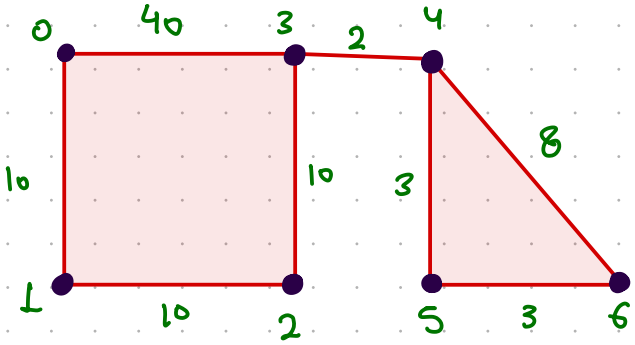
BFS → Shortest path in terms of Edge

0 to 6 → 0 → 3 → 4 → 6 @ 50



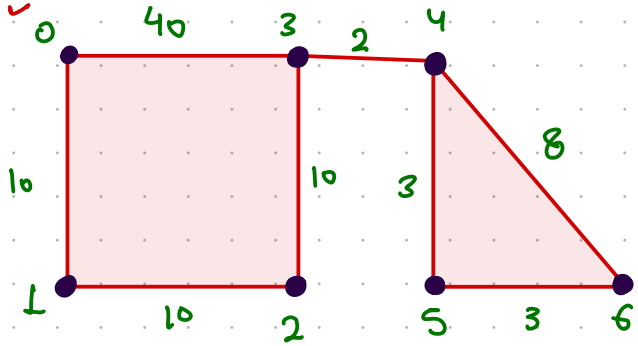
Djikstra's Algo → Shortest path from a source to destination in terms of wt:

0 to 6 → 0 → 1 → 2 → 3 → 4 → 5 → 6 @ 38

Steps of Algo is same as BFS.

D.S used in Djikstra's is Priority Queue (min-priority)

Djikstra's Algo:



src. psf. wsf → priority.

0, "0", 0

1, "01", 10

2, "012", 20

3, "0123", 30

4, "01234", 32

5, "012345", 35

6, "0123456", 38

Continue.

3, "03", 40

Continue.

6, "01236", 40

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

Steps

① get + remove

② mark*

③ work → print

④ add unvisited nbr.

From single source to any destin. we have min path in terms of weight.

0 - 0 @ 0

1 - 01 @ 10

2 - 012 @ 20

3 - 0123 @ 30

4 - 01234 @ 32

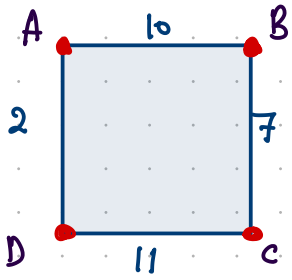
5 - 012345 @ 35

6 - 0123456 @ 38

Min wire Required to connect all PCs [Prims Algorithm]

↳ Minimum Spanning Tree

Tree] → graph → connected [single component]
→ Acyclic [No cyclic in graph]



vertex ≅ server

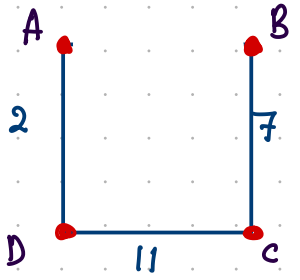
Edge cost = wire length

→ connect all server in min wire.?

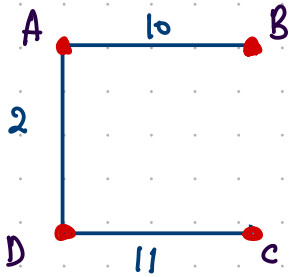
Spanning] → hold all vertex [encompassing to all]

Minimum] → Min. from all possibility

Spanning Tree

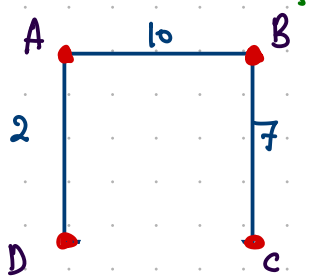


(20)



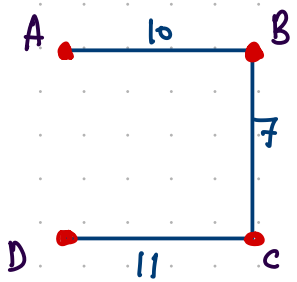
(23)

Min. Spanning Tree

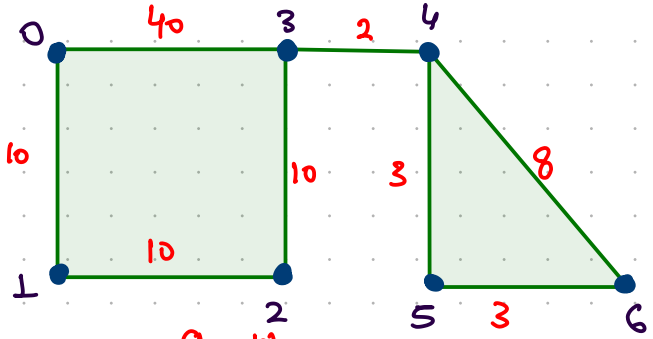


(19)

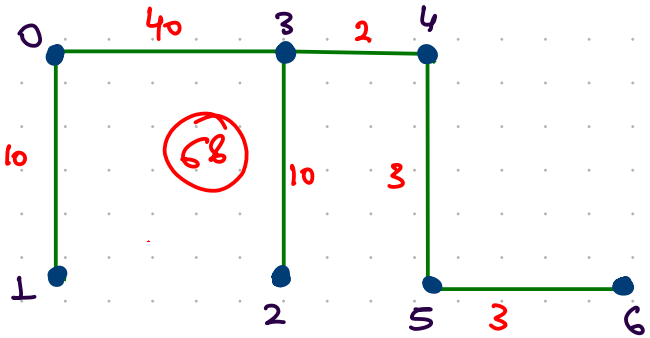
AB - ~~AD~~ - BC



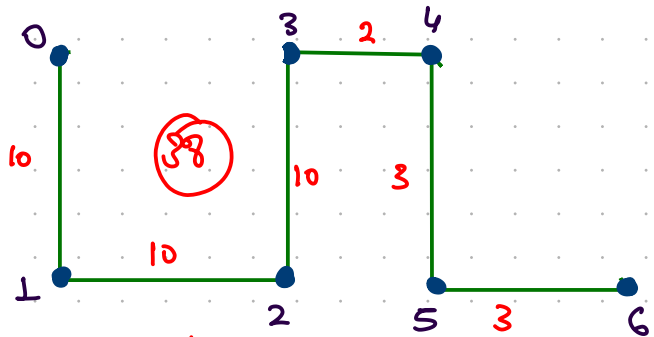
(28)



graph.



Sp. Tree -1



Sp. Tree -2 ...

Prims Algorithm [Minimum Spanning Tree]

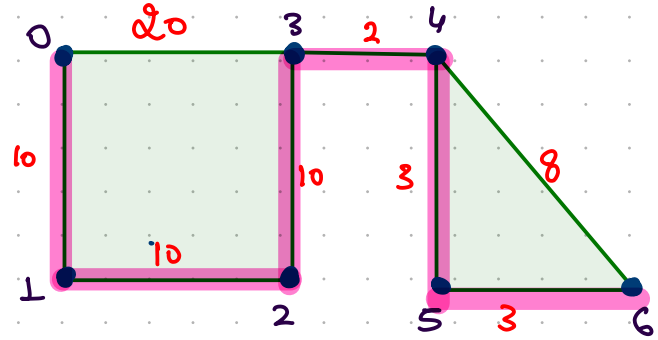
Kruskal's Algorithm
↳ minimum spanning tree
↳ L2 graph.

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

steps D.S → priority Queue

① get + remove
② mark*
③ work → Add Edge Except parent == -1
④ add unvisited nbs.

src, parent, wt



→ 0 to 6 using Dijkstra = 28
wt → 10 + 10 + 10 + 2 + 3 + 3 = 38

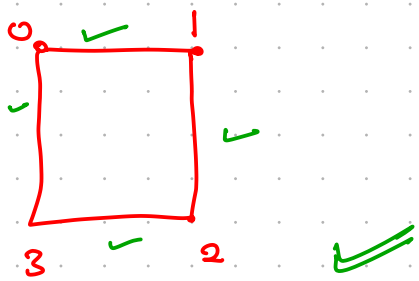
0 to 6 mst = 38 min cost

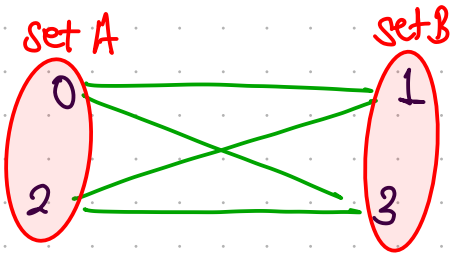
Priority on Edge wt
↳ min priority

prims → mst
Dijkstra → src to dst
Shortest path

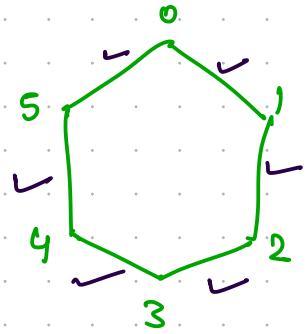
Is graph a bipartite :

If we can divide graph in two set. and there is no edge within the set then graph is bipartite graph.

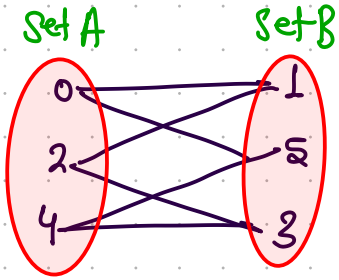




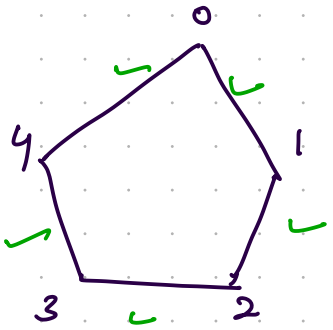
→ Bipartite graph.

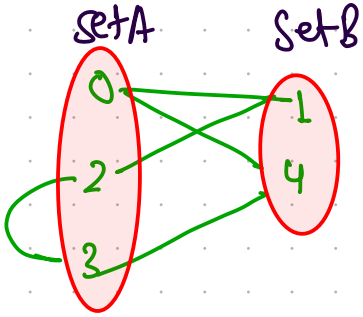


Bipartite graph.

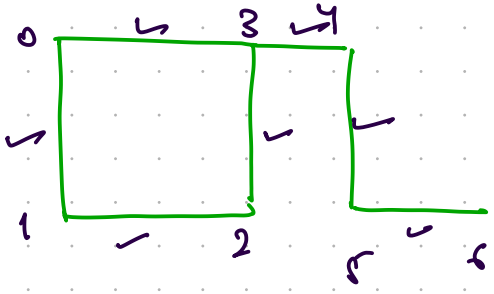


within set no edge

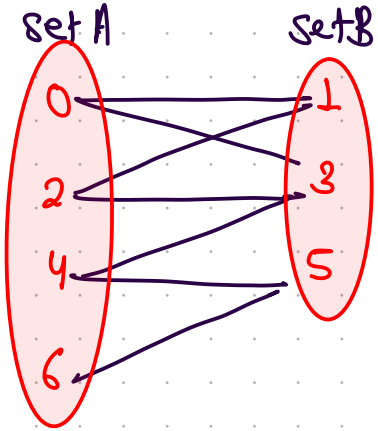




there is an edge within set
So this not a bipartite graph.



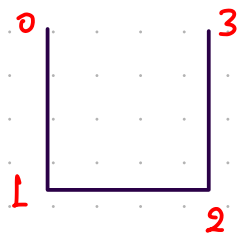
Bipartite graph.

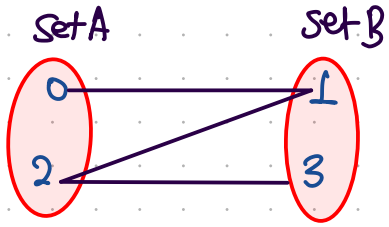


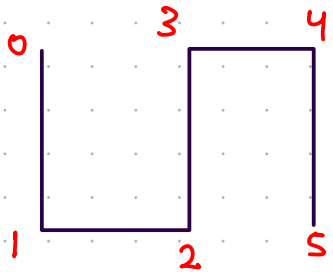
within set Edge → If graph is cyclic

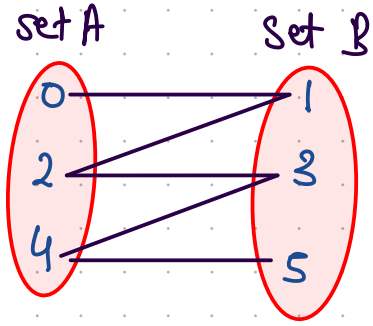
If graph is acyclic] → separation of vtx is always poss.

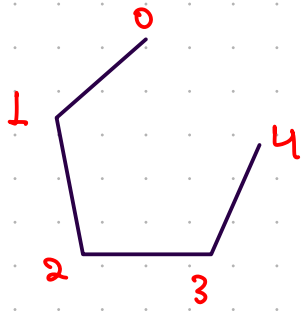
If graph is acyclic then graph is bipartite graph.

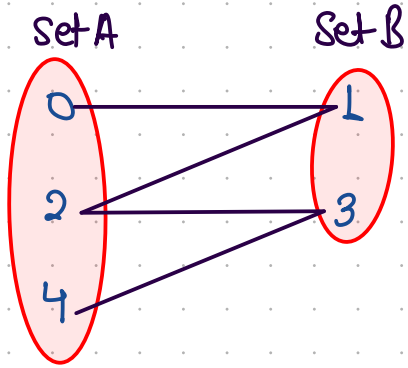


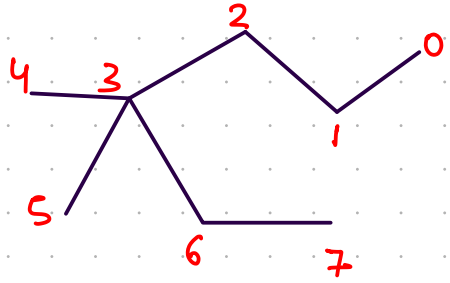


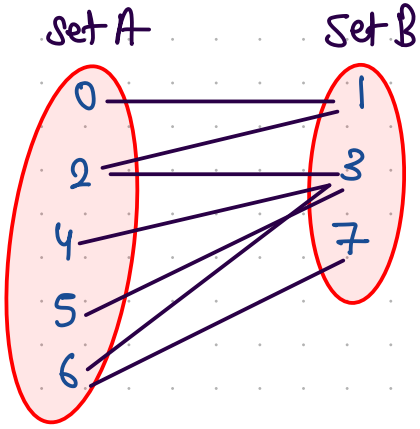






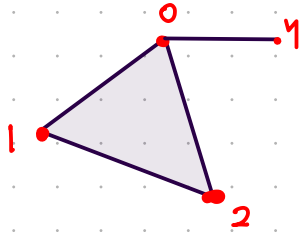


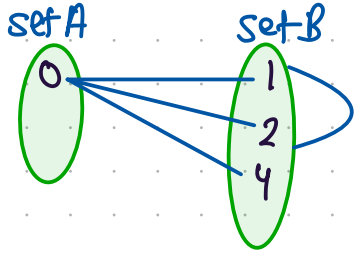




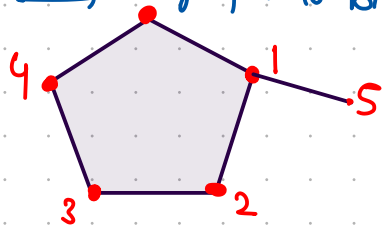
If graph is cyclic:

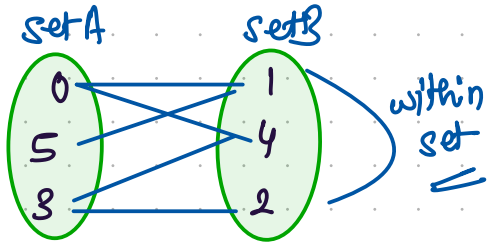
cycle length → odd → graph not bipartite
→ Even → graph is bipartite.



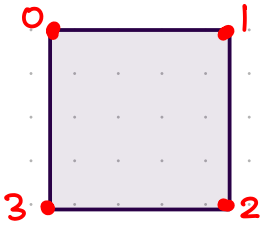


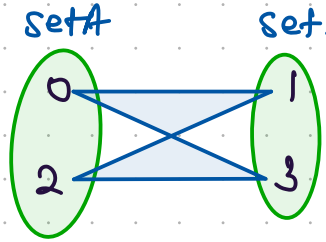
within set

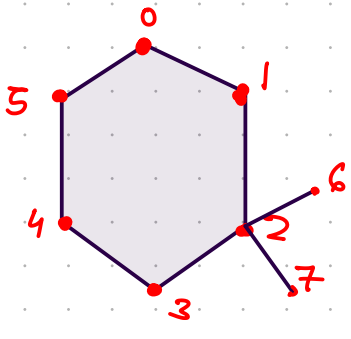


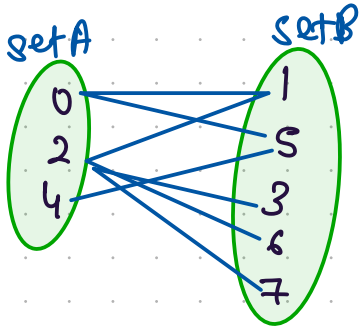


within set



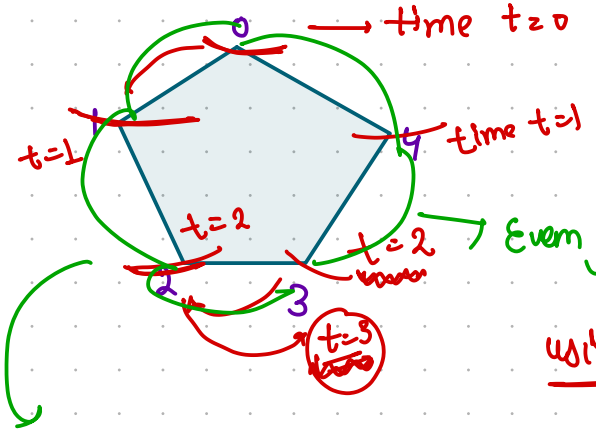




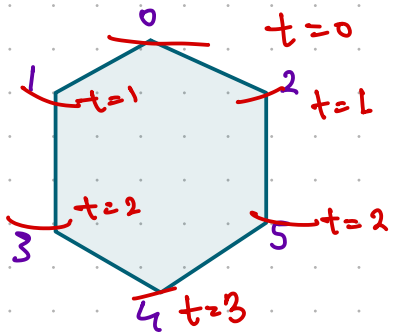


How to know that graph have odd cycle or even cycle! →

discovery time.



order of time] ×



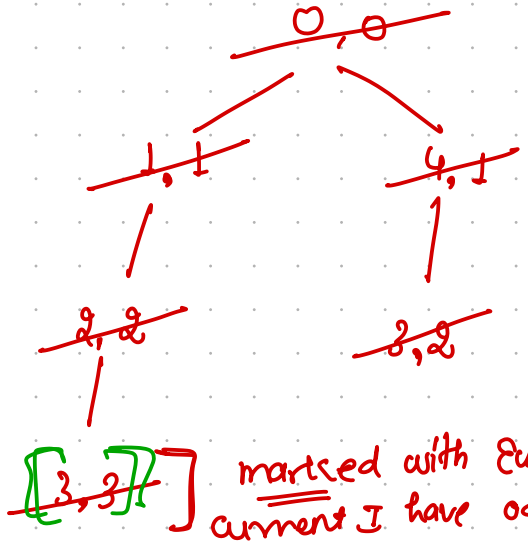
order of time] ✓

using BFS

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

3 is marked with Even.

src. discovery time] level

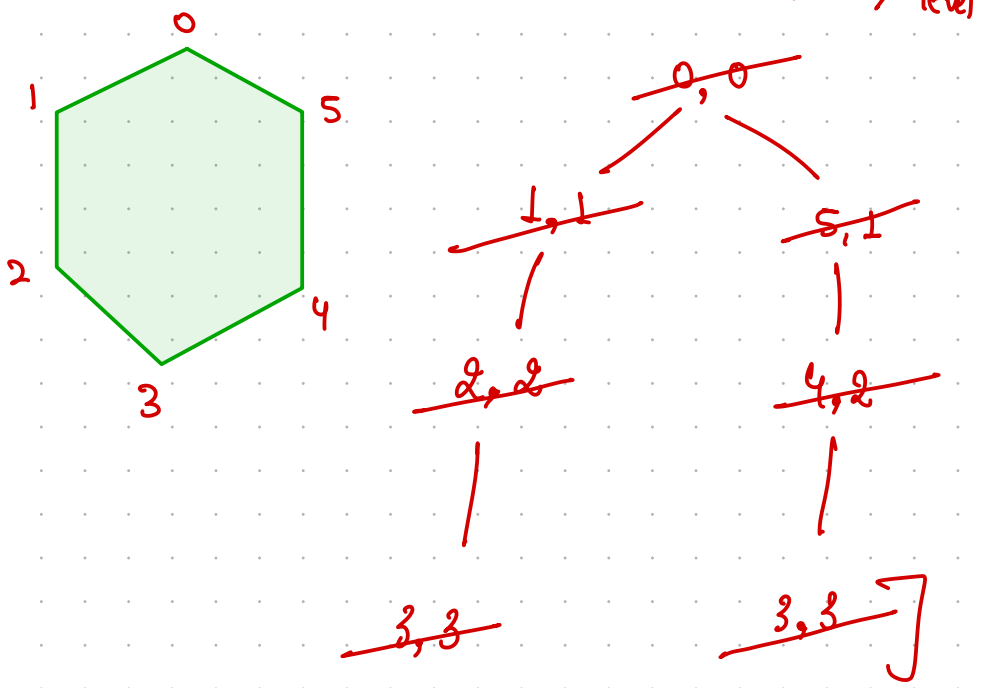


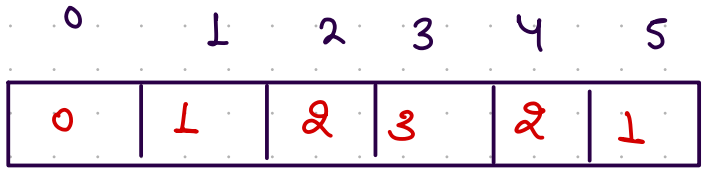
marked with Even & current I have odd time.

Even jump count + odd jump count

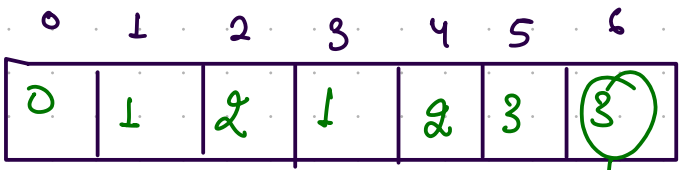
= cycle is odd length.

= graph is not a bipartite graph.



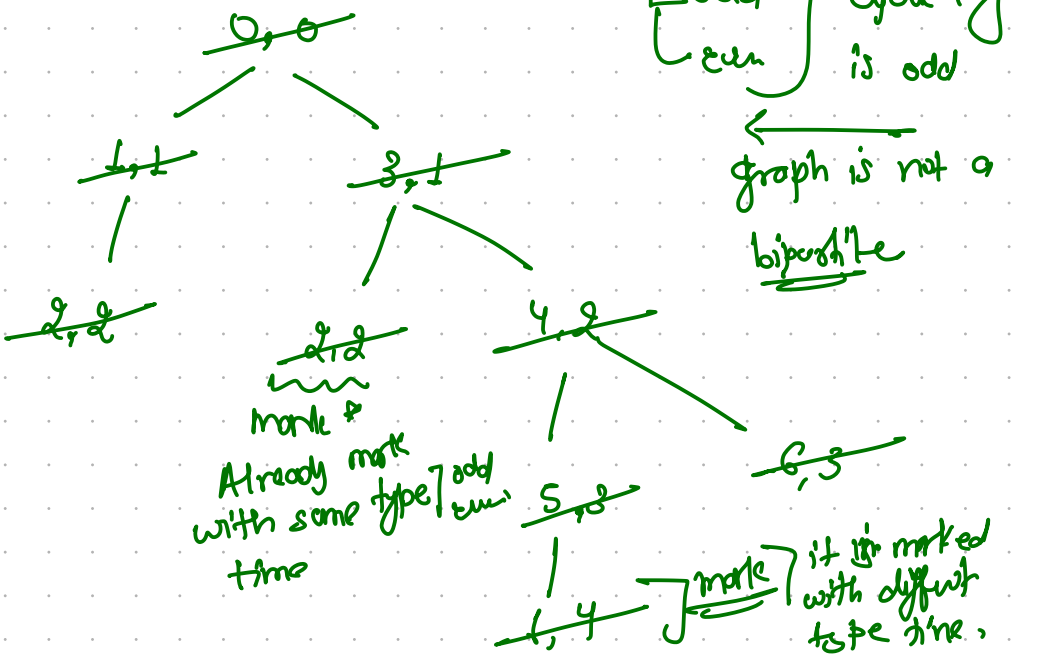


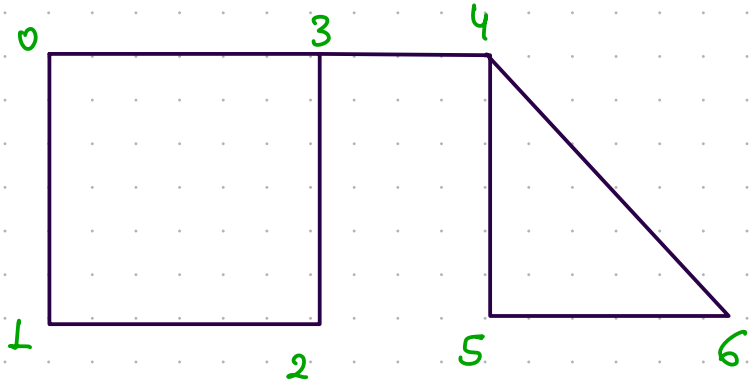
continue. but have same time. so can't stop here, because it may be bipartite.



odd / even → cycle length is odd

graph is not a bipartite



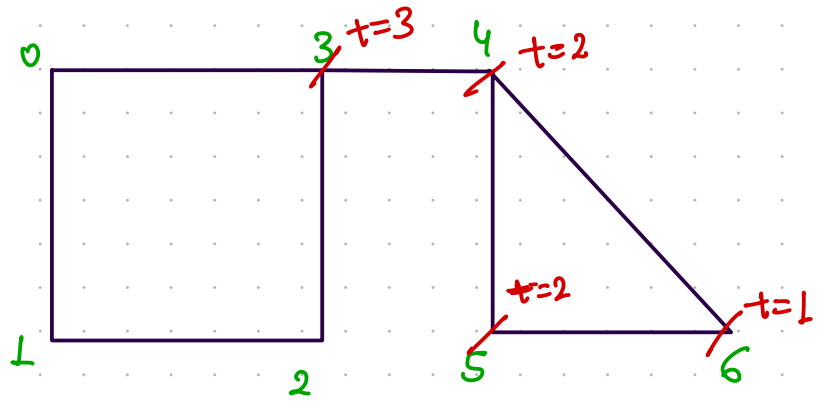


NOTE: It is not necessary that graph should be connect it may disconnected & bipartite as well.

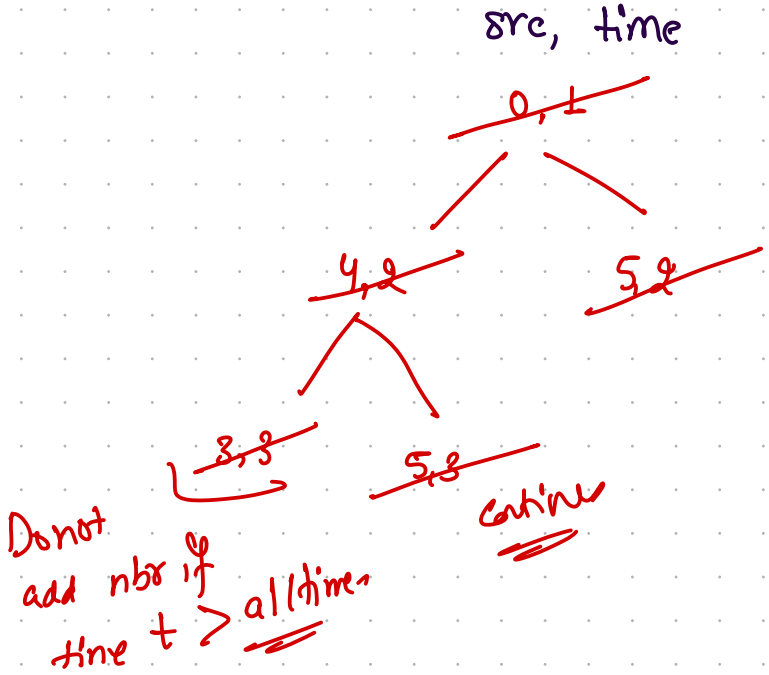
Spread of Infection:

At time, t=e . 6th person got infected at time t=3, How many people are infected.

Rate of spreading → 1 unit time.



total person: 4

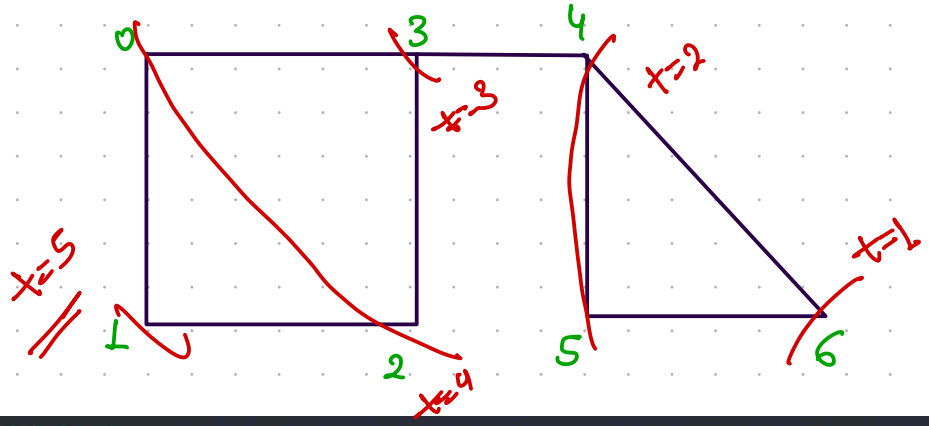


alltime : 3

count = 0 1 2 3 4 (4)

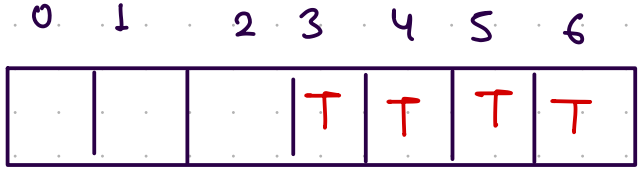
Vis	0	1	2	3	4	5	6
	T			T	T	T	

Answer : 4

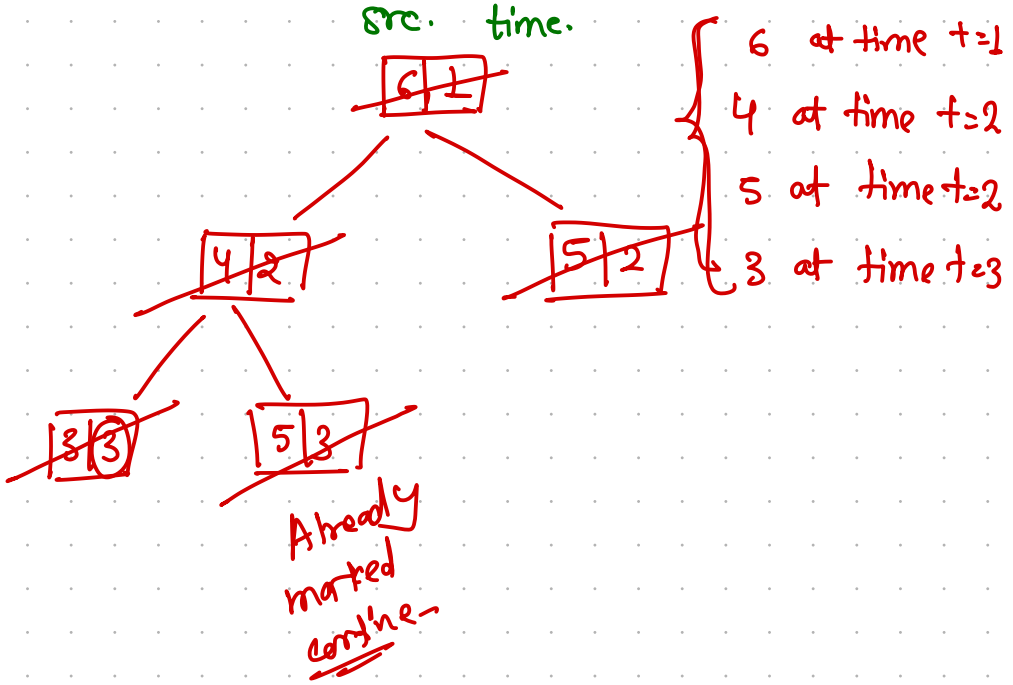


count = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

All time = 3




```java
// spread of infection
public static int spreadOfInfection(ArrayList<Edge>[] graph, int src, int time) {
    int count = 0;
    Queue<int[]> qu = new LinkedList<>();
    // int[] first element is src, and second is time
    qu.add(new int[]{src, 1});
    boolean[] vis = new boolean[graph.length];
    while(qu.size() > 0) {
        int[] rem = qu.remove();
        if(vis[rem[0]] == true) {
            continue;
        } else {
            vis[rem[0]] = true;
        }
        System.out.println(rem[0] + " at time t = " + rem[1]);
        count++;
        for(Edge e : graph[rem[0]]) {
            if(vis[e.nbr] == false && rem[1] < time) {
                qu.add(new int[]{e.nbr, rem[1] + 1});
            }
        }
    }
    return count;
}
```



Graph Home work

① order of compilation } topological sort

Auto ⟷ ② Iterative DFS

Recursive

Heap and Hash map,