Agenda

1) Topological sort
2) Cycle detection
3) Dijkstra (single source shortest path Algo)

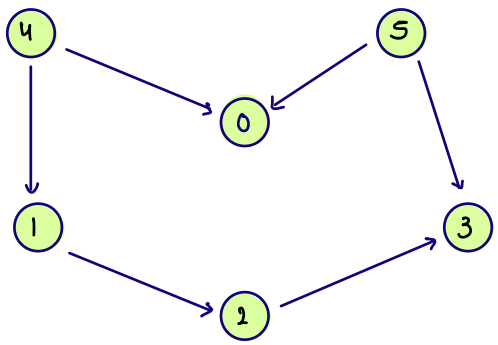
Q.1 Topological sort

linear ordering of graph such that for every directed edge u to v, vertex u comes before v in ordering. (applicable only of DAG)

↓

Directed acyclic graph

eg.



0 1 2 5 4 3 (invalid)

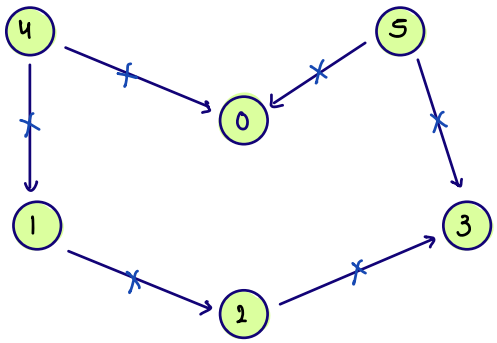
4 1 2 5 0 3 (valid)

4 5 1 2 0 3 (valid)

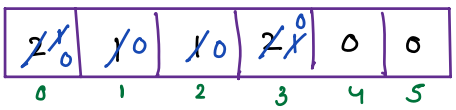
1 5 4 2 3 0 (invalid)

5 4 1 2 0 3 (valid)

Kahn's Algo

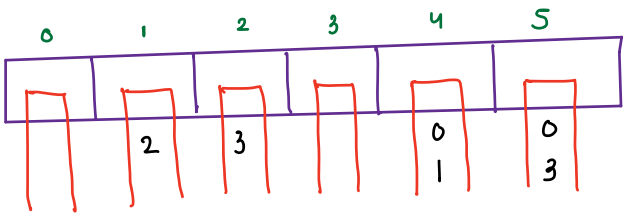


indegree :



q :

4 5 1 0 2 3

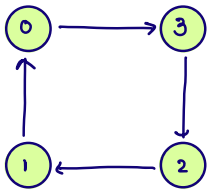



```
void topologicalSort(AL<AL<Integer>> graph){
    int vtx = graph.size();
    int[] indegree = new int[vtx];
    // fill indegree array
    for(int i=0; i<vtx; i++){
        AL<Integer> list = graph.get(i);
        for(int nbr: list){
            indegree[nbr]++;
        }
    }
}
```

```
//applying bfs (with correct starting pts)
Queue<Integer> q = new ArrayDeque<>();

for(int i=0; i<vtx; i++) {
        if(indegree[i]==0) {
                q.add(i);
        }
}

while(q.size()>0) {
        int rmv = q.remove();
        sop(rmv + " ");

        //add appropriate nbs
        AL<Integer> list = graph.get(rmv);
        for(int nbr: list) {
                indegree[nbr]--;
                if(indegree[nbr]==0) {
                        q.add(nbr);
                }
        }
}
}
```

Cycle Detection in Directed graph

eg.1

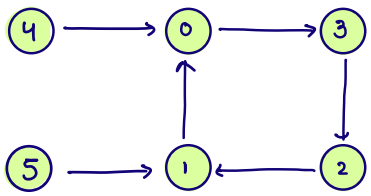


indegree :

0	1	2	3
1	1	1	1

q :

eg.2



indegree :

0	1	2	3	4	5
~~1~~	~~1~~	1	1	0	0

q : ~~4~~ | ~~5~~

4 5

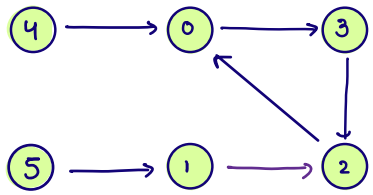
If topological sort is incomplete (like in above two examples)

this indicates directed graph has cycle.

↓

ques: How to detect cycle in directed graph

```
boolean cycleDetect(ArrayList<ArrayList<Integer>> graph) {
    int vtx = graph.size();
    int[] indegree = new int[vtx];
    // fill indegree array
    for(int i=0; i<vtx; i++) {
        ArrayList<Integer> list = graph.get(i);
        for(int nbr: list) {
            indegree[nbr]++;
        }
    }
    //applying bfs (with correct starting pts)
    Queue<Integer> q = new ArrayDeque<>();
    for(int i=0; i<vtx; i++) {
        if(indegree[i]==0) {
            q.add(i);
        }
    }
    int count=0;
    while(q.size()>0) {
        int rmv = q.remove();
        count++;
        // add appropriate nbs
        ArrayList<Integer> list = graph.get(rmv);
        for(int nbr: list) {
            indegree[nbr]--;
            if(indegree[nbr]==0) {
                q.add(nbr);
            }
        }
    }
    if(count<vtx) {return true;}   (cycle is present)
    else {return false;}
}
```



indegree :

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

q :

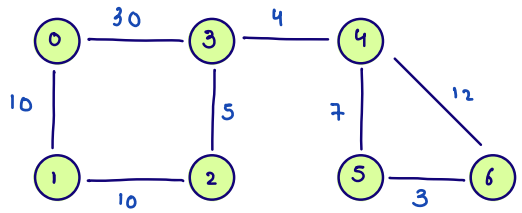
~~4~~	~~5~~	~~1~~

cocnt = ~~1~~ ~~2~~ 3

Shortest path (in terms edges) → BFS

Dijkstra

↳ single source shortest path (wt wise)

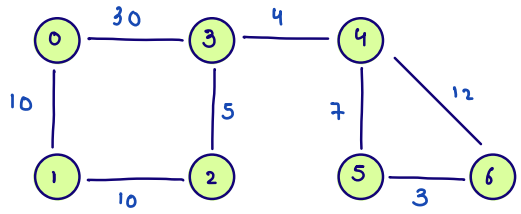


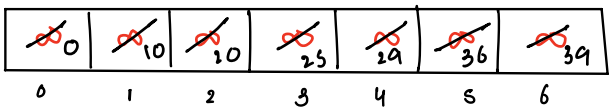
Src = 0

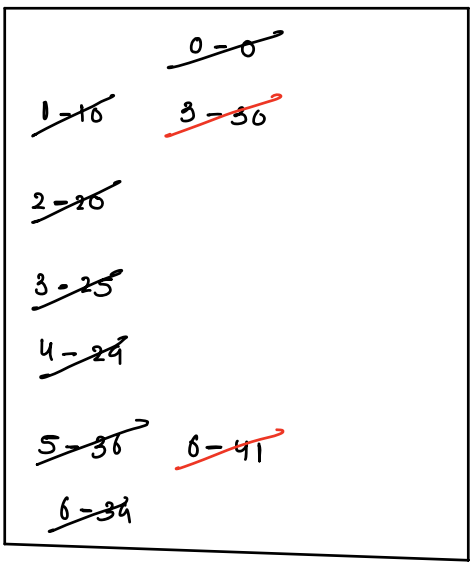
Note: Instead of simple queue we will use PQ (min heap)

0	10	20	25	29	36	39
0	1	2	3	4	5	6

Src = 0





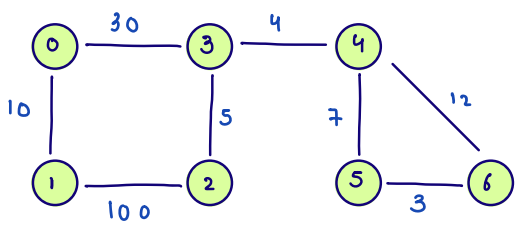


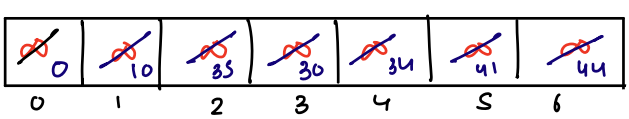
PQ (min Heap)

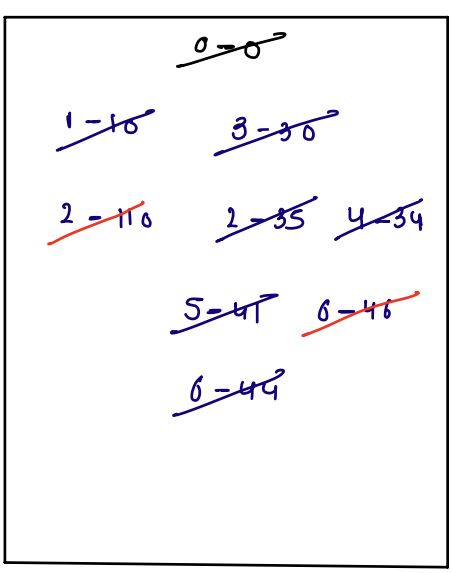
PQ < Helper > pq;

```
class Helper {
    int v;
    int wsf; // weight so far
}
```

Src=0







code in JDF

https://www.interviewbit.com/snippet/26f8b2fde6bbefaa9309/

Normal BFS (queue) → TC: O(V+E)

Dijkstra (Priority Queue) → E log E (V)

[E ≈ V²]

E log V²

E (2 log V)

≈ E log V (VV)

PQ (size)

↓

atmax : E

(single opr

→ log E)

8285040010

↳ whatsapp no.

Slack

Doubts

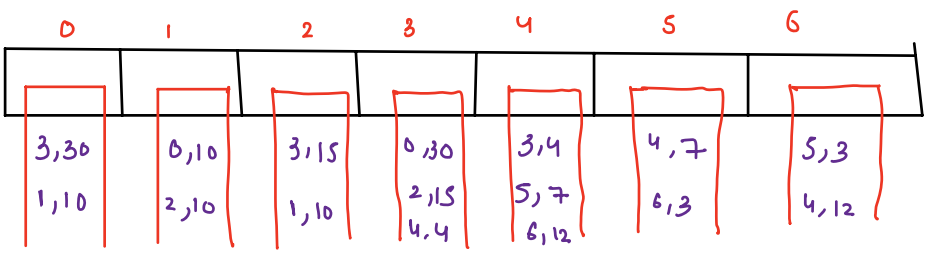
```
int vtx = 7;
int[][]edges = {
    {0,3,30},{0,1,10},{1,2,10},{2,3,5},{3,4,4},{4,5,7},{4,6,12},{5,6,3}
};

ArrayList<ArrayList<Pair>>graph = new ArrayList<>();
for(int i=0; i < vtx;i++) {
    graph.add(new ArrayList<>());
}

//fill graph with edges detail
for(int i=0; i < edges.length;i++) {
    int u = edges[i][0];
    int v = edges[i][1];
    int wt = edges[i][2];

    graph.get(u).add(new Pair(v,wt));
    graph.get(v).add(new Pair(u,wt));
}
```

vtx = 7

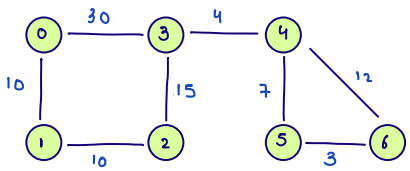
edges

0	3	30
0	1	10
2	3	15
1	2	10
3	4	4
4	5	7
5	6	3
4	6	12

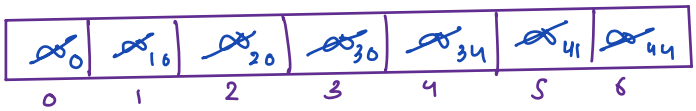
graph:



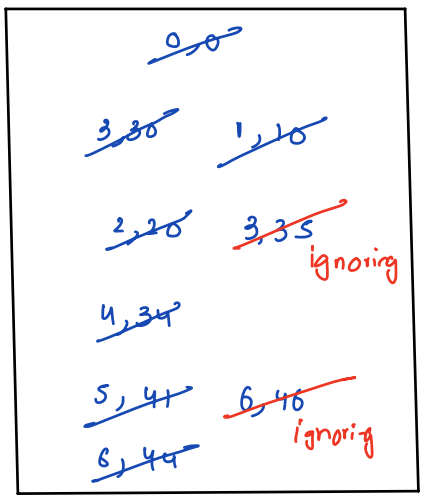
Src = 0



ans:




```
pq.add(new Helper(src,0));

while(pq.size() > 0) {
    Helper rmv = pq.remove();
    int v = rmv.v;
    int wsf = rmv.wsf;

    if(wsf < ans[v]) {
        //work
        ans[v] = wsf;

        //add unvisited nbr
        ArrayList<Pair>list = graph.get(v);
        for(Pair np : list) {
            if(ans[np.v] == Integer.MAX_VALUE) {
                pq.add(new Helper(np.v,wsf + np.wt));
            }
        }
    }
    else {
        //do nothing
    }
}
```



minPQ <Helper>

```
class Helper {
    int v;
    int wsf;
}
```

graph: