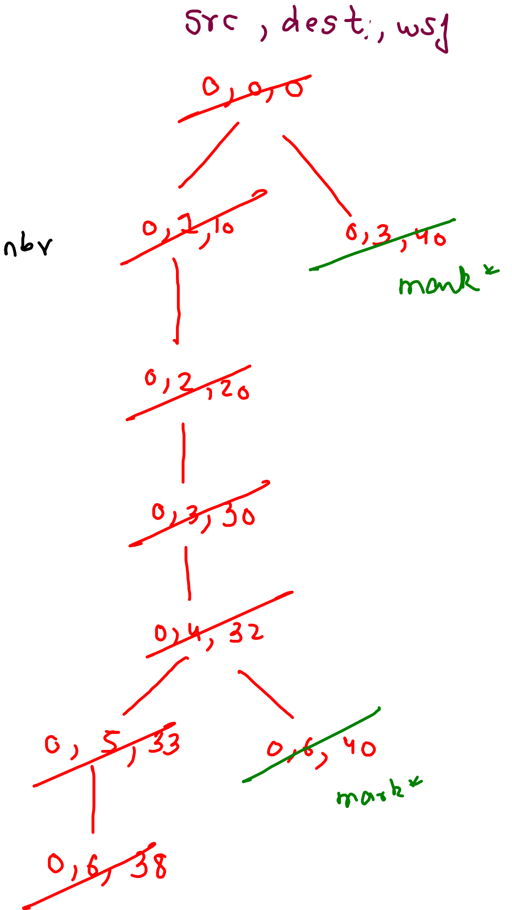
mark*

work

add unvisited nbr

0	10	20	30	32	33	38
0	1	2	3	4	5	6

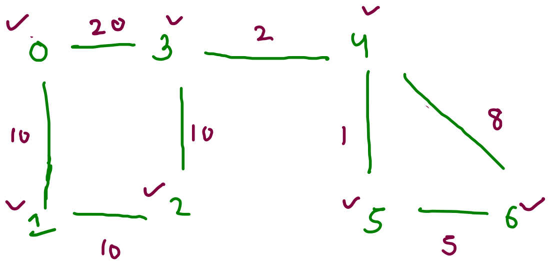
src to 4 ans



src = 0

u, v, wsf

Prims : (i) MST (minimum spanning tree)

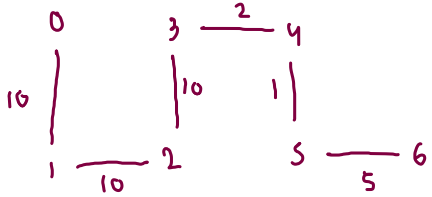


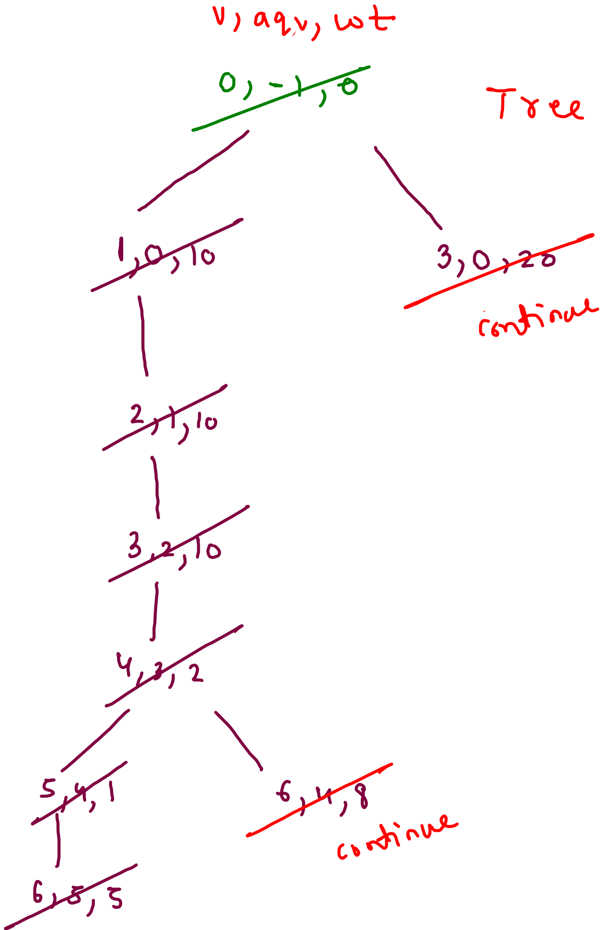
remove

mark*

work

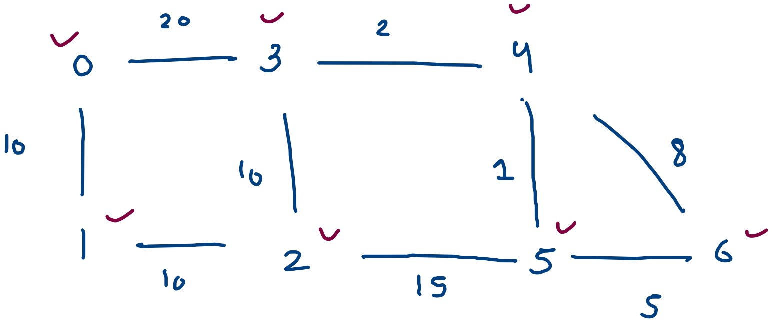
add unvisited nbr



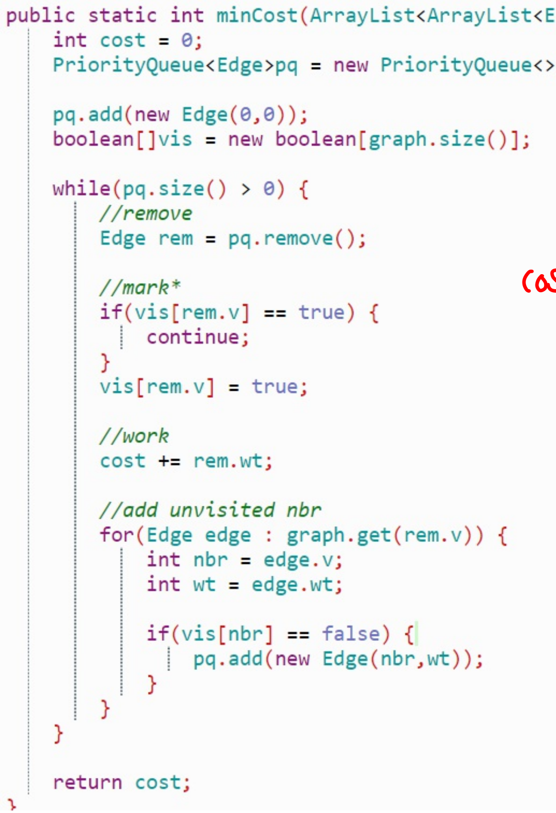


Tree : A connected acyclic graph is a tree.

min cost to connect all cities

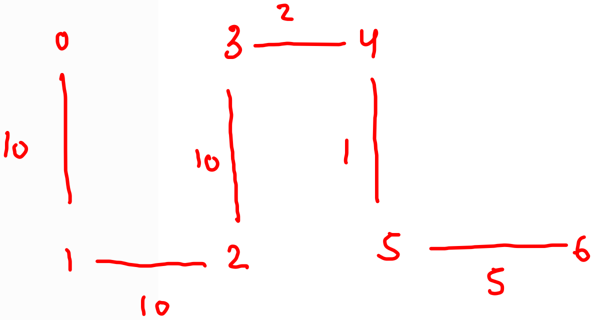


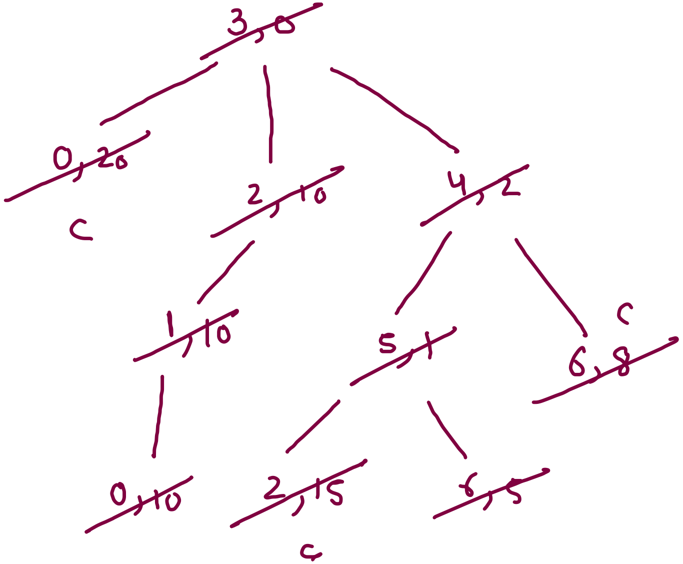
v, wt



```
public static int minCost(ArrayList<ArrayList<Edge>>graph) {
    int cost = 0;
    PriorityQueue<Edge>pq = new PriorityQueue<>();

    pq.add(new Edge(0,0));
    boolean[]vis = new boolean[graph.size()];

    while(pq.size() > 0) {
        //remove
        Edge rem = pq.remove();

        //mark*
        if(vis[rem.v] == true) {
            continue;
        }
        vis[rem.v] = true;

        //work
        cost += rem.wt;

        //add unvisited nbr
        for(Edge edge : graph.get(rem.v)) {
            int nbr = edge.v;
            int wt = edge.wt;

            if(vis[nbr] == false) {
                pq.add(new Edge(nbr,wt));
            }
        }
    }

    return cost;
}
```

cost : 2+1+5+10+10+10

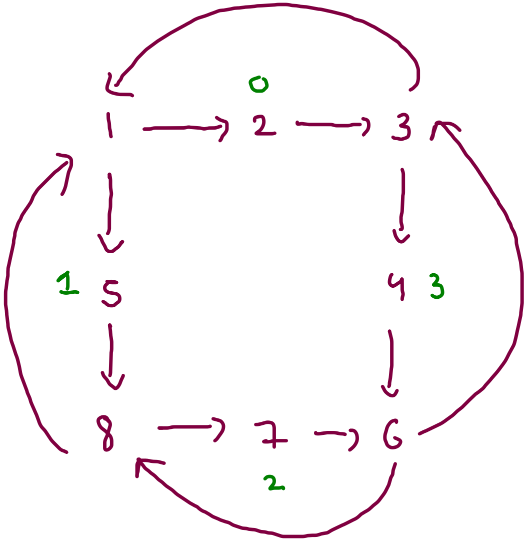




815. Bus Routes

[[1, 2, 3] , [1, 5, 8], [8, 7, 6], [3, 4, 6]]

0 1 2 3



src bs = 2

dest bs = 6

$2 \xrightarrow{0} 1 \xrightarrow{1} 8 \xrightarrow{2} 6$ (buses : 3)

$2 \xrightarrow{0} 3 \xrightarrow{3} 6$ (buses: 2) routes: [[1, 2, 3] , [1, 5, 8], [8, 7, 6], [3, 4, 6]]

0 1 2 3

1 -> 0, 1

2 -> 0

3 -> 0, 3

4 -> 3

5 -> 1

6 -> 2, 3

7 -> 2

8 -> 1, 2

map

map: bus stand vs bus

Integer vs AL <Integer>

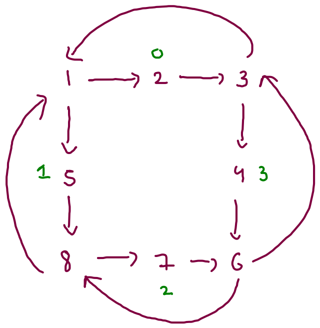
bfs, marking on addition

normal BFS

vis: bus stand

vis : buses

```
Pair {
  int bus_stand;
  int lev;
}
```



routes : [[1, 2, 3] , [1, 5, 8], [8, 7, 6], [3, 4, 6]]

0 1 2 3

1 -> 0, 1

2 -> 0

3 -> 0, 3

4 -> 3

5 -> 1

6 -> 2, 3

7 -> 2

8 -> 1, 2

map

vis -> bus stand : 2, 1, 3, 5, 8, 4, 6, 7

bus : 0, 1, 3, 2

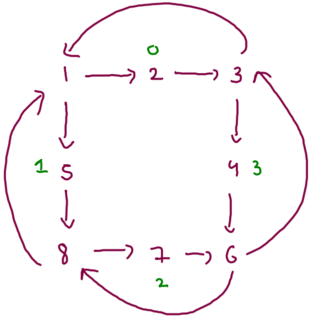
src = 2

dest = 6

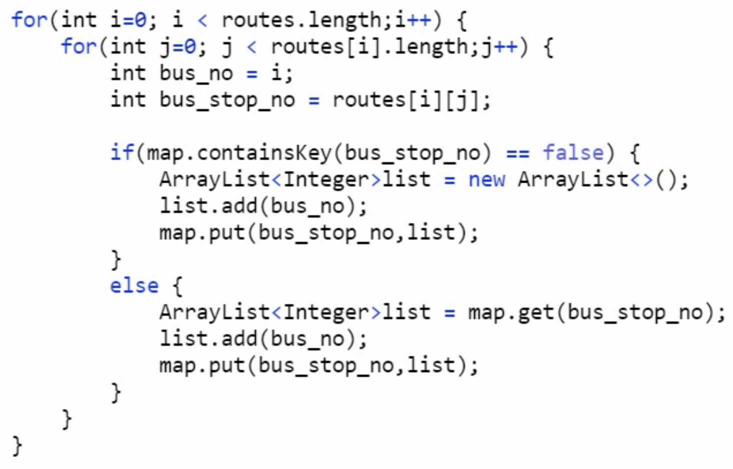
```
Pair {
  bus stand;
  int lev;
}
```

ans

~~2,0~~	~~1,1~~	~~3,1~~	~~5,2~~	~~8,2~~	~~4,2~~	~~6,2~~	7, 3
0	1		2				3



map : bus stand vs bus

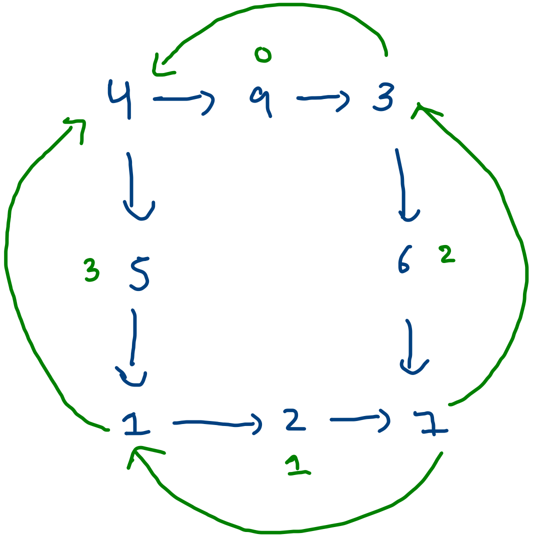


```
for(int i=0; i < routes.length;i++) {
    for(int j=0; j < routes[i].length;j++) {
        int bus_no = i;
        int bus_stop_no = routes[i][j];

        if(map.containsKey(bus_stop_no) == false) {
            ArrayList<Integer>list = new ArrayList<>();
            list.add(bus_no);
            map.put(bus_stop_no,list);
        }
        else {
            ArrayList<Integer>list = map.get(bus_stop_no);
            list.add(bus_no);
            map.put(bus_stop_no,list);
        }
    }
}
```

routes :

[[4, 9, 3], [1, 2, 7], [3, 6, 7], [4, 5, 1]]

0 1 2 3



bus stop vs buses

4 -> 0, 3

9 -> 0

3 -> 0, 2

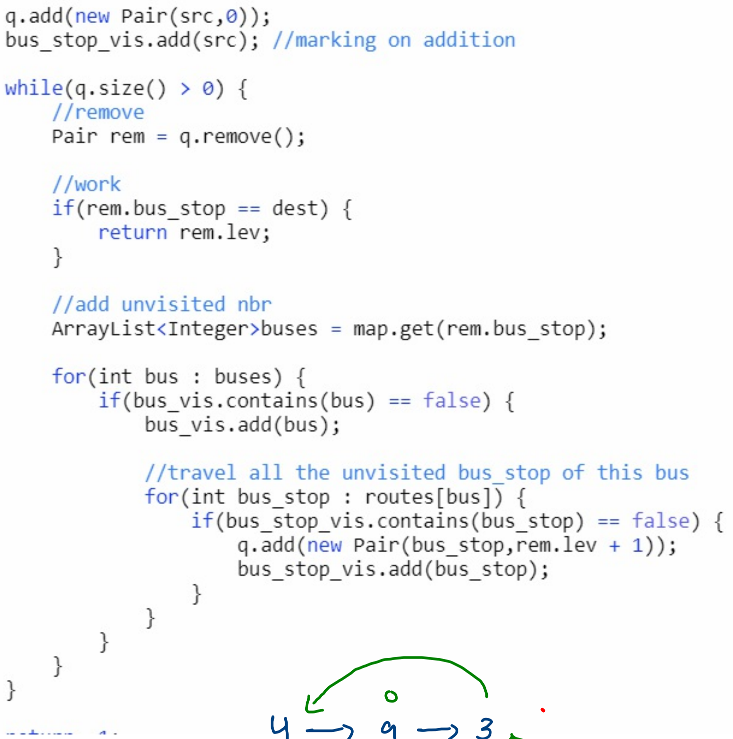
1 -> 1, 3

2 -> 1

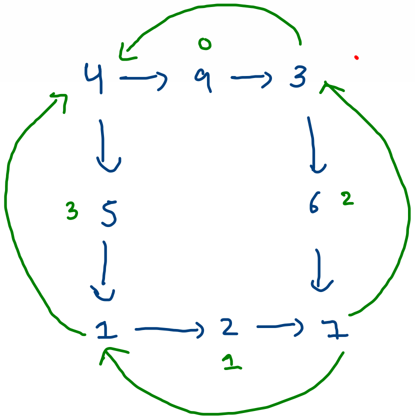
7 -> 1, 2

6 -> 2

5 -> 3



```
q.add(new Pair(src,0));
bus_stop_vis.add(src); //marking on addition

while(q.size() > 0) {
    //remove
    Pair rem = q.remove();

    //work
    if(rem.bus_stop == dest) {
        return rem.lev;
    }

    //add unvisited nbr
    ArrayList<Integer>buses = map.get(rem.bus_stop);

    for(int bus : buses) {
        if(bus_vis.contains(bus) == false) {
            bus_vis.add(bus);

            //travel all the unvisited bus_stop of this bus
            for(int bus_stop : routes[bus]) {
                if(bus_stop_vis.contains(bus_stop) == false) {
                    q.add(new Pair(bus_stop,rem.lev + 1));
                    bus_stop_vis.add(bus_stop);
                }
            }
        }
    }
}
```



4 -> 0, 3

9 -> 0

3 -> 0, 2

1 -> 1, 3

2 -> 1

7 -> 1, 2

6 -> 2

5 -> 3

bus stop vs buses

bus_ stop - vis

3, 4, 9
6, 7, 2

bus_vis

0, 2, 3
1

src = 3

dest = 5

ans = 2

~~3,0~~	~~4,1~~	~~9,1~~	~~6,1~~	~~7,1~~	~~5,2~~	1, 2
0	1				2	