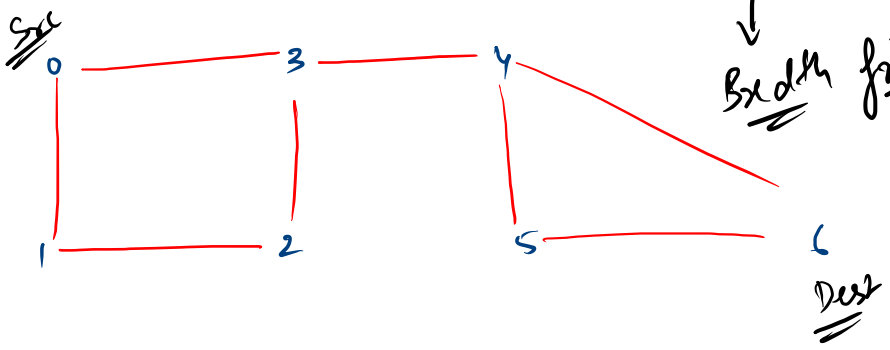
Graph -> {V, E}

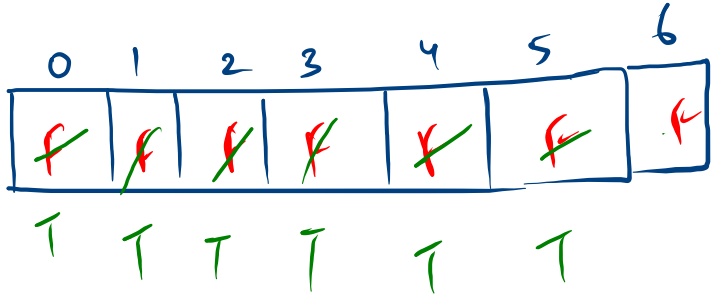
Depth first traversal

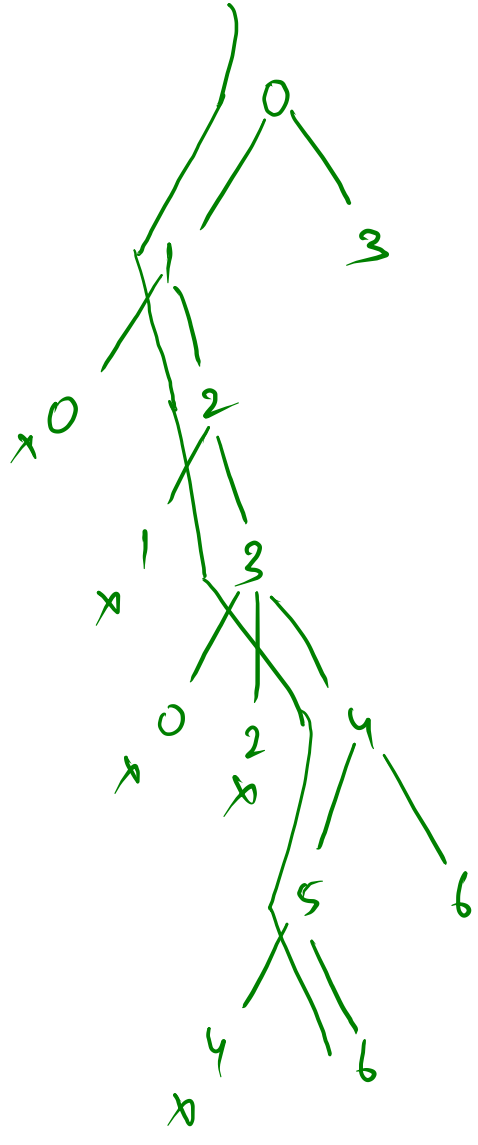
DFS → pre/post/in

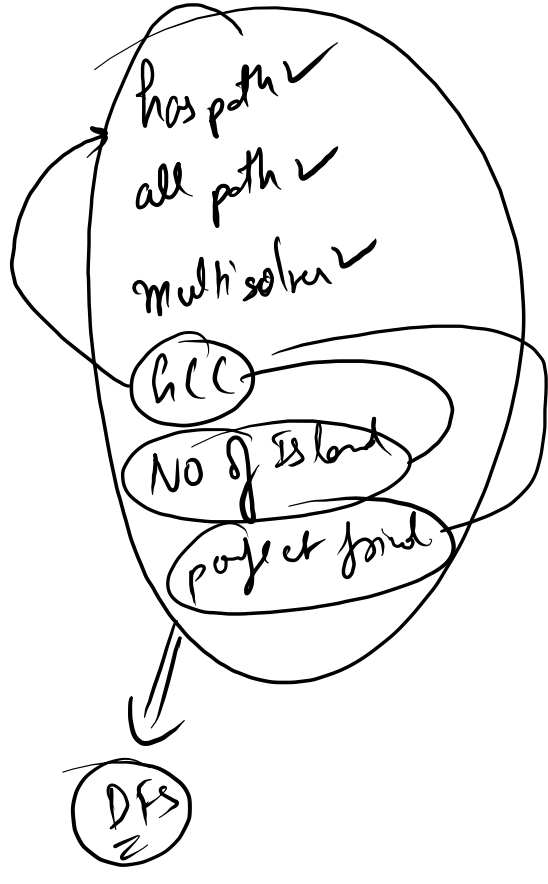
BFS → Level order

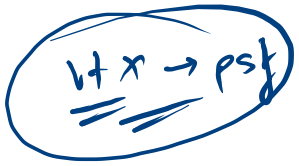
Breadth first traversal

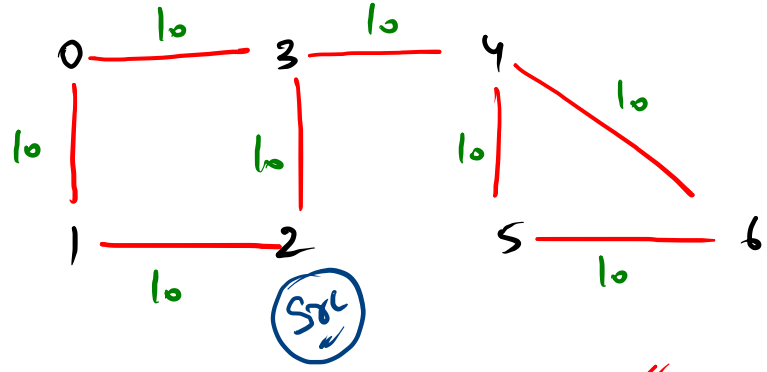


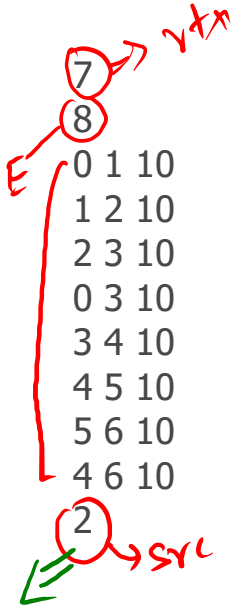


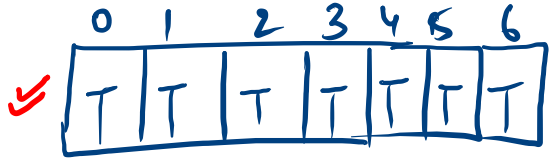


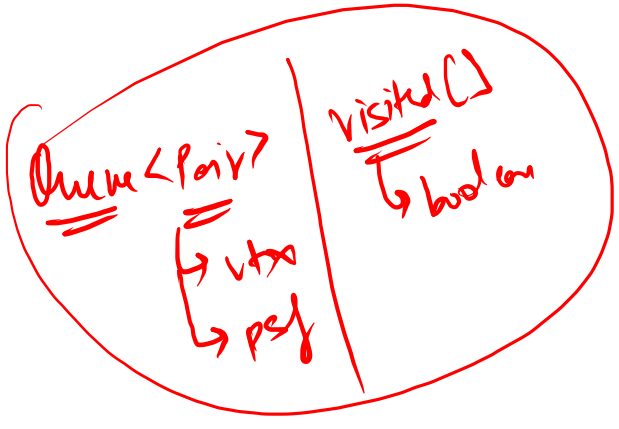


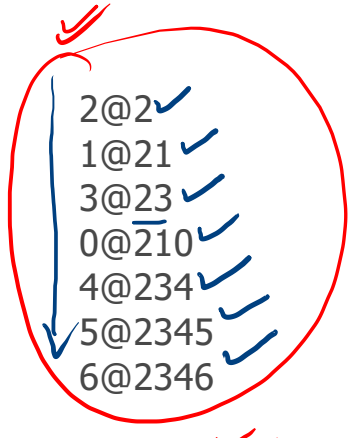

vtx → psf

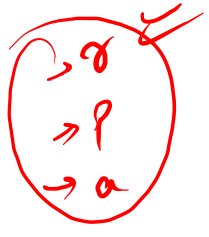

0
3
4
1
2
5
6
10
src

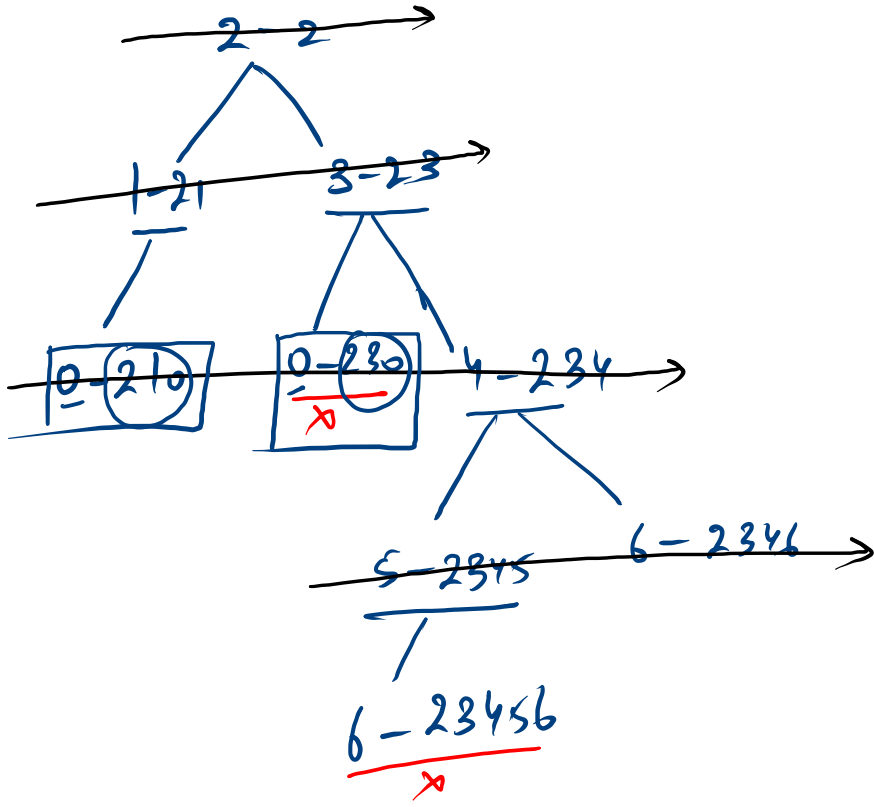

vtx
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
2
src

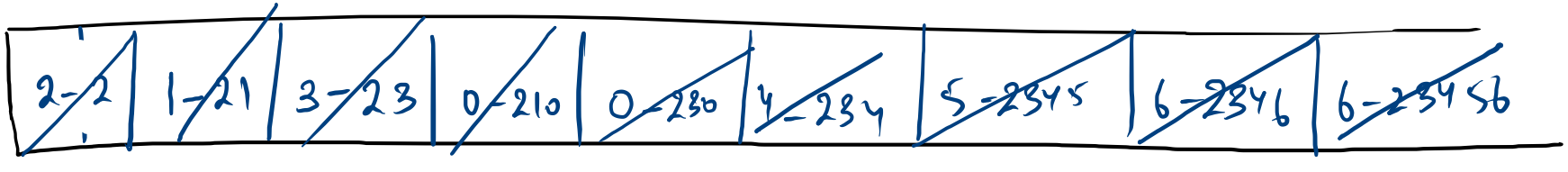

0 1 2 3 4 5 6
T T T T T T T

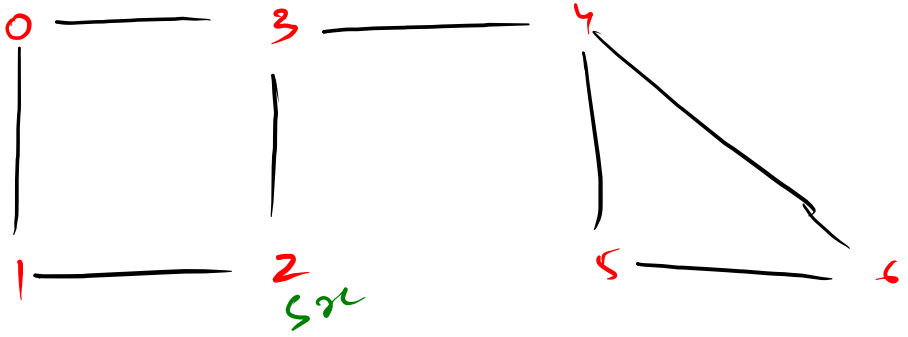

Queue<Pair>
vtx
psf
visited[]
boolean

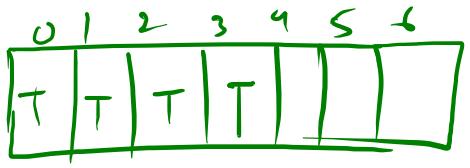

2@2
1@21
3@23
0@210
4@234
5@2345
6@2346


g
p
a


2-2
1-21
3-23
0-210
0-230
4-234
5-2345
6-2346
6-23456

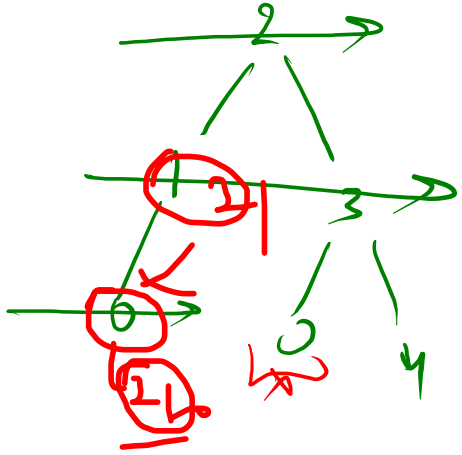

2-2
1-21
3-23
0-210
0-230
4-234
5-2345
6-2346
6-23456

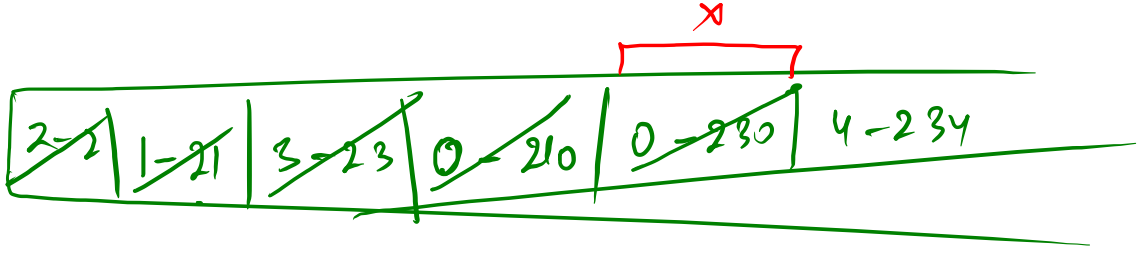




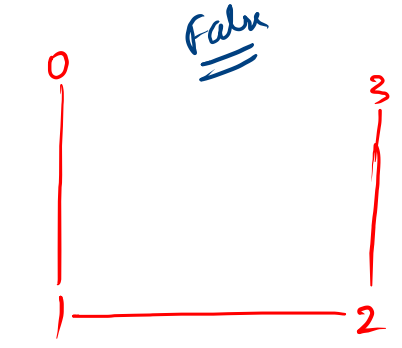

```
public static class Pair{
    int vtx;
    String psf;

    Pair(int vtx , String psf){
        this.vtx = vtx;
        this.psf = psf;
    }
}
public static void bfs(ArrayList<Edge>[] graph,int src){
    Queue<Pair> queue = new ArrayDeque<>();

    queue.add(new Pair(src,""+src));
    boolean visited[] = new boolean[graph.length];
    while(queue.size() > 0){
      Pair fpair = queue.remove();

      if(visited[fpair.vtx] == false){
          visited[fpair.vtx] = true;
          System.out.println(fpair.vtx+"@"+fpair.psf);

          for(Edge e : graph[fpair.vtx]){
              if(visited[e.nbr] == false){
                  queue.add(new Pair(e.nbr,fpair.psf+e.nbr));
              }
          }
      }
    }
}
```

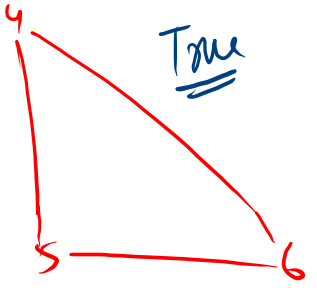
2@2
1@21
3@23
0@210

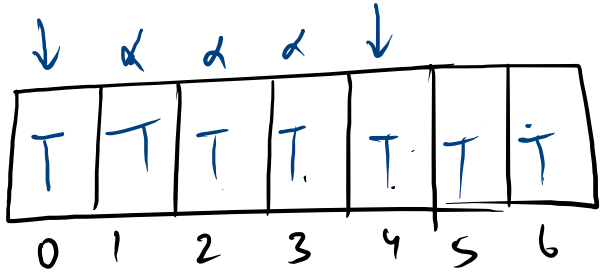


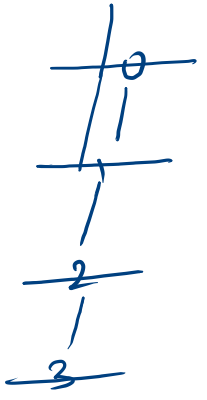


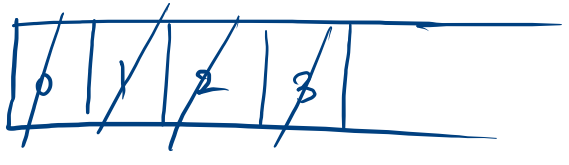
Cycle

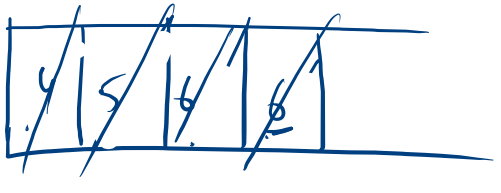






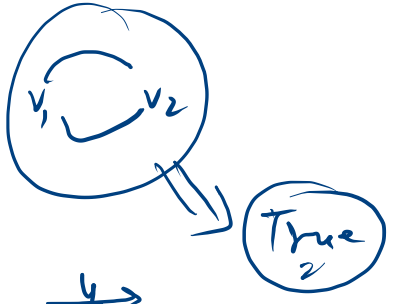


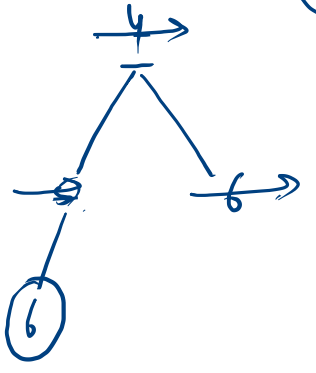




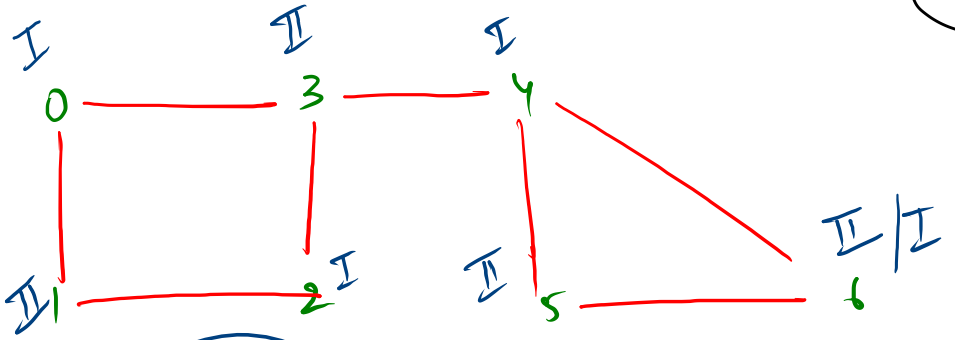
Graph

True





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



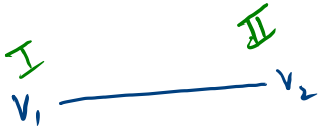
Non-bipartite

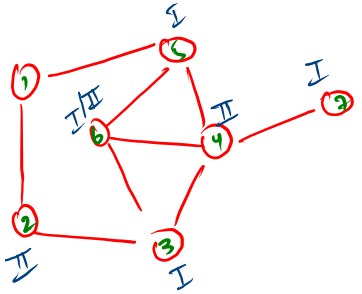
I
0, 2, 4, 6

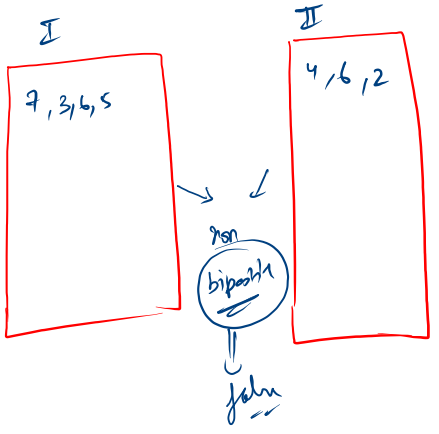
II
1, 3
5, 6

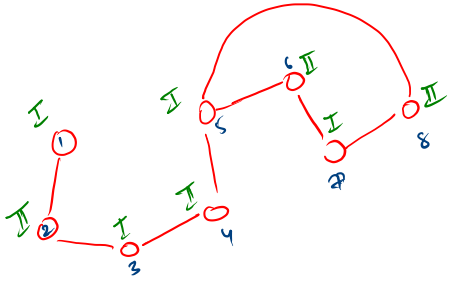
Bi-partite

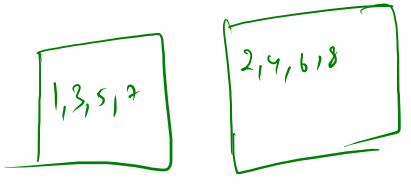
graph
2 parts
Intersection == zero









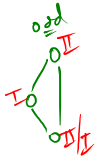


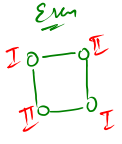
observation

graph → acyclic → (Bipartite → true)

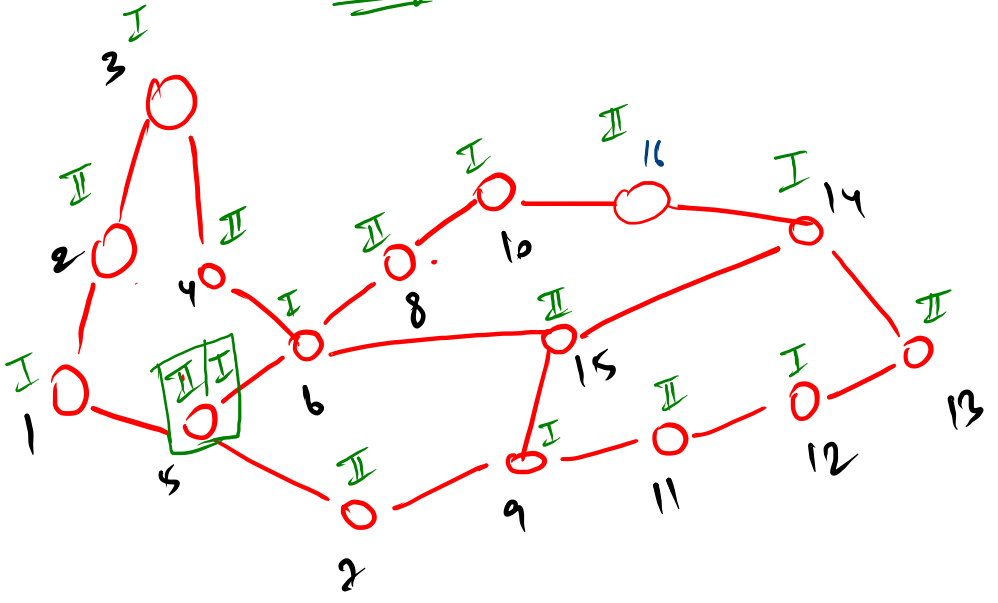
cyclic → odd → (Bipartite → false)

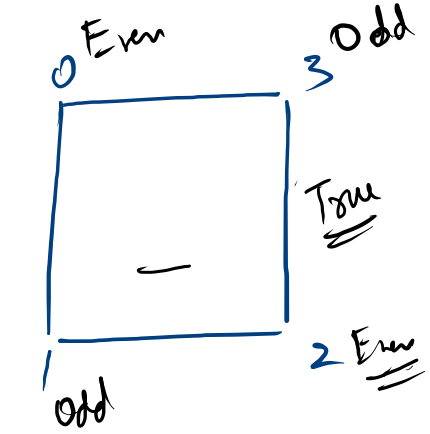
Even → (Bipartite → true)

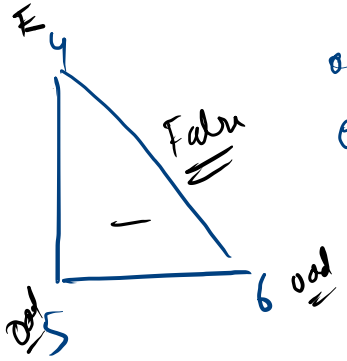




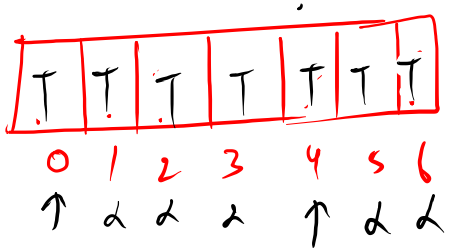
True / false ?





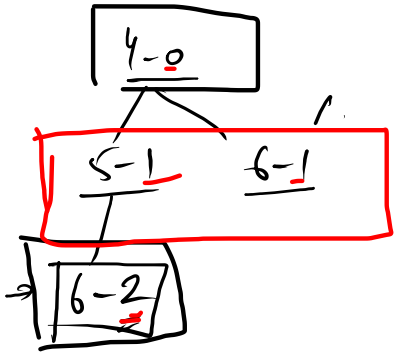


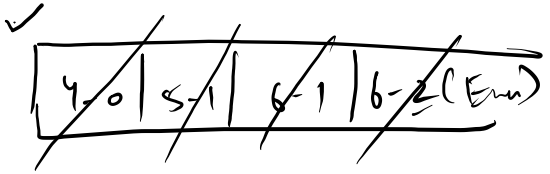
odd level
even level

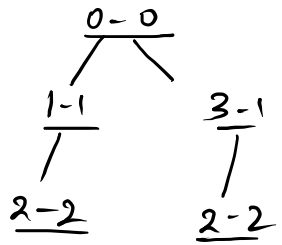


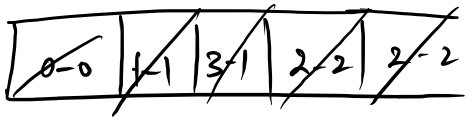
Integer Integer

Vtx	level
4	0
5	1
6	1 (odd)



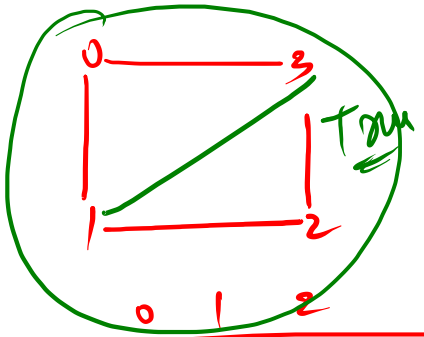


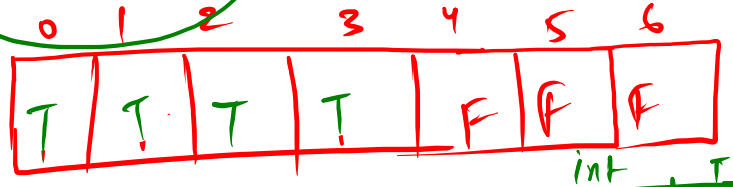


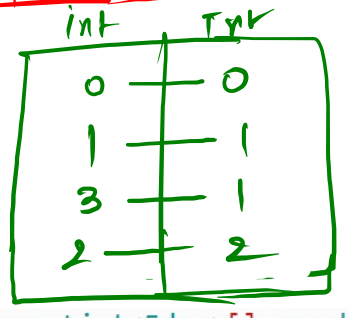


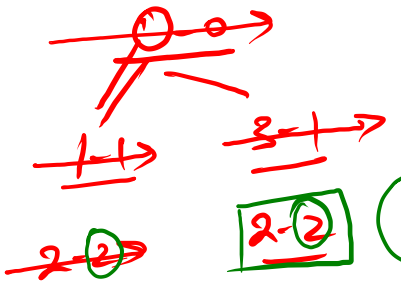
Integer Integer

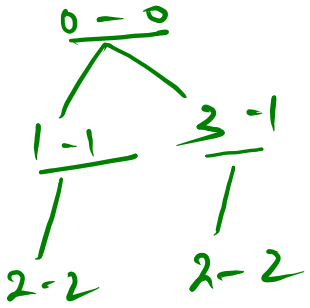
Vtx	level
0	0
1	1
3	1
2	2

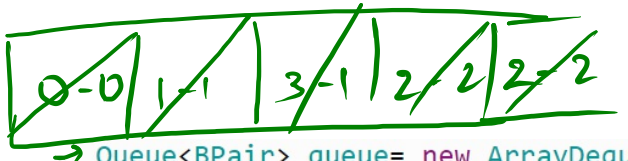










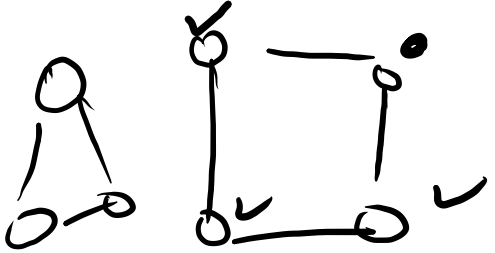



```
public static boolean isBipartite(ArrayList<Edge>[] graph){
    boolean visited[]=  new boolean[graph.length];
    for(int v = 0 ; v < graph.length ; v++){
        if(visited[v] == false){
            boolean res = isCompBipartite(graph,v,visited);
            if(res == false){
                return false;
            }
        }
    }

    return true;
}
```

```
Queue<BPair> queue= new ArrayDeque<>();
queue.add(new BPair(src,0));
HashMap<Integer,Integer> hm = new HashMap<>();
while(queue.size() > 0){
    BPair pair = queue.remove();

    if(visited[pair.vtx] == true){
        if(pair.level%2 != hm.get(pair.vtx)%2){
            return false;
        }
    }else{
        hm.put(pair.vtx,pair.level);
        visited[pair.vtx] = true;

        for(Edge e : graph[pair.vtx]){
            if(visited[e.nbr] == false){
                queue.add(new BPair(e.nbr,pair.level+1));
            }
        }
    }
}
return true;
```

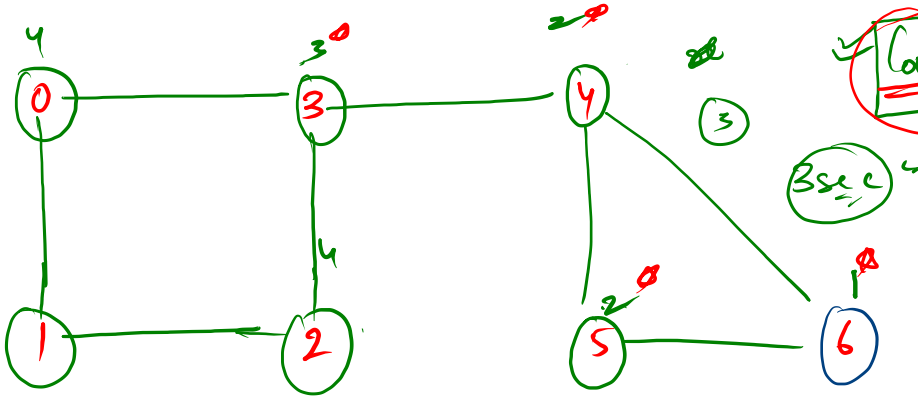
src = 0

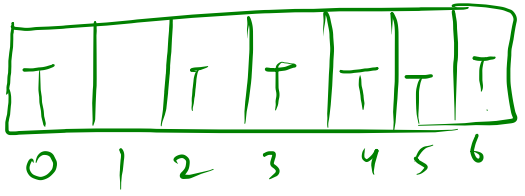
2%2 == 2%2

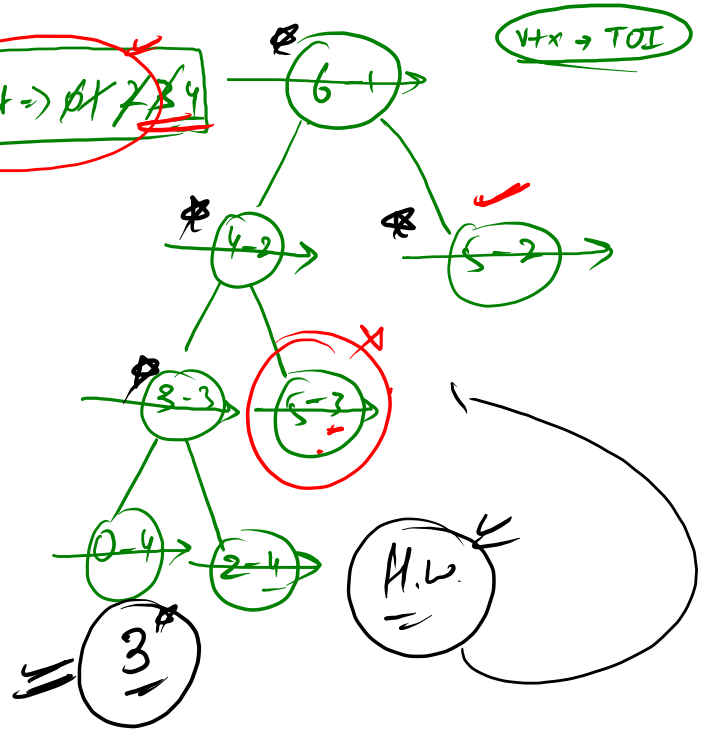


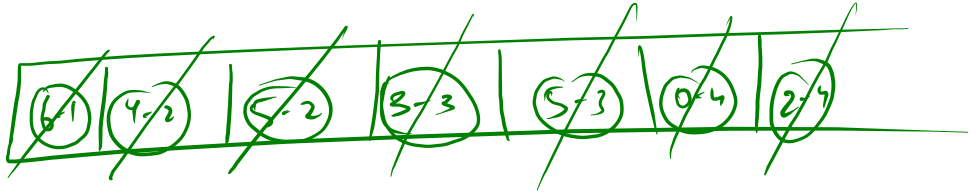
BFS

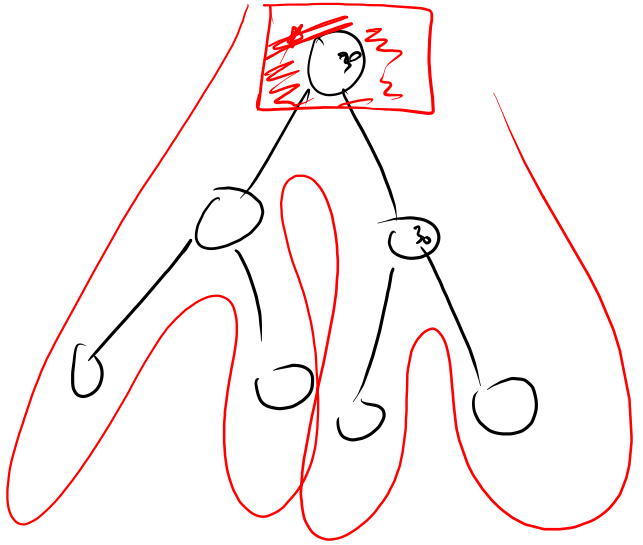
```
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10
6
3
```



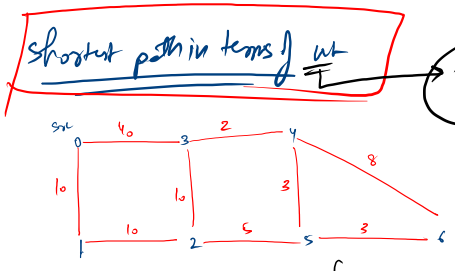


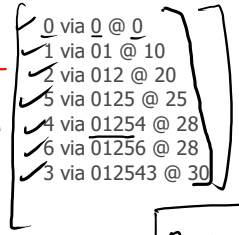


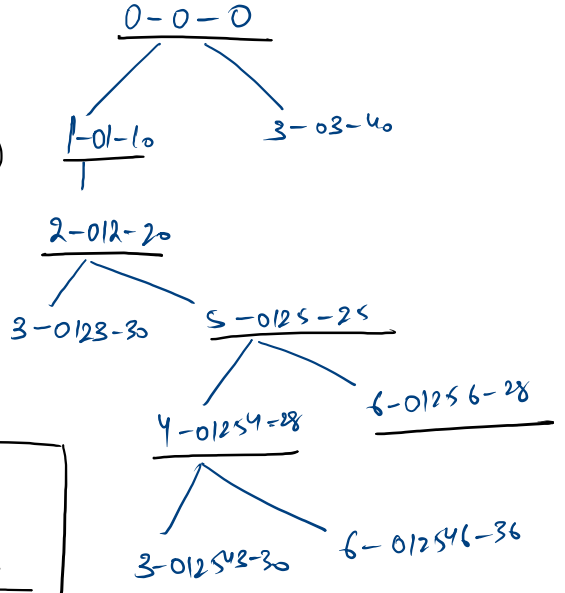




```
7
9
0 1 10
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
2 5 5
0
```







- Introduction To Graphs And Its Representation
- Has Path?
- Print All Paths
- Multisolver - Smallest, Longest, Ceil, Floor, Kthlar...
- Get Connected Components Of A Graph
- Is Graph Connected → H.W. / hCC → application → 1 Line code
- Number Of Islands
- Perfect Friends
- Hamiltonian Path And Cycle
- Knights Tour → H.W. / same as recursion
- Breadth First Traversal
- Is Graph Cyclic
- Is Graph Bipartite
- Spread Of Infection → H.W. / BFS application / Discussed
- Shortest Path In Weights
- Minimum Wire Required To Connect All Pcs
- Order Of Compilation
- Iterative Depth First Traversal

→ Double → for tomorrow