

1. remove
2. 1st row
3. visited
4. self loop
5. n-1

1	2	3
4	5	6

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

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

-1-2 ✓
1-3 ✓
1-4 ✓
1-5 ✓
1-6 ✓
1-7 ✓
1-8 ✓

1	1	0	0
2	1	0	1
3	1	0	2
4	1	0	7
5	1	0	13
6	1	0	16
7	1	0	17
8	1	0	17

Single Source Shortest Path
SSSP

1 2
3 4
5 6
7

(i) Dijkstra
(ii) Bellman
Ford

1. remove
2. 1st row
3. visited
4. self loop
5. n-1

```
public void Dijkstra(int src) {  
    PriorityQueue<DijkstraPair> pq = new PriorityQueue<>();  
    HashSet<Integer> visited = new HashSet<>();  
    pq.add(new DijkstraPair(src, "" + src, 0)); // 1, "1", 0  
    while (!pq.isEmpty()) {  
        DijkstraPair rp = pq.poll();  
        if (visited.contains(rp.vtx)) {  
            continue;  
        }  
        visited.add(rp.vtx);  
        System.out.println(rp);  
        for (int nbrs : map.get(rp.vtx).keySet()) {  
            if (!visited.contains(nbrs)) {  
                int cost = map.get(nbrs).get(rp.vtx); // 4 se 5 ke edge ka cost  
                pq.add(new DijkstraPair(nbrs, rp.acqpath + nbrs, rp.cost + cost));  
            }  
        }  
    }  
}
```

1 2 3 4
7 6

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

1 2 0
2 1 2 0 1
3 1 2 0 2
4 1 2 0 7
5 1 2 0 13
6 1 2 0 16
7 1 2 0 17
8 1 2 0 17

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

1 3 2

1	1	0	0
2	1	0	1
3	1	0	2
4	1	0	7
5	1	0	13
6	1	0	16
7	1	0	17
8	1	0	17

1 2 3 4
7 6

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

1 2 3 4
7 6

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

1 2 3 4
7 6

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

1 2 3 4
7 6

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

1 2 3 4
7 6

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

```
public void BellmanFord() {  
    int v = map.size();  
    int[] dis = new int[v + 1];  
    for (int i = 1; i <= dis.length; i++) {  
        dis[i] = 999999999;  
    }  
    List<Edge_Pair> ll = getAllEdge();  
    for (int i = 1; i <= v - 1; i++) {  
        for (Edge_Pair e : ll) {  
            if (dis[e.v2] > dis[e.v1] + e.cost) {  
                dis[e.v2] = dis[e.v1] + e.cost;  
            }  
        }  
    }  
    for (int i = 1; i < dis.length; i++) {  
        System.out.print(dis[i] + " ");  
    }  
}
```

1 2 3 4
7 6

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

1 2 3 4
7 6

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

1 2 3 4
7 6

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

Topological Sort

1 2 3 4
7 6

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

1 2 3 4
7 6

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

1 2 3 4
7 6

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

```
public void Topological() {  
    int[] in = Indgree();  
    Queue<Integer> q = new LinkedList<>();  
    for (int i = 0; i < in.length; i++) {  
        if (in[i] == 0) {  
            q.add(i);  
        }  
    }  
    while (!q.isEmpty()) {  
        int v = q.poll();  
        System.out.print(v + " ");  
        for (int nbrs : map.get(v).keySet()) {  
            in[nbrs]--;  
            if (in[nbrs] == 0) {  
                q.add(nbrs);  
            }  
        }  
    }  
}
```

1 2 3 4
7 6

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

1 2 3 4
7 6

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

1 2 3 4
7 6

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