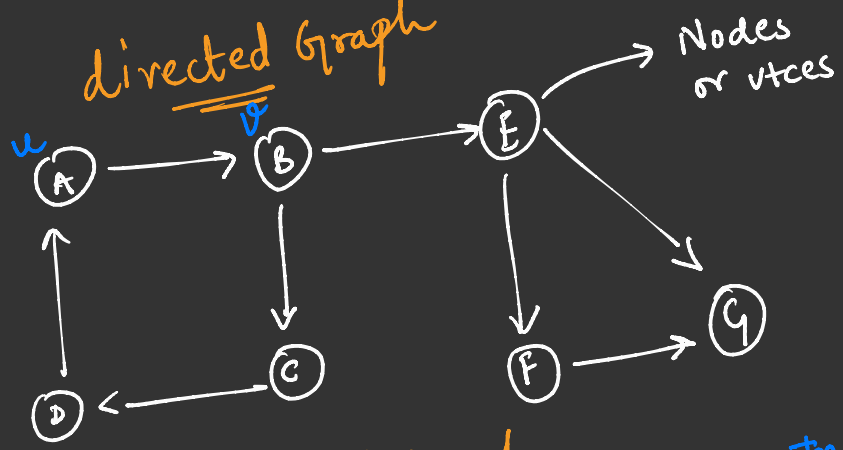
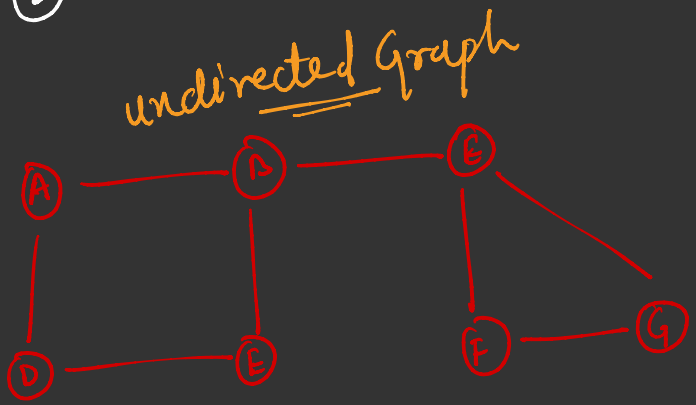
Graph

Graph.



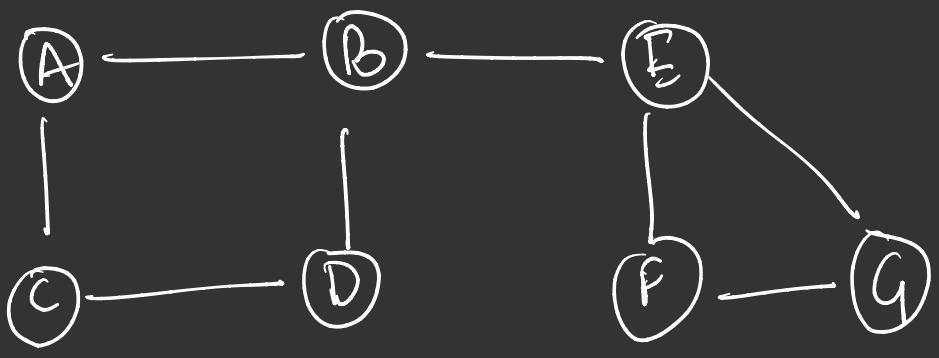


Input

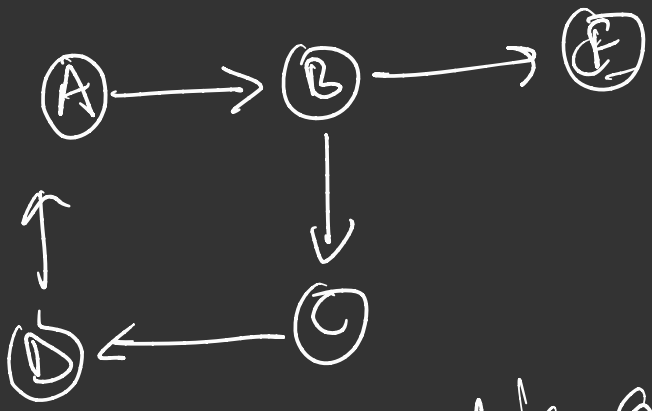
[u v w]

↓ ↓ ↓

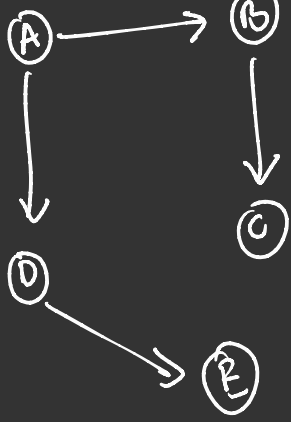
initial final weight



undirected cyclic graph



directed cyclic graph



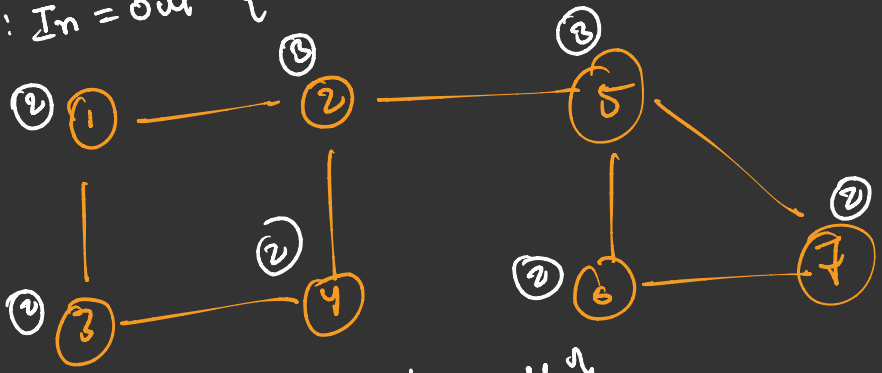
Directed Acyclic Graph

(DAG)

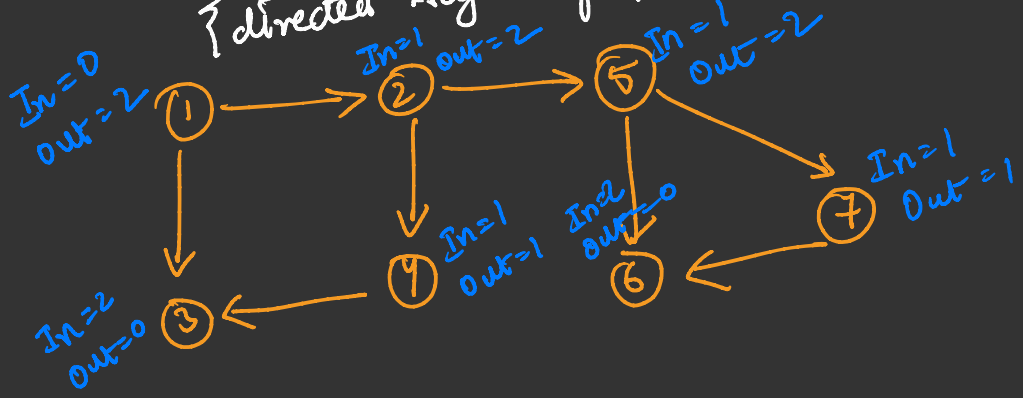
{ generic tree }

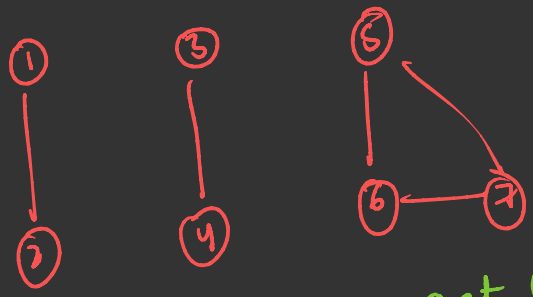
Indegree | Out degree

NOTE : In = out { in undirected graphs }



{ directed Acyclic graph }





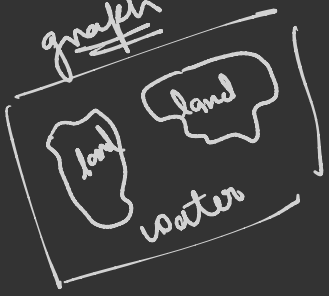
Undirected Graph

7 vtces

get Connected Components

3

u	v	w
1	2	10
3	4	100
5	6	20
6	7	30
7	5	40



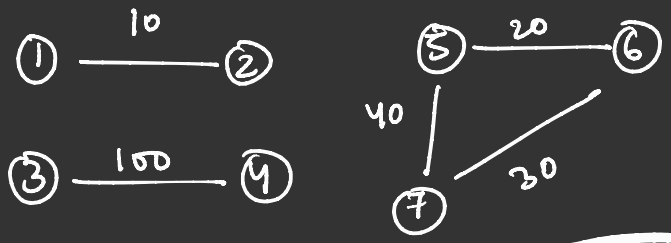
lets build [undirected graph]

arr[u][v] = w
arr[v][u] = w

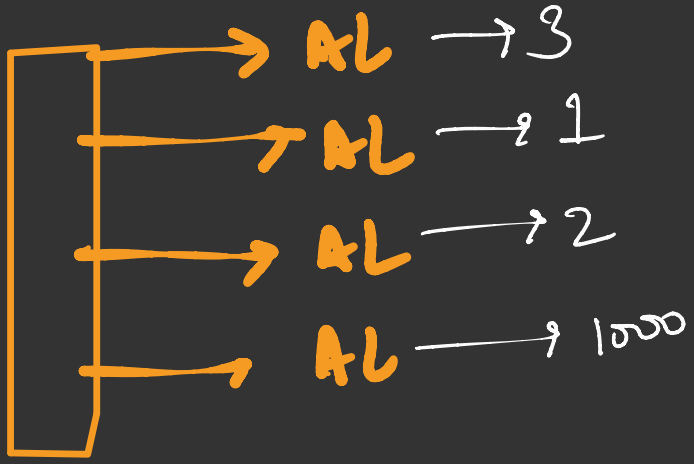
	1	2	3	4	5	6	7
1	0	10	0	0	0	0	0
2	10	0	0	0	0	0	0
3	0	0	0	100	0	0	0
4	0	0	100	0	0	0	0
5	0	0	0	0	0	20	40
6	0	0	0	0	20	0	30
7	0	0	0	0	40	30	0

Adj Matrix

N x N [vtc x vtc]



Arraylist<Integer>[] graph

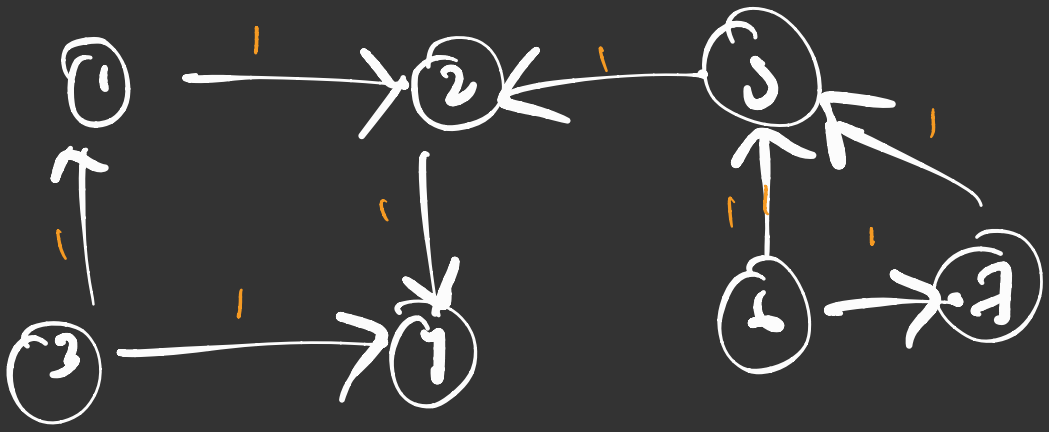


1 → {2}
2 → {1}
3 → {4}
4 → {3}
5 → {6,7}

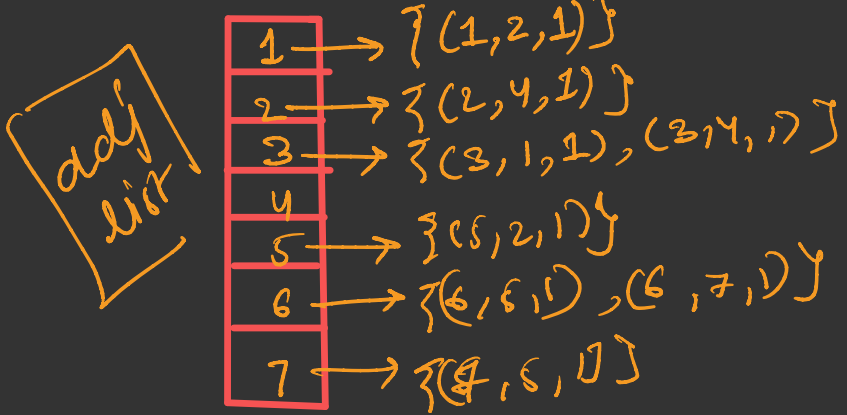
{ Adj List }

6 → {5, 7}

7 → {5, 6}



pair { src dest wt }



HAS PATH

```
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
0
6
```

7 → N

8 → No. of query of (u, v, w)

0 → {(0,1,10), (0,3,10)}

1 → {(1,0,10), (1,2,10)}

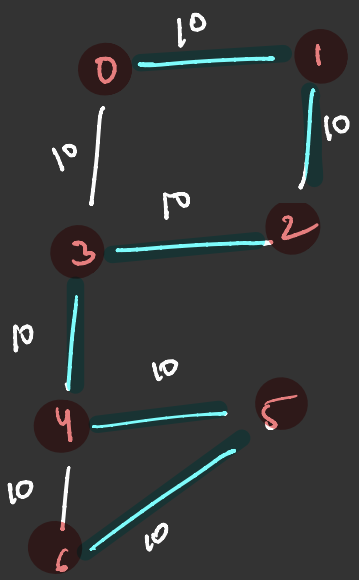
2 → {(2,1,10), (2,3,10)}

3 → {(3,0,10), (3,2,10), (3,4,10)}

4 → {(4,3,10), (4,5,10), (4,6,10)}

5 → {(5,4,10), (5,6,10)}

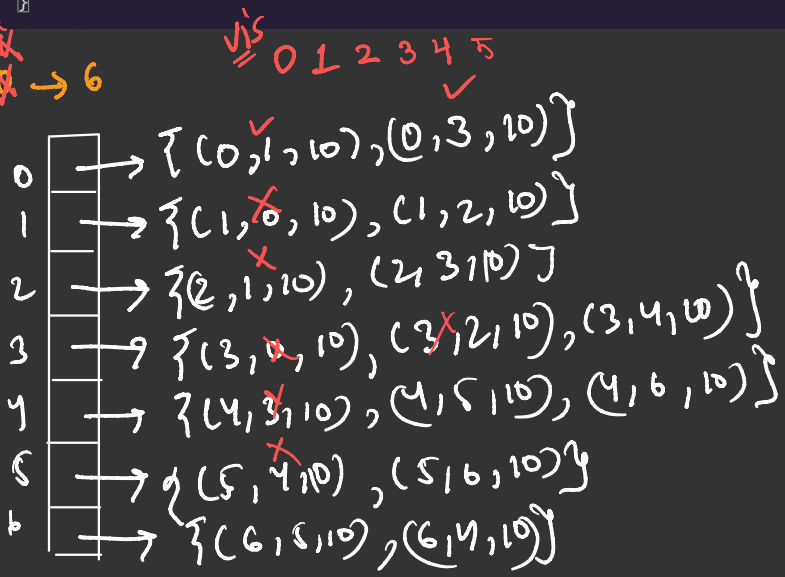
6 → {(6,5,10), (6,4,10)}



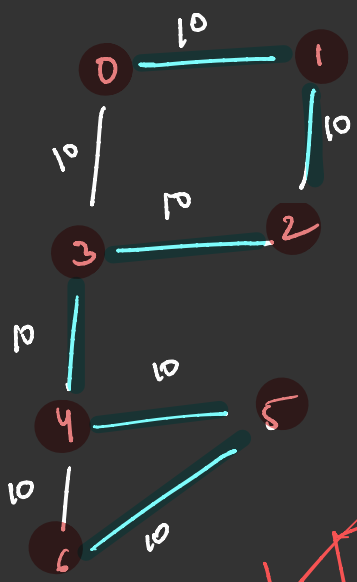
path traversed

0 → 1 → 2 → 3 → 4 → 5 → 6

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

    vis[src] = true;
    for(Edge edge : graph[src]) {
        if(vis[edge.nbr] != true)
            return hasPath(graph, edge.nbr, dest, vis);
    }

    return false;
}
```



Print All Paths



paths 0–6

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

DFS

```
public static void printAllPaths(ArrayList<Edge>[] graph, int src, int dest, boolean[] vis, String psf) {
    if(src == dest) {
        System.out.println(psf + dest);
        return;
    }

    vis[src] = true;
    for(Edge edge : graph[src]) {
        if(vis[edge.nbr] != true)
            printAllPaths(graph, edge.nbr, dest, vis, psf + src);
    }
    vis[src] = false;
}
```

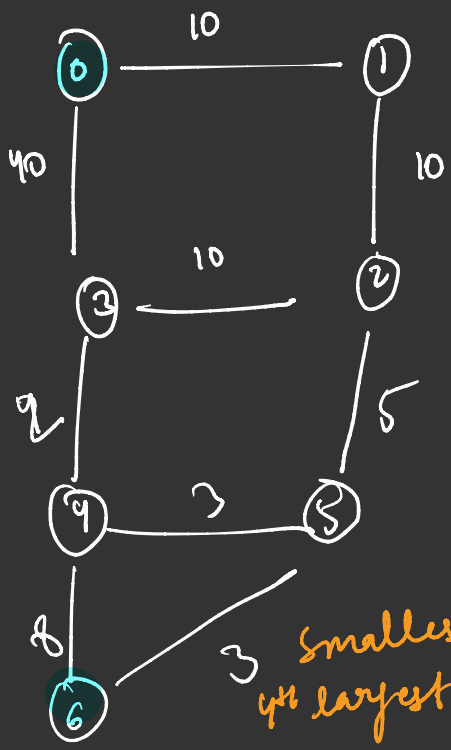
Multisolver !

- → Smallest
- → Longest
- → Ceil
- → Floor
- → K^{th} largest path

```
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
6
30
4
```

Sample Output

```
Smallest Path = 01256@28
Largest Path = 032546@66
Just Larger Path than 30 = 012546@36
Just Smaller Path than 30 = 01256@28
4th largest path = 03456@48
```



src = 0

dest = 6

Criteria = 30

K = 4

0 1 2 3 4 5 6 @ 38 (3)

0 1 2 3 4 6 @ 40 (4)

0 1 2 5 4 6 @ 36 (2)

0 1 2 5 6 @ 28 ✓ (1) ← Smallest

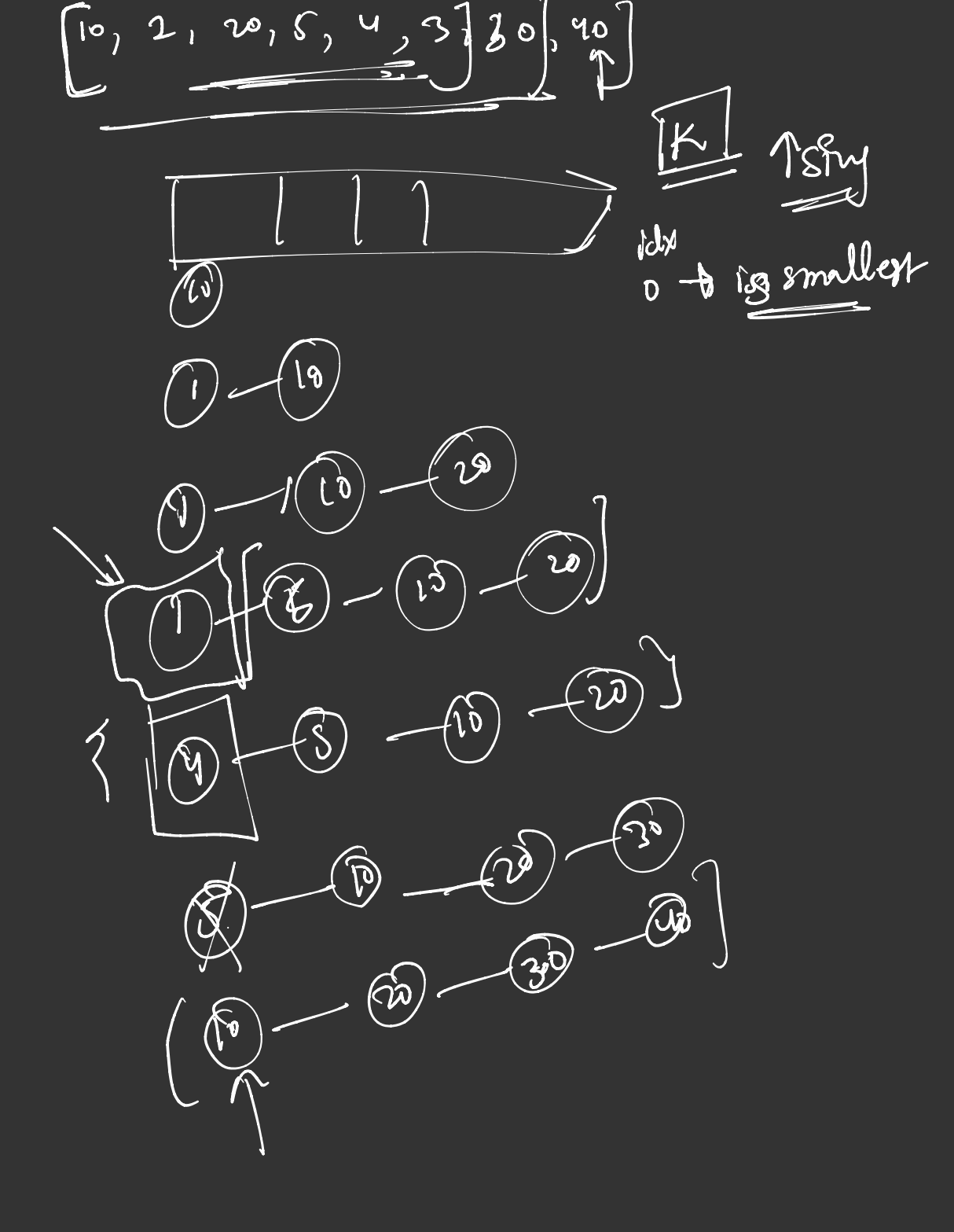
0 3 4 5 6 @ 48 (5)

0 3 4 6 @ 50 (6)

4th largest →

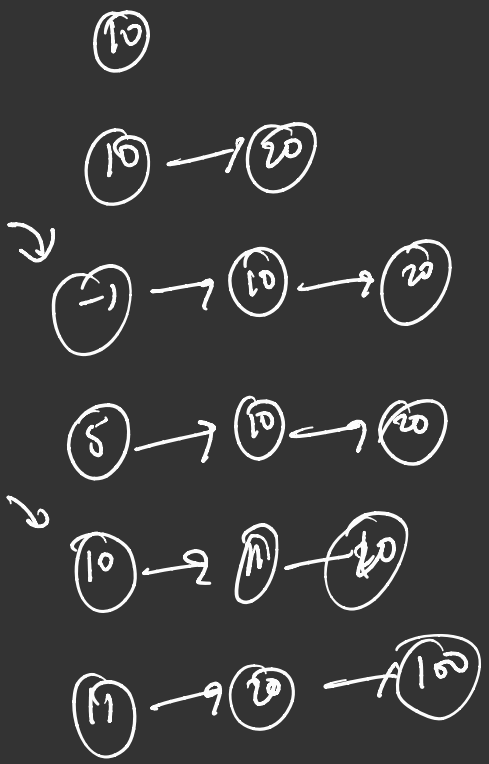
0 3 2 5 4 6 @ 66 ✓ (8)

largest → 0 3 2 5 6 @ 58 (7)


[10, 2, 20, 5, 4, 3] [30, 40]
K
↑Sm
idx
0 → is smallest

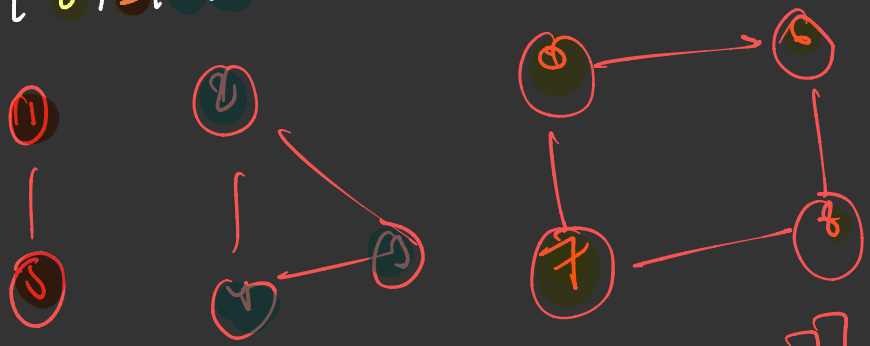
arr = [10, 20, -1, 5, 3, 2, 1, 11, 100, 200, 300]

3rd layer



Get Connected Comp

[0, 1, 2, 3, 4, 5, 6, 7, 8]



[[0, 6, 7, 8], [1, 5], [2, 3, 4]]

for (vtc → 0 ⟶ n) {

if (vtc → not vis) {

dfs

}

}

dfs {

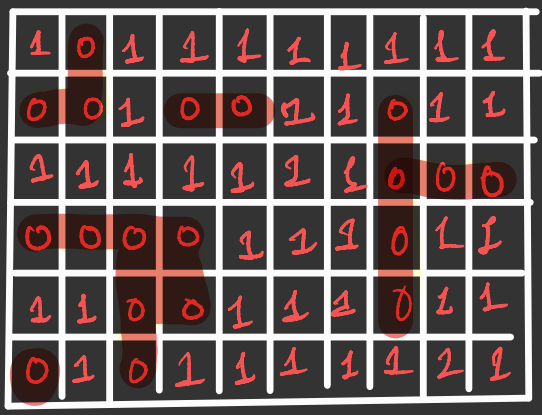
true

call nbrs

}

Number of Islands

0 → land 1 → water



water

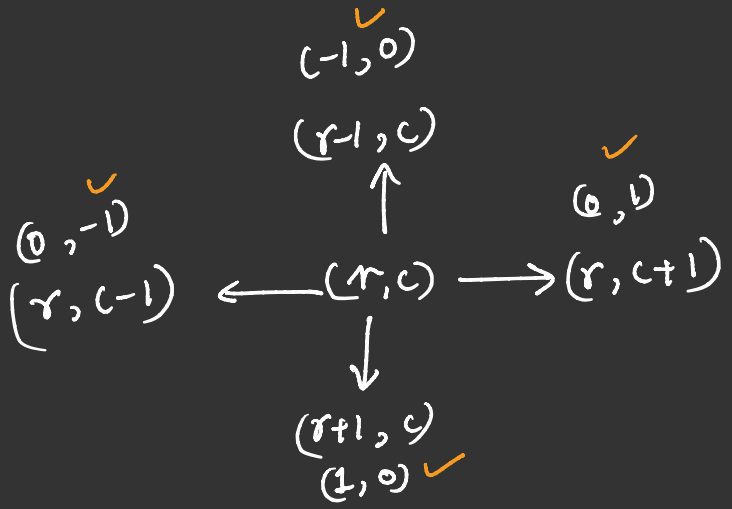
water

water

No. of Islands = 5

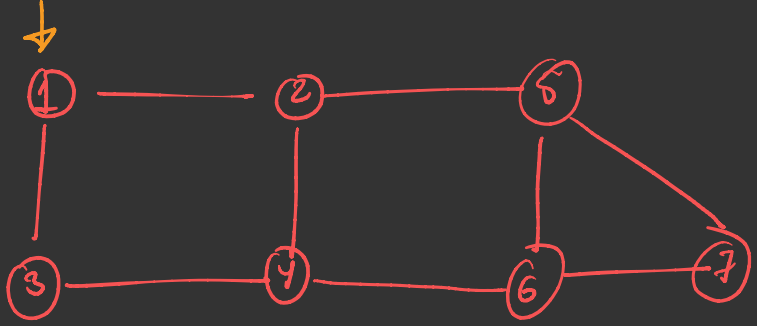
Hint

- GCC (Get connected comp)



dir[4][2] = {{-1,0}, {1,0}, {0,-1}, {0,1}}

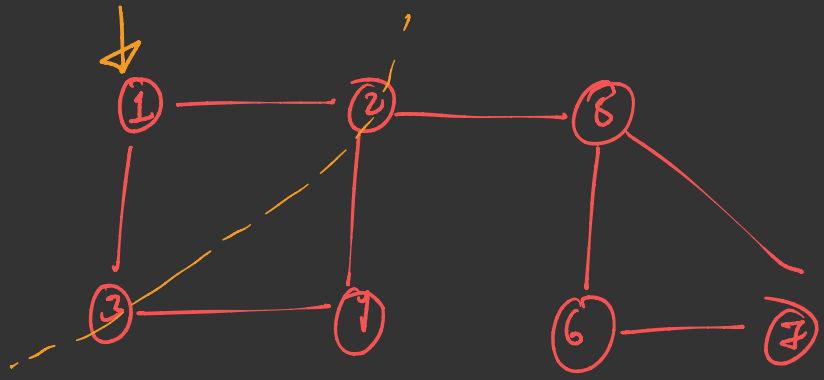
Hamiltonian Path & Cycle

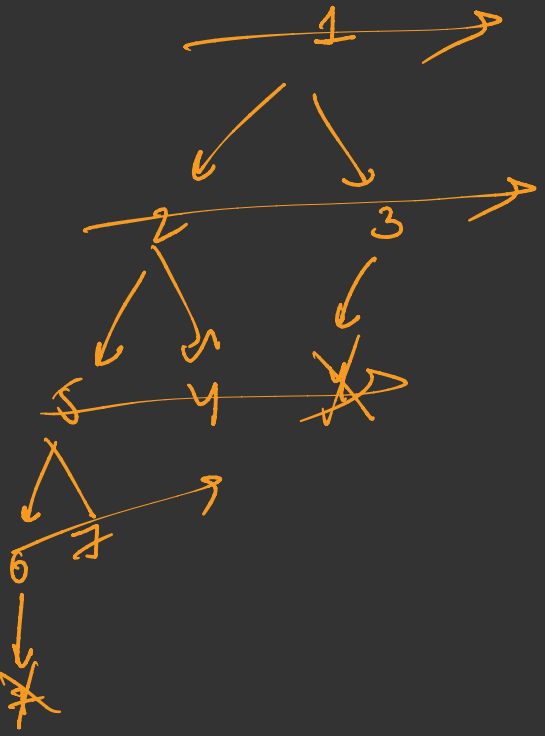


path → "."
cycle → "*"
} at the end

Till Now: Recursive Call (DFS)

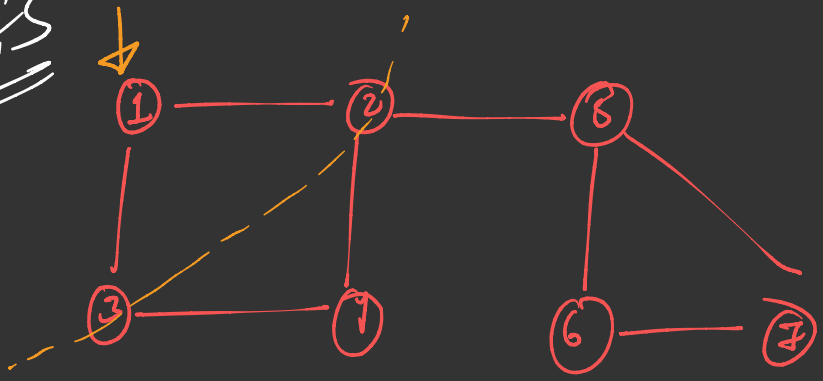
BFS



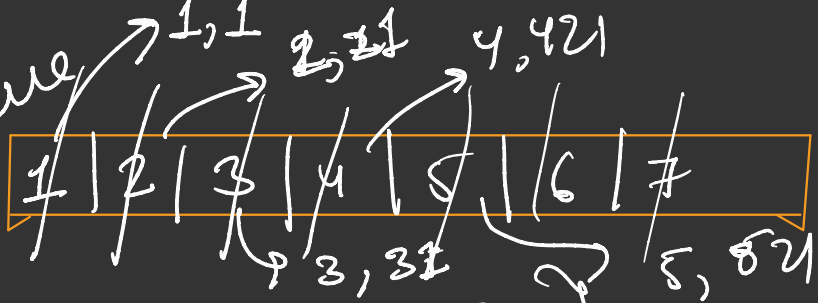


{1, 2, 3, 8, 4, 6, 7}

BFS



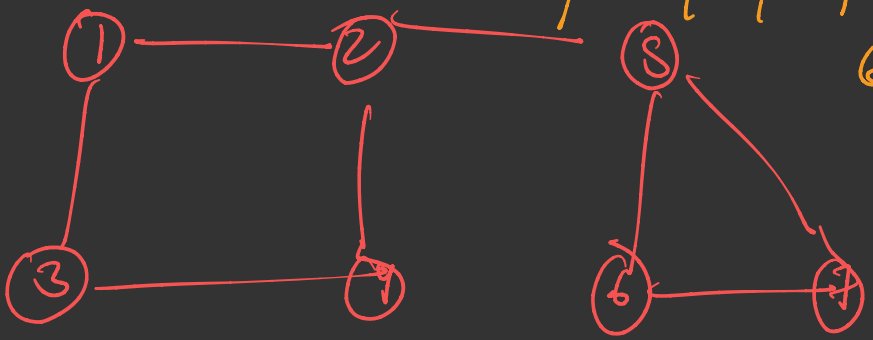
Queue

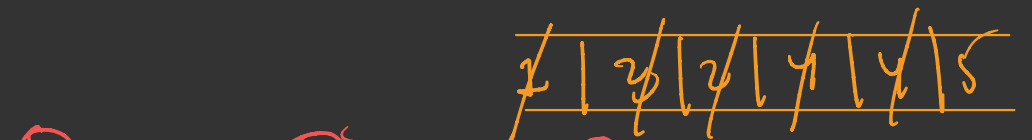


{ 1, 2, 3, 4, 5, 6, 7

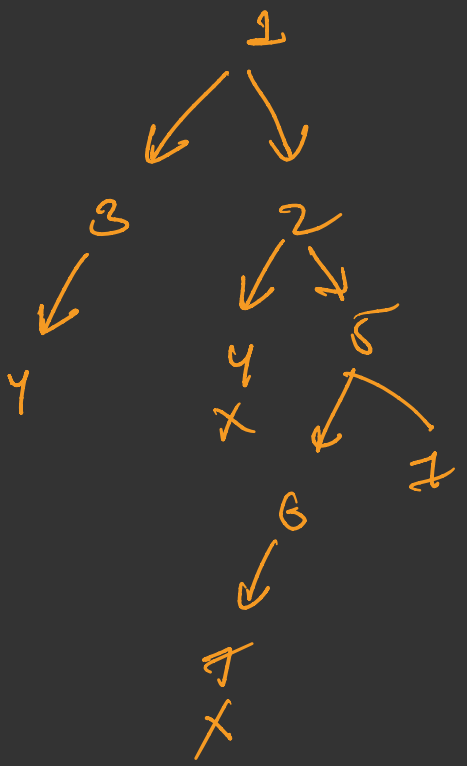
1 @ 1
2 @ 21
3 @ 31
4 @ 421
5 @ 521
6 @ 621

7@ 7521



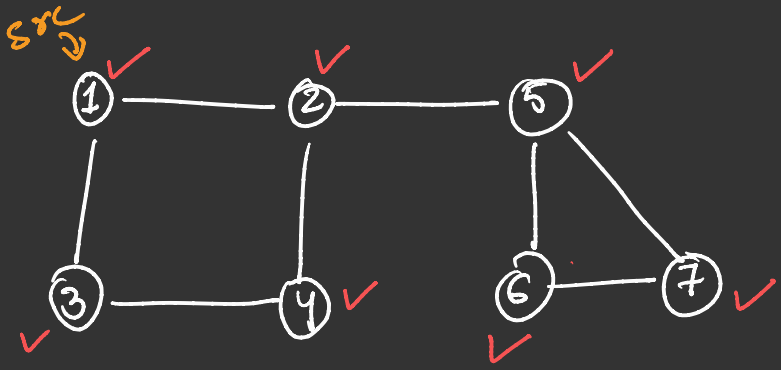


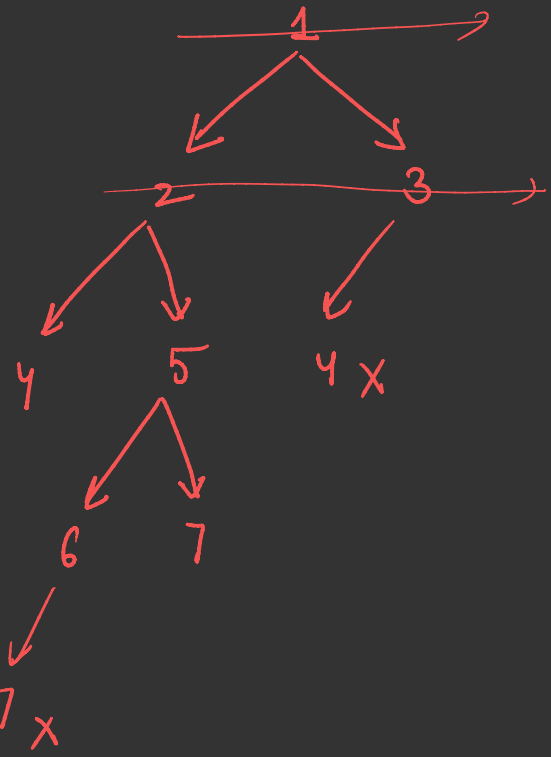
6, 7, 7



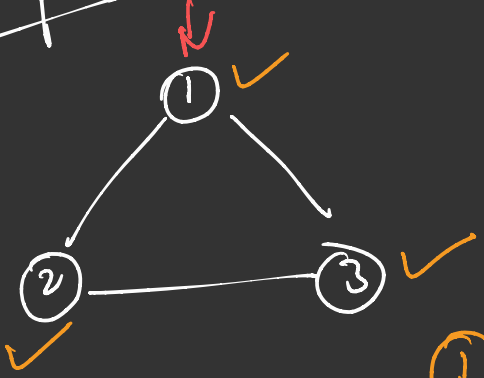
1, 3, 2, 4, 5, 6, 7

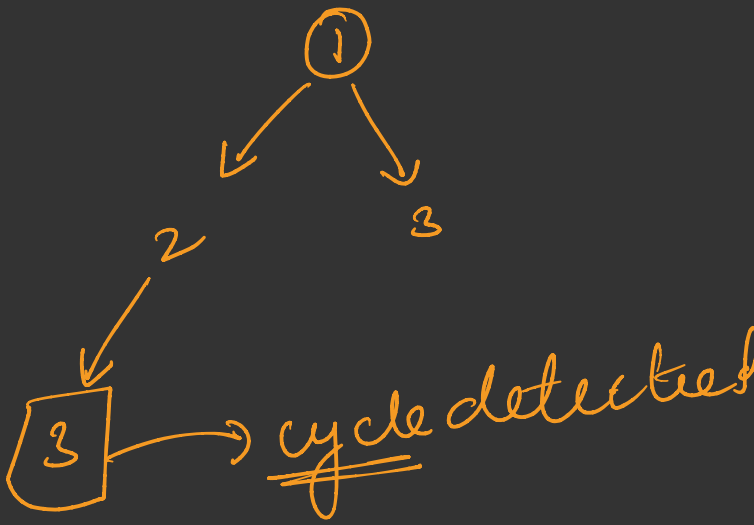
BFS



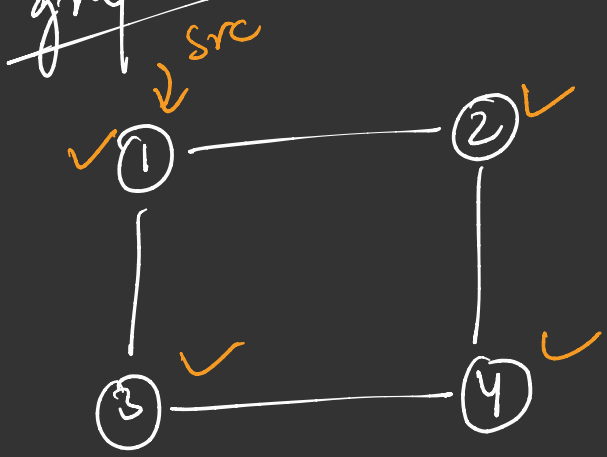


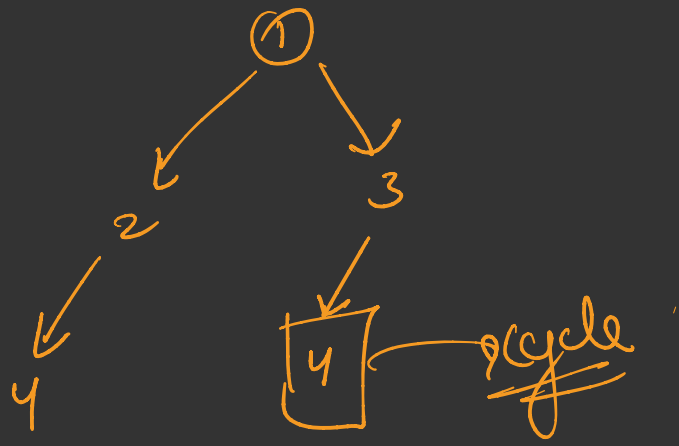
graph 1



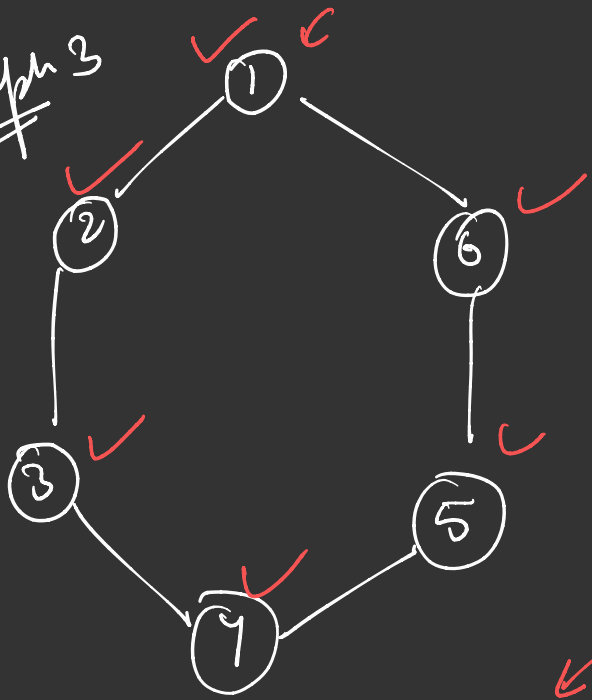


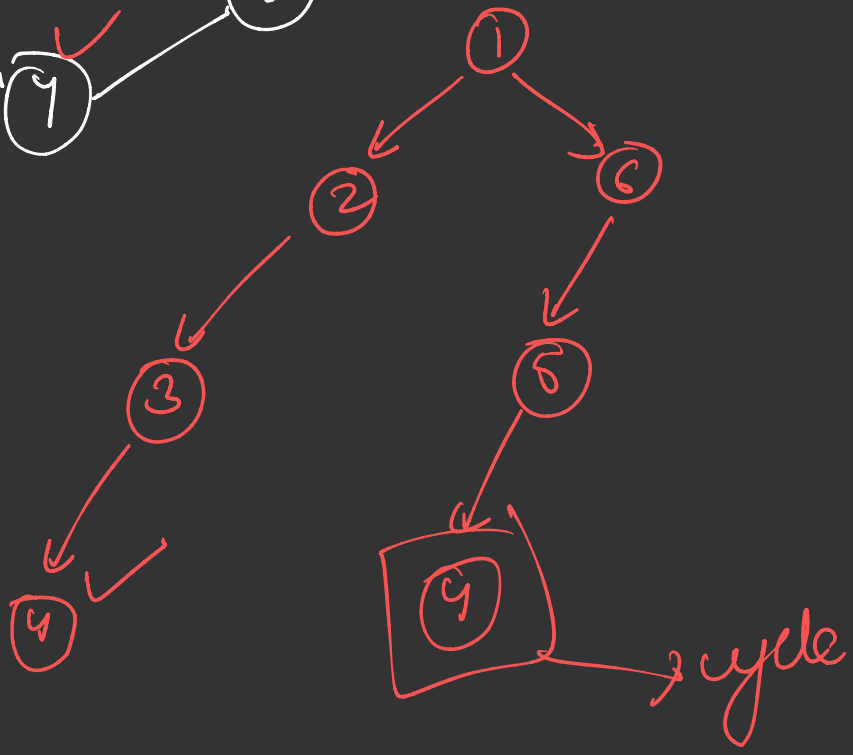
graph 2





Graph 3





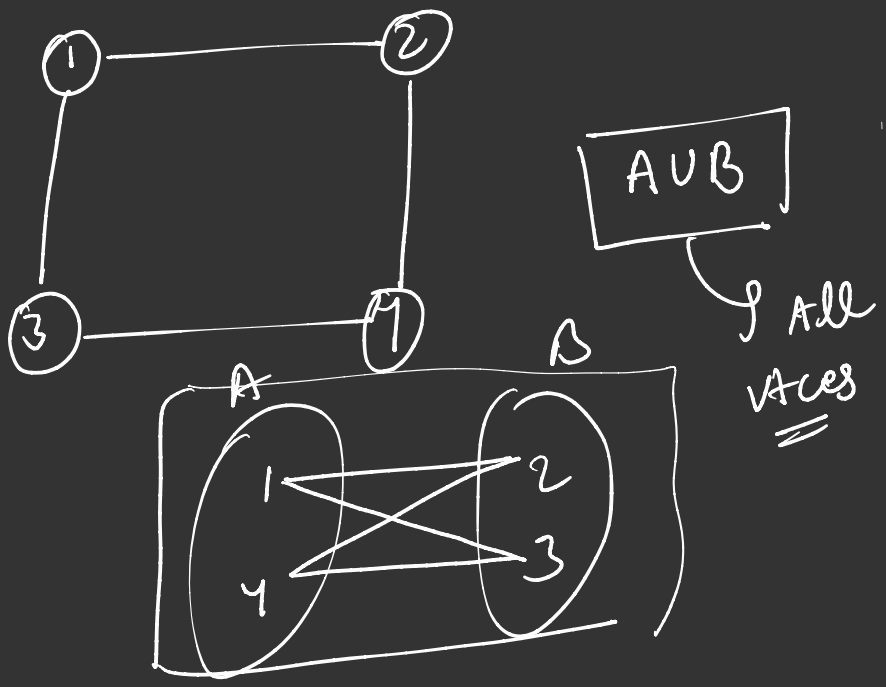
Is GRAPH CYCLIC

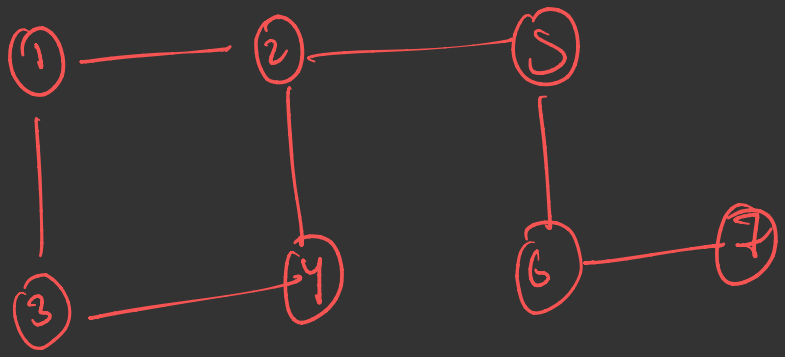
```
   // write your code here
   boolean[] vis = new boolean[vtces];
   for(int i = 0; i < vtces; i++) {
      if(vis[i] == false) {
         boolean hasCycle = isCycle(graph, i, vis);
         if(hasCycle == true) {
            System.out.println(true);
            return;
         }
      }
   }
   System.out.println(false);
}

public static boolean isCycle(ArrayList<Edge>[] graph, int src, boolean[] vis) {
   Queue<Integer> que = new ArrayDeque<>();
   que.add(src);
   while(que.size() != 0) {
      int rSrc = que.remove();
      if(vis[rSrc] == true) {
         // found cycle
         return true;
      }
      vis[rSrc] = true;
      for(var e : graph[rSrc]) {
         if(vis[e.nbr] == false) {
            que.add(e.nbr);
         }
      }
   }

   return false;
}
```

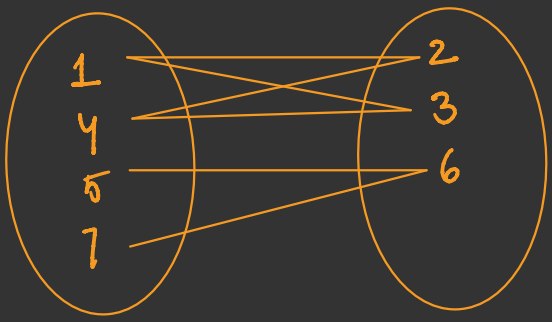
Bipartite

① divide all vtces of a graph into two sets

② these two sets should exhaustive & mutually exclusive



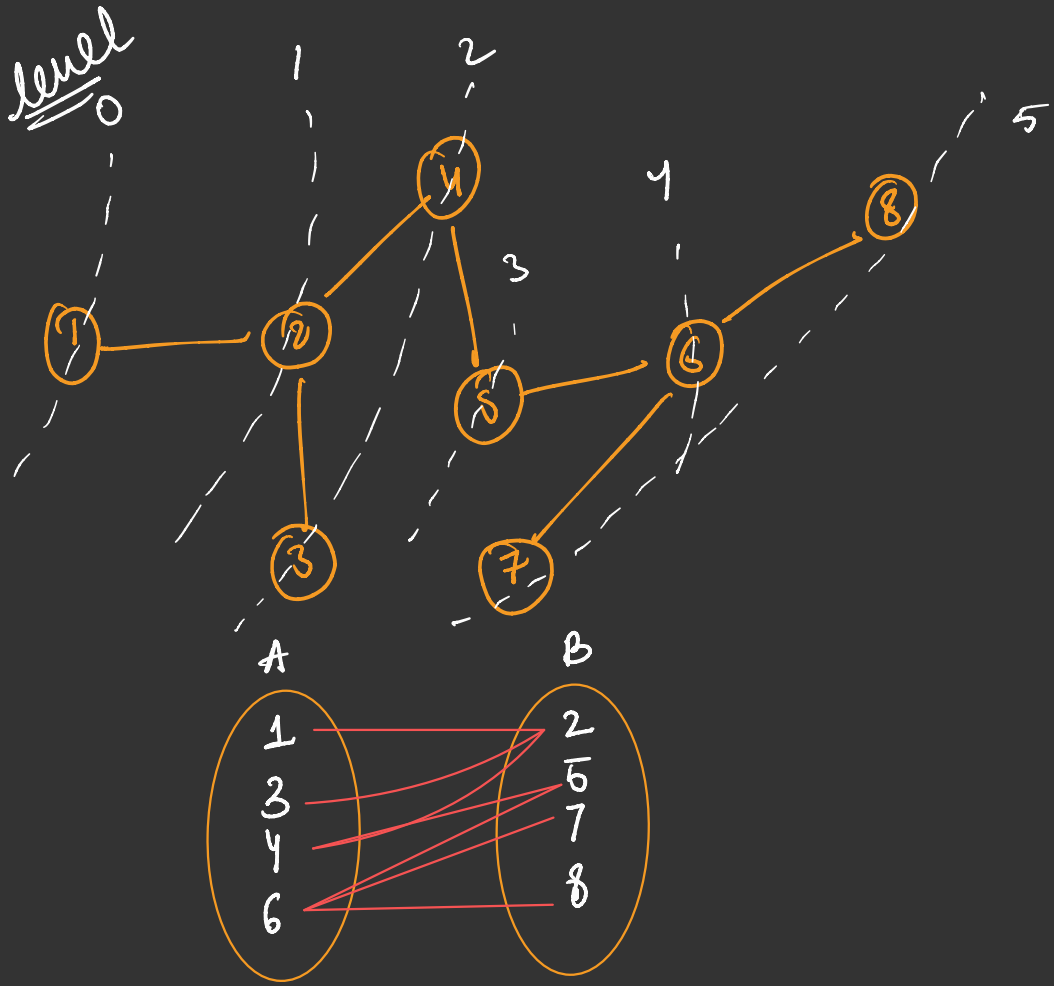


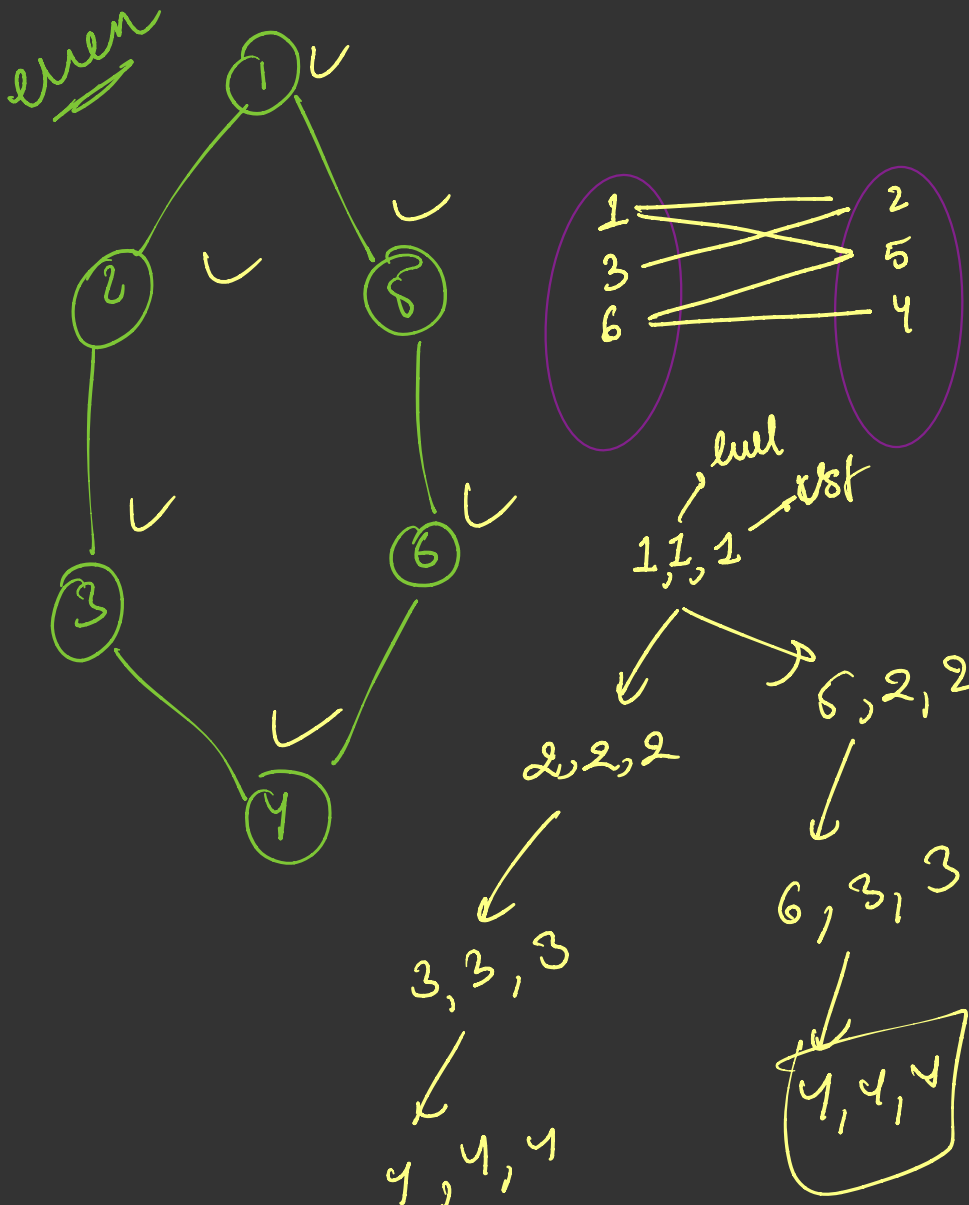


Is Bipartite?

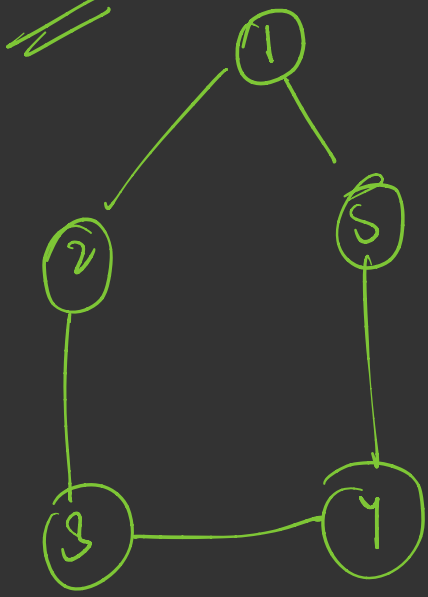
true

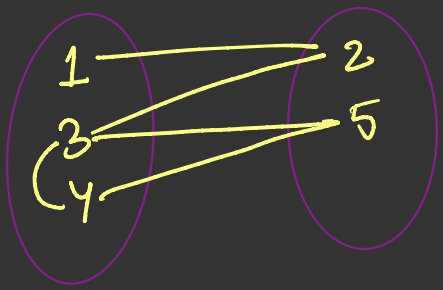
- Acyclic → true
- Cyclic → even → true
  - odd → false

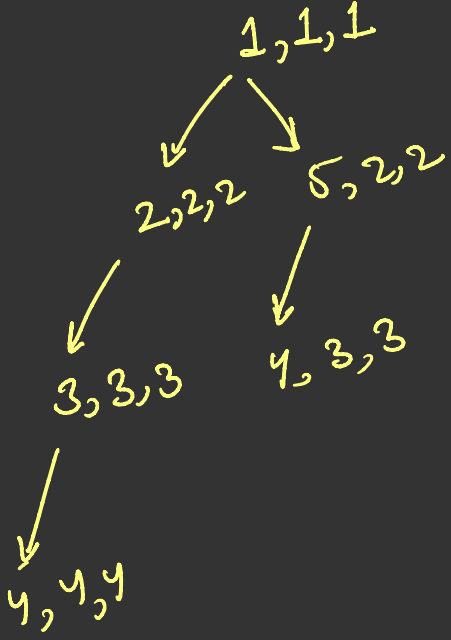


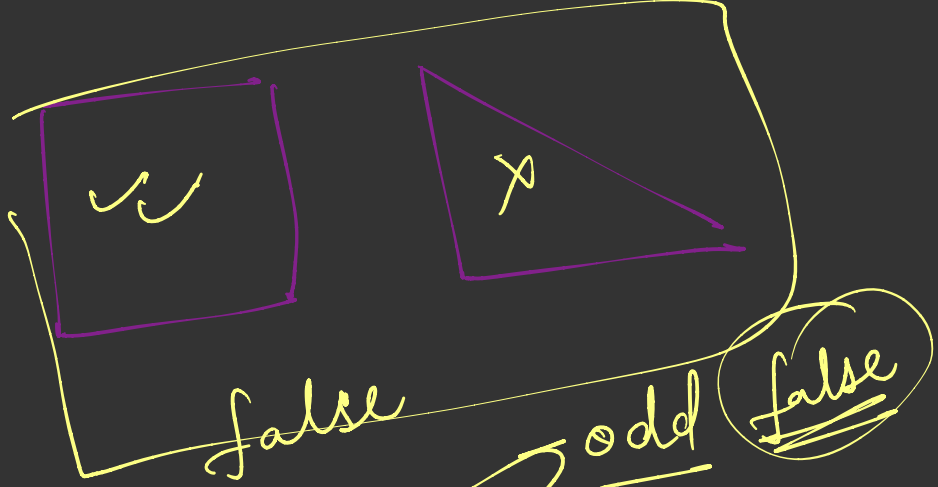

even
1
2
5
3
6
4
1 3 6
2 5 4
last
last
1,1,1
2,2,2
6,2,2
3,3,3
6,3,3
4,4,4
4,4,4

odd









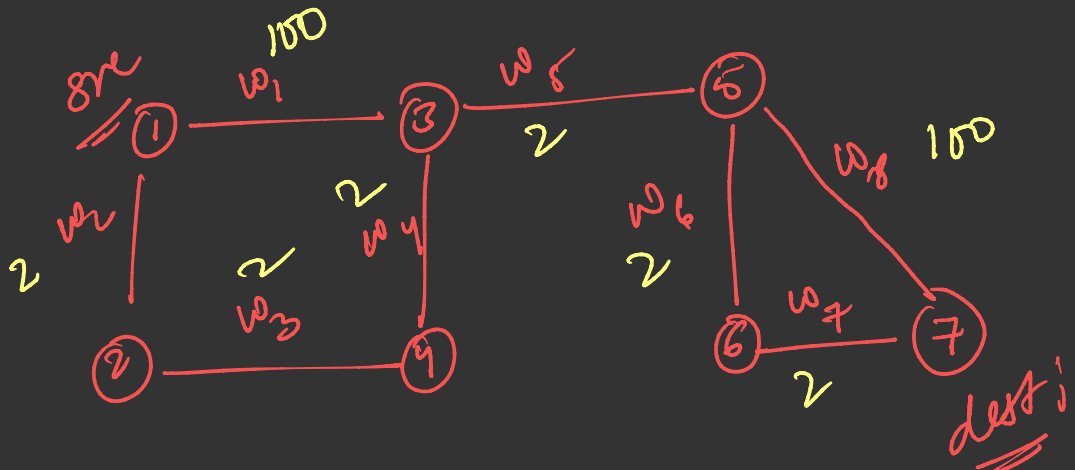
odd → false

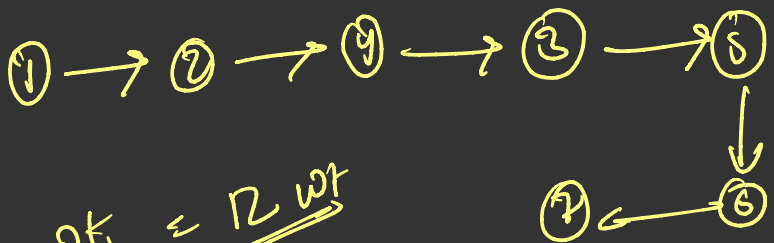
cycle size

even → continue

true

Dijkstra



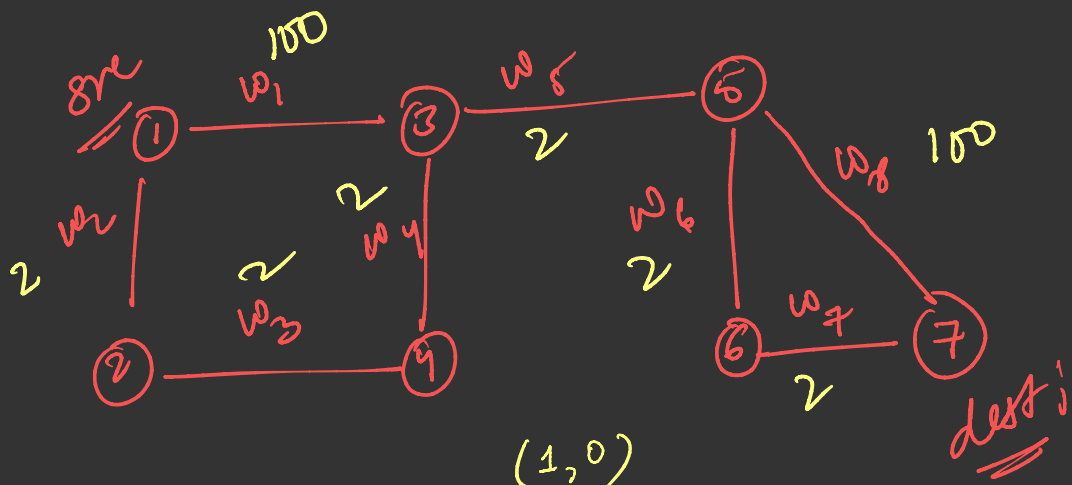


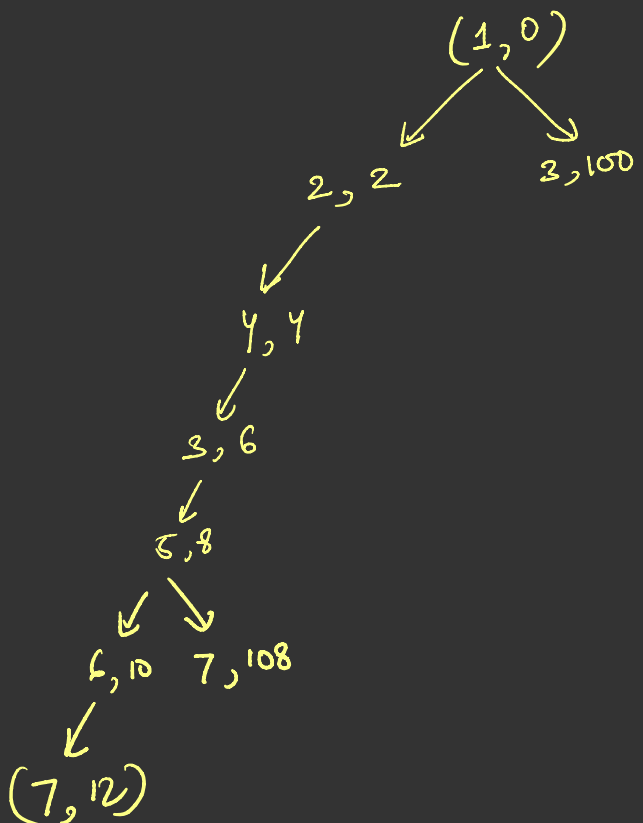
tot wt = 12 wt

→ change queue with PQ
for BFS

wtf on

PQ


src
100
w_1
w_2
2
w_3
2
w_4
2
w_5
2
w_6
2
w_7
2
w_8
100
dest


(1, 0)
2, 2
3, 100
4, 4
3, 6
5, 8
6, 10
7, 108
(7, 12)

```
class Pair {
    int src;
    int wsf;
    Pair(int src, int wsf) {
        this.src = src;
        this.wsf = wsf;
    }
}
```

PriorityQueue<Pair> pq = new PQ<>();

wsf

class Pair implements Comparator<Pair>

{

public int compareTo(Pair o){

return this.wsf - o.wsf;

}

default min

if max type

return o.wsf - this.wsf;

Prim's Algo

Sample Input

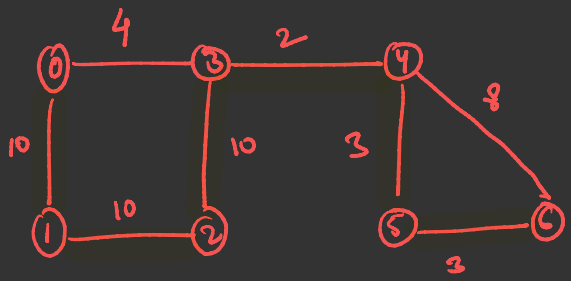
```
7
8
0 1 10
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
```

Sample Output

```
[1-0@10]
[2-1@10]
[3-2@10]
[4-3@2]
[5-4@3]
[6-5@3]
```

child → parent wt

```
Pair {
    int src;
    int parent;
    int wt;
}
```



MST → Min. Spanning Tree { one for whole graph }

↓

All vtces

BFS
- (pq) → wsf {DJK}
- (pq) → wt {prim's}

Topological Sort

Sample Input

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

Sample Output

```
4
5
6
0
1
2
3
```

4560123 → uv

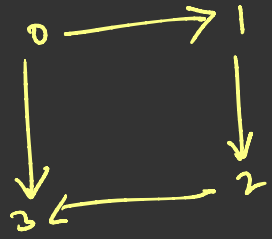
u executed v

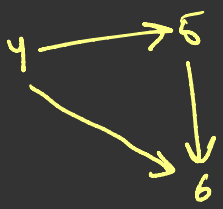
"0123456

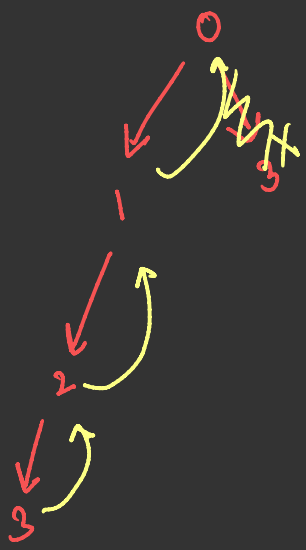
only applicable → DAG → directed, Acyclic

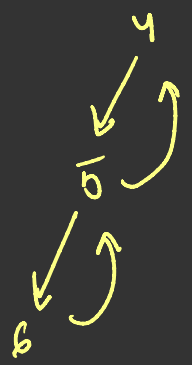
Course scheduling

dfs



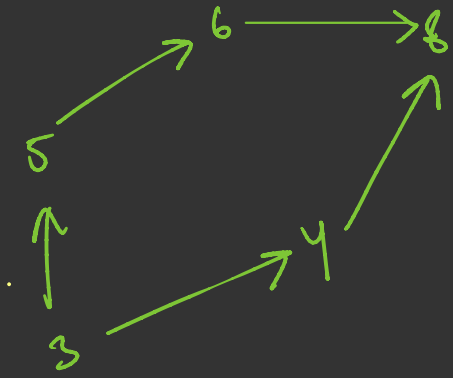






3 2 1 0 6 5 4

4 5 6 0 1 2 3



8 4 6 5 3

3 5 6 4 8