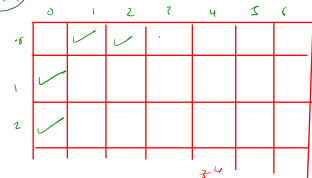
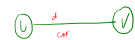
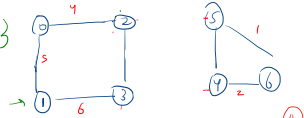


edge  $\rightarrow$

$$\{ \sqrt{0,1}, \sqrt{0,2}, \sqrt{0,3} \}$$


```

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

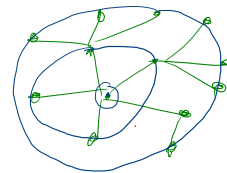
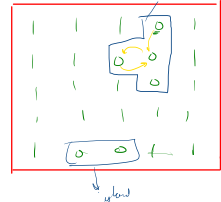
    vis[src]=true;
    boolean ans = false;

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

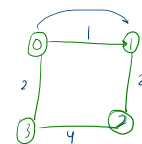
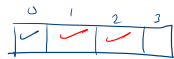
        if(vis[nbr]==false){
            ans = ans || hasPath(nbr, des, vis, graph);
        }
    }
}

```

DFS


$$\begin{aligned} d_{ii} &= 0 \\ d_{ij} &= 1 \\ d_{ij} &= 2 \end{aligned}$$


1 → water  
0 → land


$$\underline{\underline{6,2}}$$
 $2 \rightarrow 2$ 

0, "2"

```

1  public static void heaviestPath(int src, int des, int[][] vis, ArrayList<Edge>[] graph){
2      if(src == des)
3          return new Pair(0,src+"->"+des);
4      vis[src]=true;
5
6      Pair ans=null;
7
8      for(Edge e:graph[src]){
9          int nbr=e.v;
10
11          if(!vis[nbr]){
12              Pair rec= heaviestPath(nbr, des, vis, graph);
13
14              if(recdes<=e.w+rec.w) ans=rec;
15
16              if(recdes>e.w+rec.w) ans=new Pair(e.w+rec.w,src+"->"+nbr);
17          }
18      }
19  }
20
21  }

```

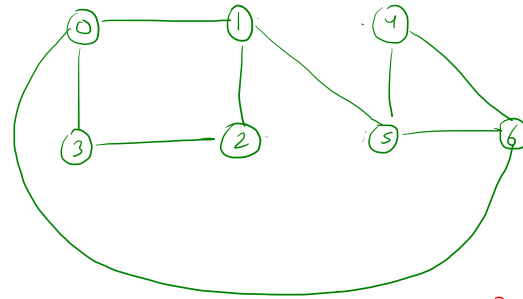
wsf, psf

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

0	1 <sub>3</sub>	0	0
1 <sub>3</sub>	1 <sub>0</sub>	1 <sub>3</sub>	0
0	1 <sub>2</sub>	0	0
3 <sub>1</sub>	2 <sub>1</sub>	0	0

(16)

$$3+3+3+3+2+2 \Rightarrow 16$$



Hamiltonian path  
Hamiltonian cycle

$$\text{edgcount} = n - 1$$

↓                      ↓  
0, 3, 2, 1, 5, 4, 6 → cycle  
0, 3, 2, 1, 5, 6, 4 → path  
0, 6, 4, 5, 1, 2, 3 → cycle

⑦ ⇒ vertices

X	X	X	\$	X
X	X	\$	\$	X
X	X	X	X	X
X	X	0	0	X
X	\$	X	0	X
\$	\$	X	X	X

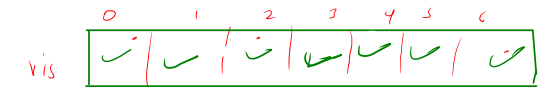
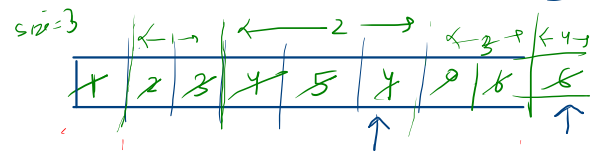
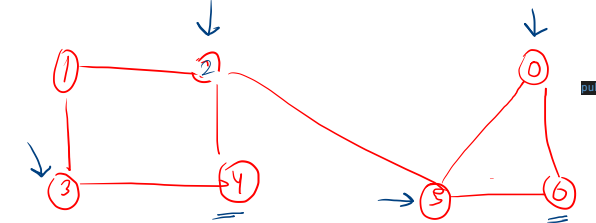
⑦

\$ → '0'  
'0' → 'X'

vis  
dfs

bfs  
vis

Queue  
level



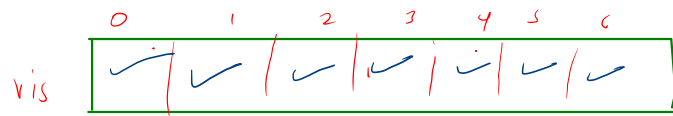
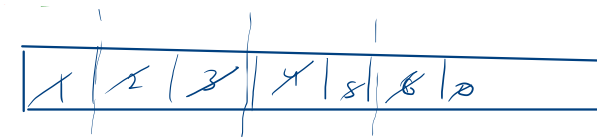
0 → 1  
1 → 2, 3  
2 → 4, 5  
3 → 0, 6  
4 →

```
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++;
        }
    }
}
```

Queue insert ✓  
Queue remove ✓



$$idx = i * m + j$$

$$m = 4$$

$$n = 3$$

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

$$1 * 4 + 1 \Rightarrow 5$$

$$2 * 4 + 2 \Rightarrow 10$$

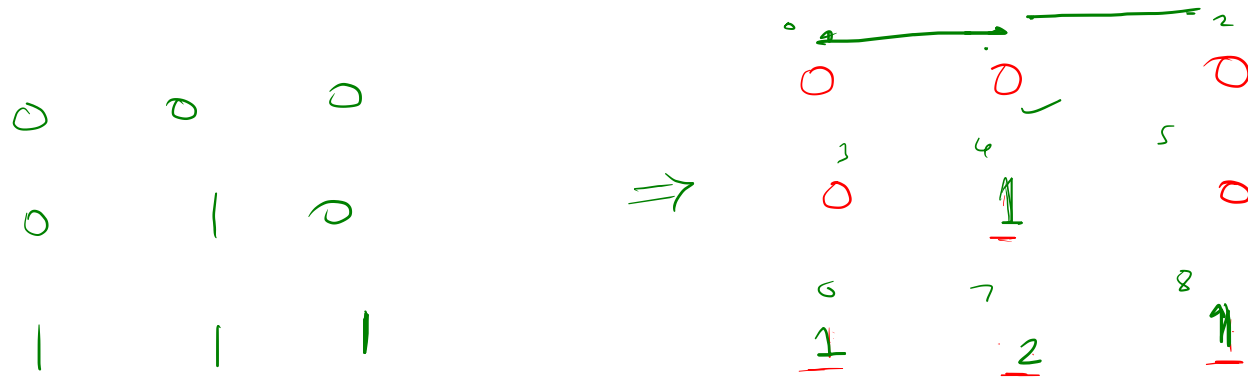
last level  
↳ ✓ ✓ ✓ ✓

level++

idx

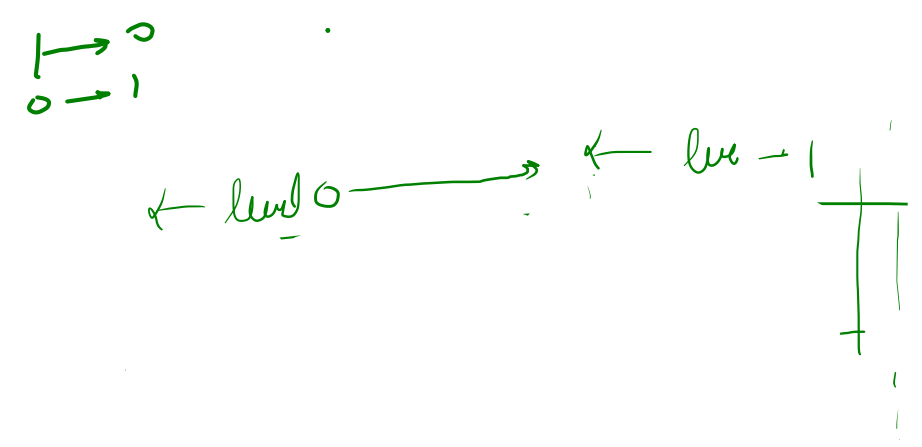
$$i = idx / m$$

$$j = idx \% m$$



$n \times m$

bfs ensures  
min. distance



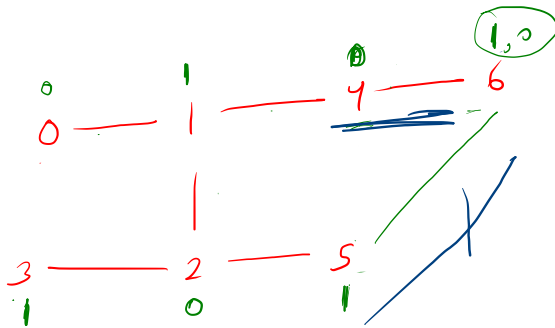
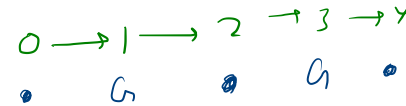
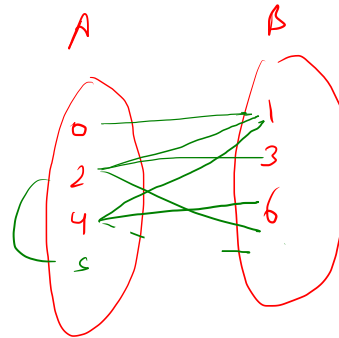
insertion time  
removed time  $\Rightarrow$  costly

● → A  
● → B

Bipartite graph

Graph  $\begin{cases} \text{cycle} \begin{cases} \text{odd length} \rightarrow \text{not bipartite} \\ \text{even length} \rightarrow \text{Bipartite} \end{cases} \\ \text{no cycle} \rightarrow \text{Bipartite} \end{cases}$

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();

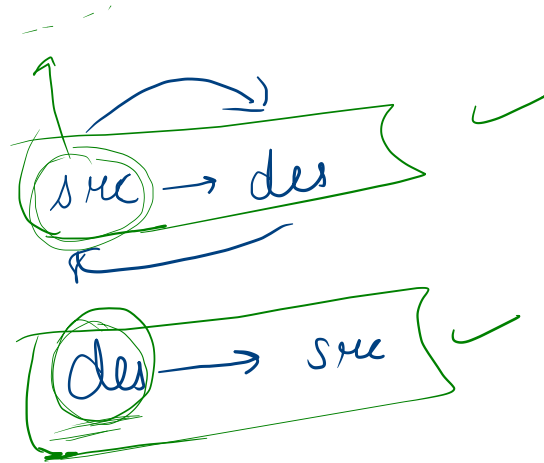
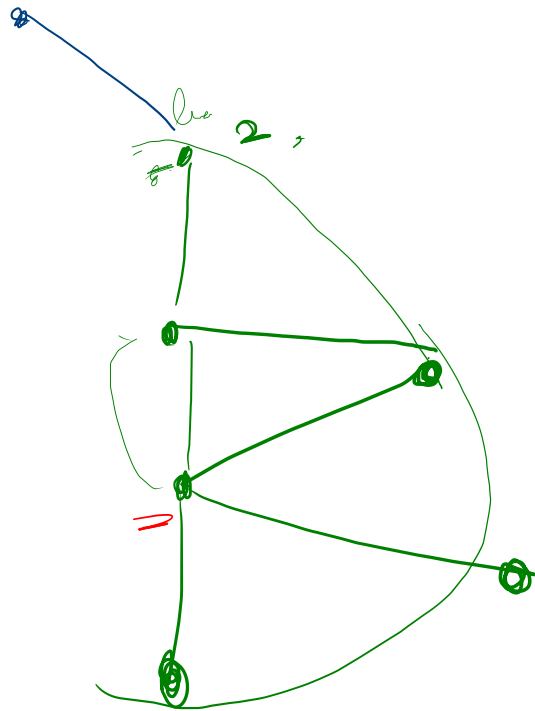
        while(size -- > 0){
            int u = que.removeFirst();

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

    return false;
}
```

src → des

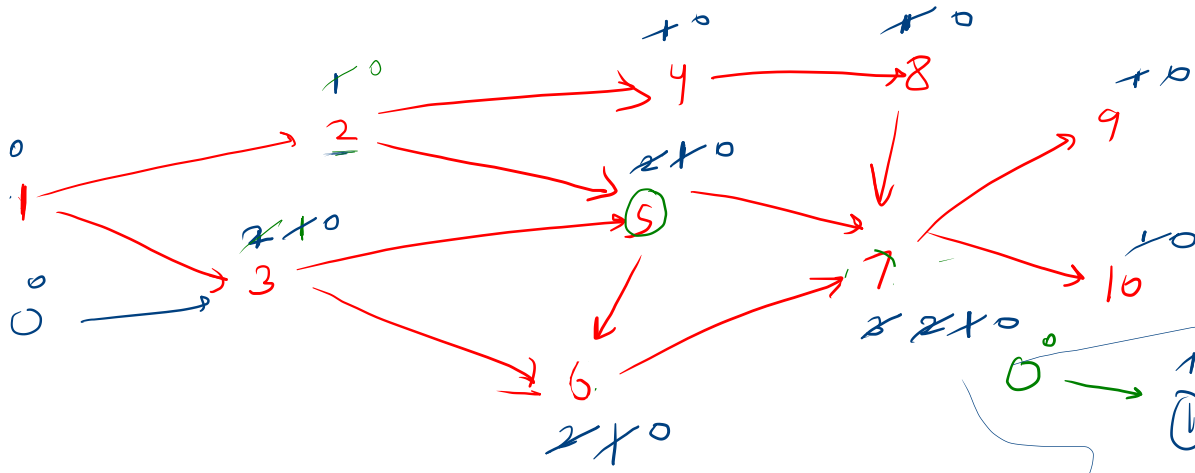
1 → 2





0 → 2

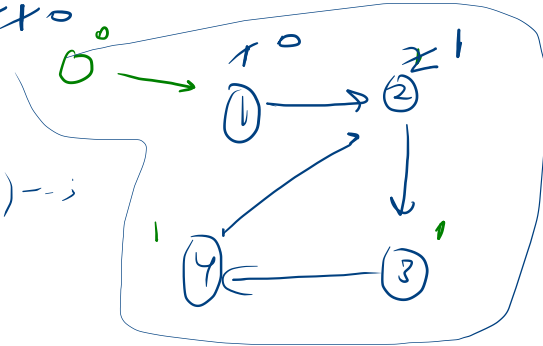
Topo logical sort



number of incoming edges

indegree(0)

indegree(nbrs) ---



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

0, 1