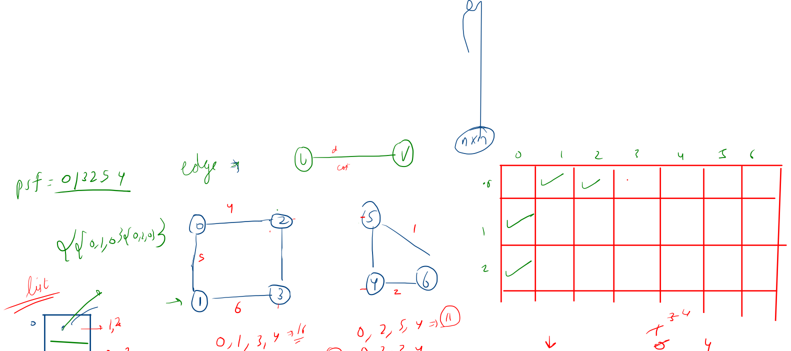
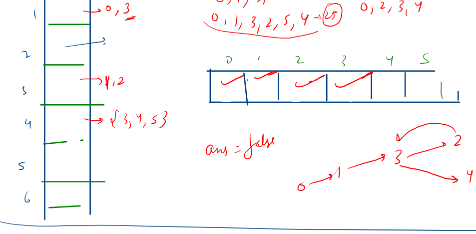
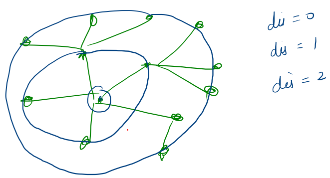
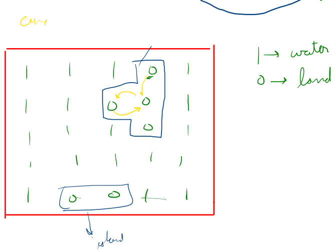


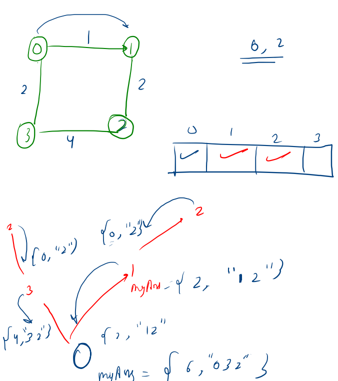



```java
public static boolean hasPath(int src, int des, boolean[] vis, ArrayList<Edge>[] graph){
        if(src==des){
            return true;
        }


        vis[src]=true;
        boolean ans = false;


        for(Edge e:graph[src]){
            int nbr=e.v;


            if(vis[nbr]==false){
                ans = ans || hasPath(nbr, des, vis, graph);
            }
        }
    }
```

DFS







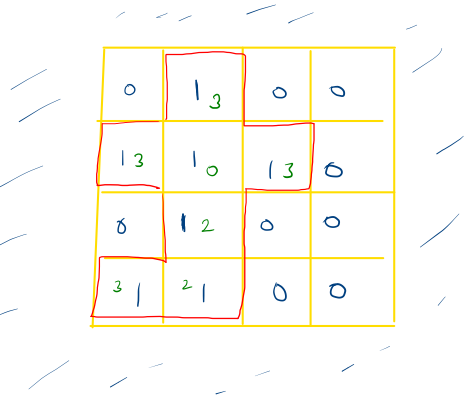
2 → 2

0, "2"

```java
public static Pair heaviestPath(int src, int des, boolean[] vis, ArrayList<Edge>[] graph){
        if(src==des){
            return new Pair(0,src+"");
        }
        vis[src]=true;


        Pair myAns= new Pair();


        for(Edge e:graph[src]){
            int nbr=e.v;


            if(!vis[nbr]){
                Pair recAns = heaviestPath(nbr, des, vis, graph);


                if(recAns.wsf + e.w > myAns.wsf){
                 myAns.psf=src + recAns.psf;
                 myAns.wsf=recAns.wsf + e.w;
              }
            }
        }
```

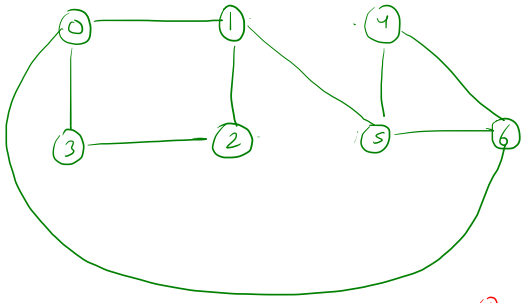
wsf, psf

[[0,1,0,0],[1,1,1,0],[0,1,0,0],[1,1,0,0]]



16

3 + 3 + 3 + 3 + 2 + 2 = 16



Hamiltonian path

Hamiltonian cycle

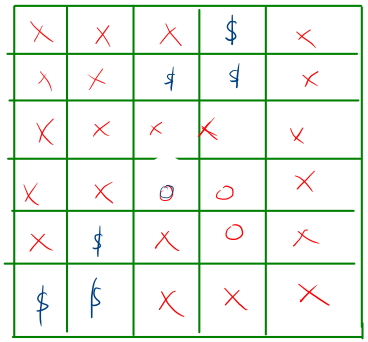
edgecount == n-1

n => vertices

0, 3, 2, 1, 5, 4, 6 → cycle

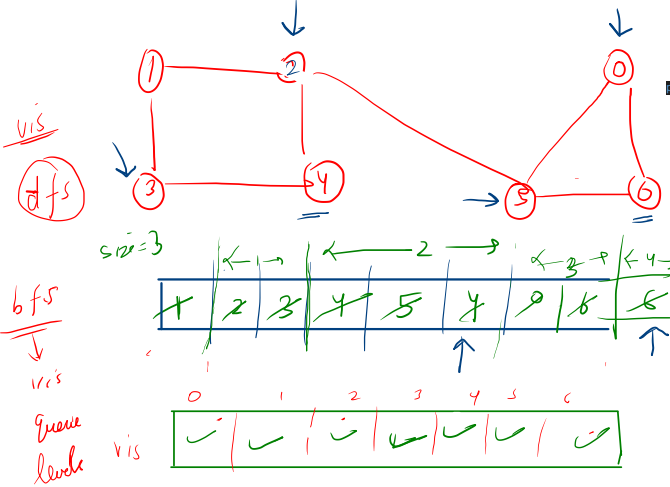
0, 3, 2, 1, 5, 6, 4 → path

0, 6, 4, 5, 1, 2, 3 → cycle

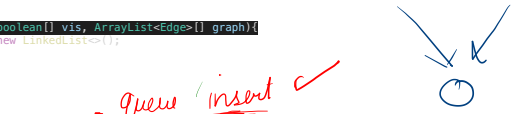


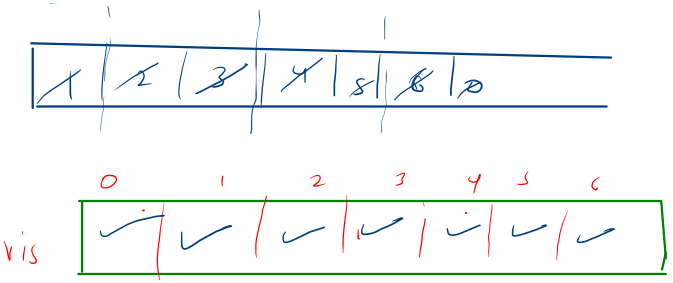
$ → 'O'

'O' → 'X'




```
public static void BFS(int src, boolean[] vis, ArrayList<Edge>[] graph){
    LinkedList<Integer> que=new LinkedList<>();

    que.addLast(src);
    int level=0;

    while(que.size()>0){
        int size=que.size();

        System.out.print("This is level "+level+" -> ");

        while(size-->0){
            int vtx = que.removeFirst();

            if(vis[vtx]) continue;

            System.out.print(vtx);
            vis[vtx]=true;

            for(Edge e: graph[vtx]){
                if(vis[e.v]==false){
                    que.addLast(e.v);
                }
            }
        }

        level++;
    }
}
```





0 → 1
1 → 2, 3
2 → 4, 5
3 → 0, 6
4 →

$idx = i * m + j$

$m = 4$

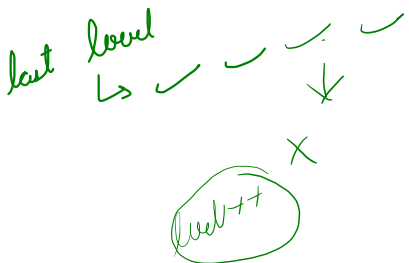
$n = 3$

0	1	2	3
4	5 (1,1)	6	7
8	9	10 (2,2)	11

$1 * 4 + 1 \Rightarrow 5$

$2 * 4 + 2 \Rightarrow 10$

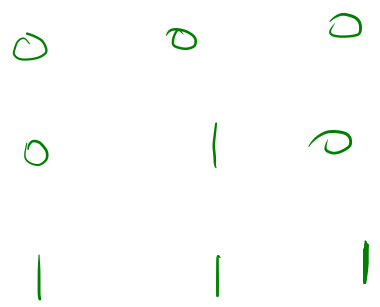
last level

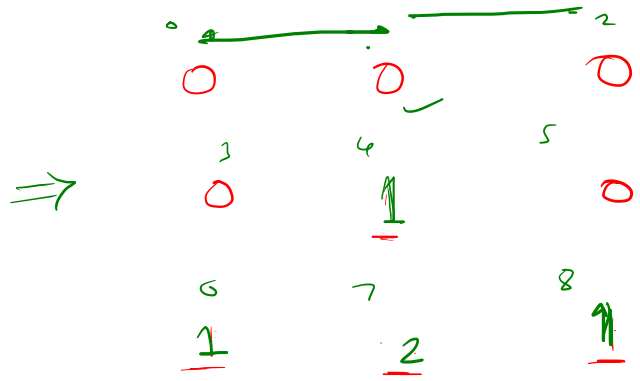


idx

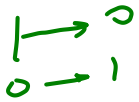
$i = idx / m$

$j = idx \% m$

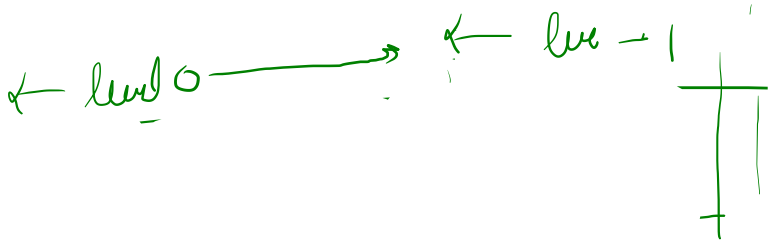


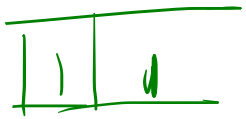


n x m

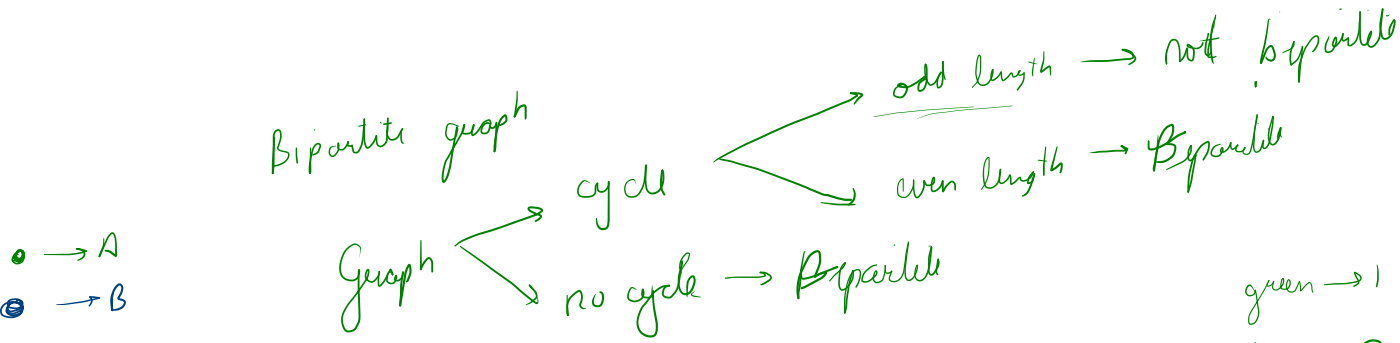


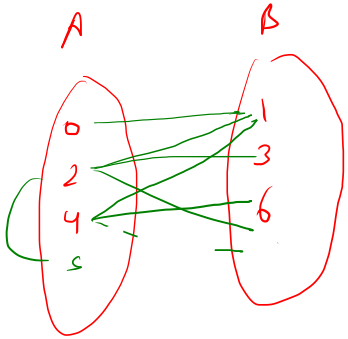
bfs answers min. distance



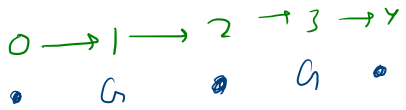


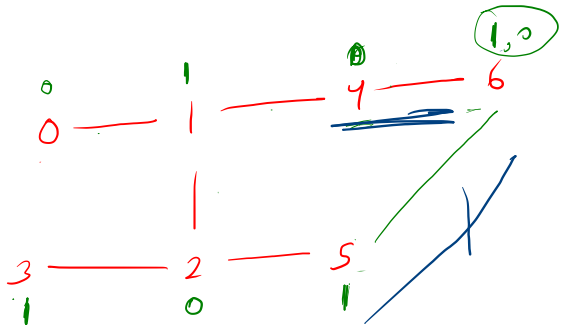
insertion time = removal time ⟹ costly





green → 1
blue → 0
unvisited → -1

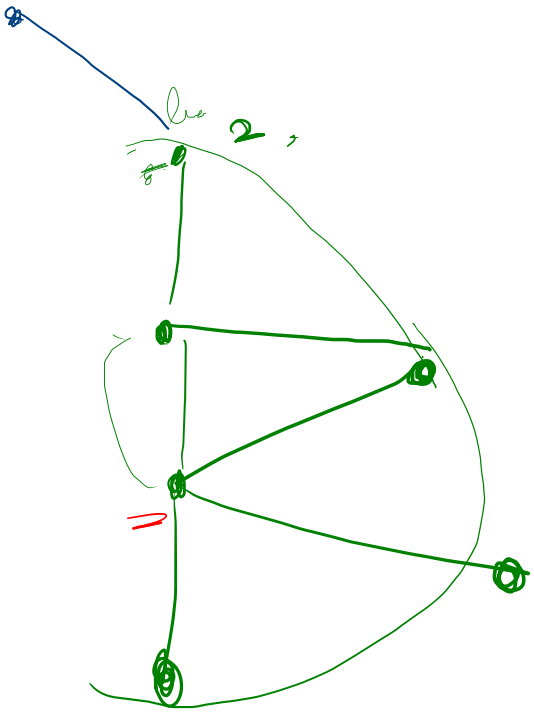


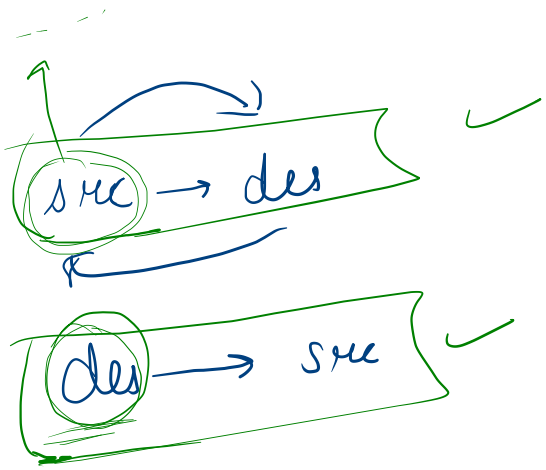



```
public boolean isOddCycle(int src, int[][] graph, int[] vis){
    LinkedList<Integer> que = new LinkedList<>();
    que.addLast(src);
    vis[src] = 0;

    int color = 0; // level
    while(que.size()>0){
        int size = que.size();

        while(size -- > 0){
            int u = que.removeFirst();

            if(vis[u]!=color) return true;
            vis[u] = color;
            for(int v : graph[u]){
                if(vis[v]==-1){
                    que.addLast(v);
                    vis[v] = 1-color;
                }
            }
        }
        color = (color+1)%2;
    }

    return false;
}
```

src → des

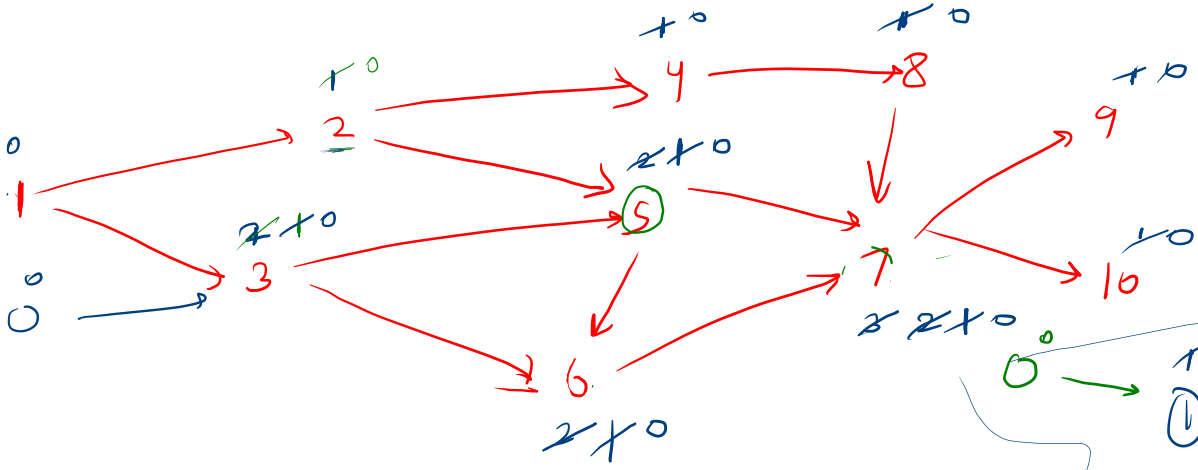
1 → ②

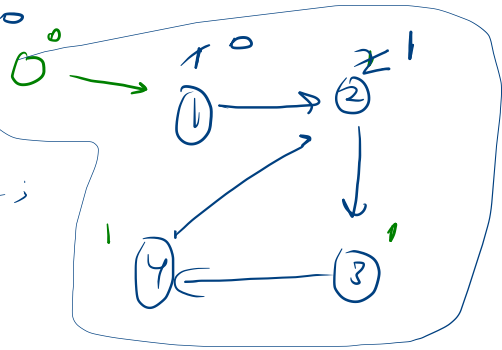




0 ⇒ 2

Topological sort





number of incoming edges

indegree(0)

indegree(nbrs)--;

1, 0, 2, 3, 4, 5, 8, 6, 7, 9, 10

0, 1