Graphs 2

Content

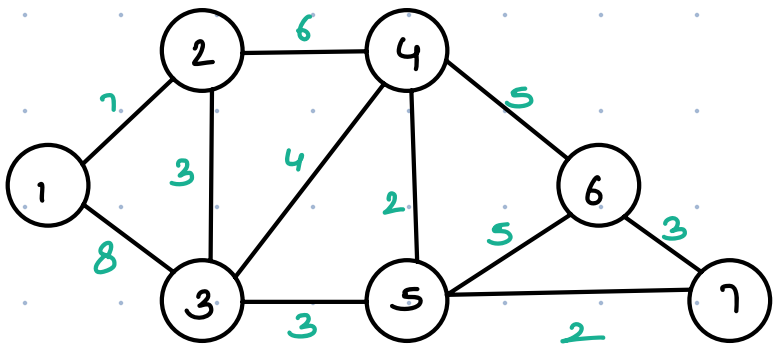
01. Djikstra
02. Bellman Ford Algo
03. Floyd Warshall Algo

} shorkst distance in a graph

- Single source shortest Distance Algo
  - → Weights must be non negative

* Djikstra's Algorithm (Shortest path Algo)

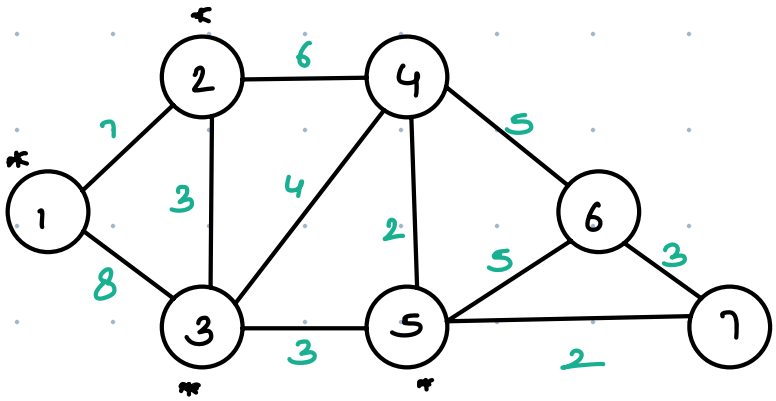
There are N cities in a country, you are living in city-1. Find minimum distance to reach every other city from city 1.



Dist[] = { ~~0~~ 0 7 8 12 11 16 13 }
0 1 2 3 4 5 6 7

Dist[i] = minimum distance to reach i^th city from src city.

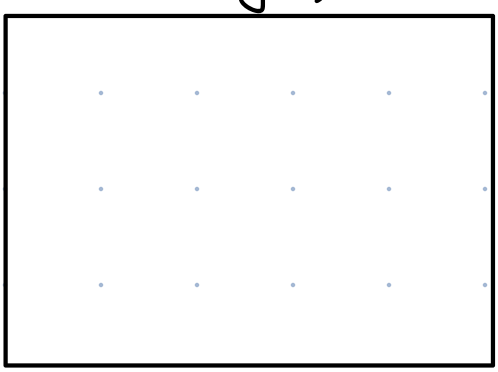
Which DS helps in Djikstra ? ⇒ Min heap or Priority Queue



Dist [ ] =

0	1	2	3	4	5	6	7
∞	0	7	8	12	11	16	13

Priority Queue

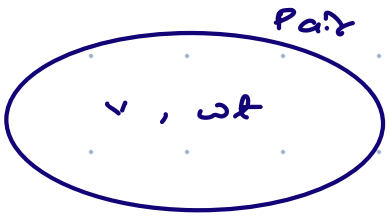


Pair {

int vertex;

int wsf;

}

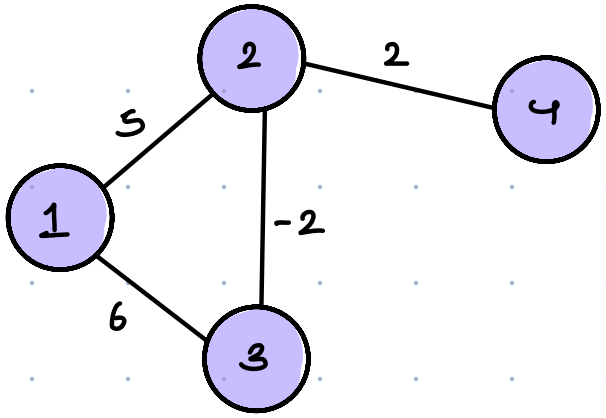



```
int [] Dist = new int [n+1]

Dist [src] = 0

PriorityQueue < pair> pq = new PriorityQueue <>(); // custom comparison

pq.add(new pair(src,0));

while (pq.size()>0) {

    pair rem = pq.remove();

    int vertex = rem.v;

    int wsf = rem.wt;

    if ( Dist [vertex] != Integer.MAX_VALUE) continue;

    else {

        Dist [vertex] = wsf;

        for ( Edge nbr : graph [vertex] ){

            if ( Dist [nbr.vt] == ∞ ){ // add only unvisited nbrs

                pq.insert (new pair (nbr.vt, wsf + nbr.w));

            }

        }

    }

}

return dist [];
```

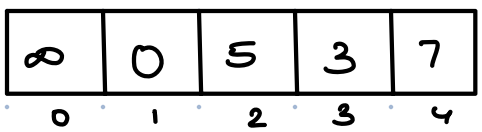
Tc : O(E log E)

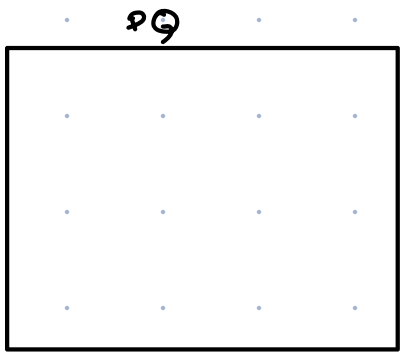
Sc : O(E)

* If we have -ve weights



Shortest Distance from 1 to 4 = 6

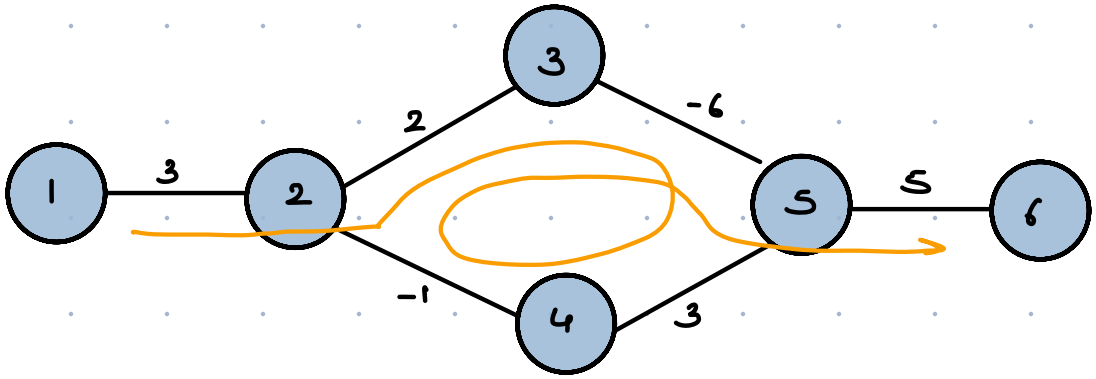




If we apply Dijkstra, to this -ve weight graph, the distance from 1 to 4 ⇒ 7 X

Greedy Algo → it fails for -ve weights.

Is it always possible to find the shortest distance from one node to another node? ⇒ NO



$3 + 2 + (-6) + 5 \Rightarrow 4$

$3 + 2 + (-6) + 3 + (-1) + 2 + (-6) + 5 = 2$

Distance from 1 to 6 $\Rightarrow \infty$

* If a graph doesn't have a -ve edge wt cycle, (but -ve wt edges are allowed), how to find the shortest distance to all nodes?

⇓

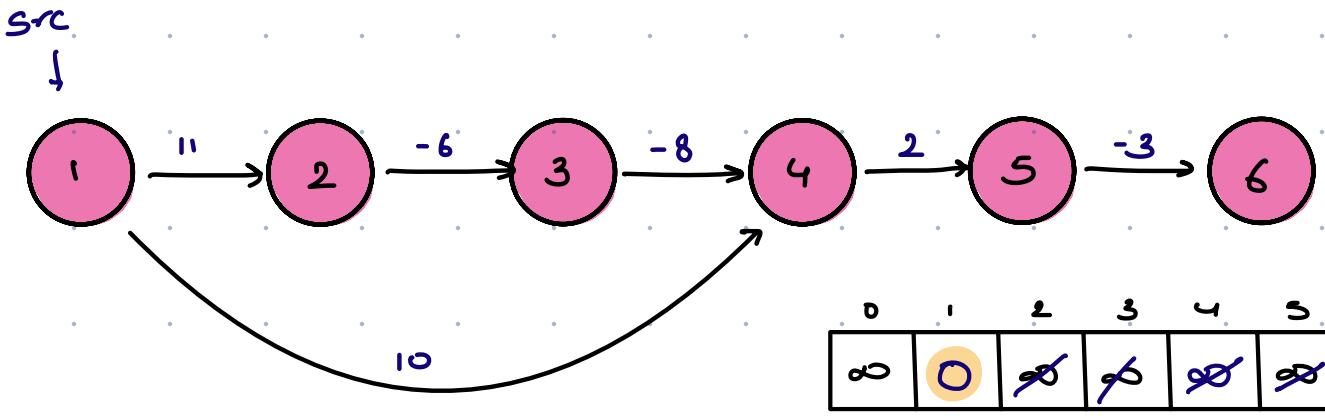
Bellman Ford Algo

Identifying the -ve weight cycles

Key Idea → Minimum distance can be formed by relaxing all the edges at max N-1 times, irrespective of the order in which edges are selected

Why N-1 times?

→ we can have a maximum of N-1 edges between any two nodes



0	1	2	3	4	5	6
∞	0	~~∞~~ 11	~~∞~~ 5	~~∞~~ 10 -3	~~∞~~ 2 -1	~~∞~~ 9 -4

5	6	-3
4	5	2
1	4	10
3	4	-8
2	3	-6
1	2	11

itr 1	itr 2	itr 3	itr 4	itr 5
X	X	✓	X	✓
X	✓	X	✓	X
✓	X	X	X	X
X	X	✓	X	X
X	✓	X	X	X
✓	X	X	X	X

Relaxing an edge means

```
if (dist[u] + wt < dist[v]){
    dist[v] = dist[u] + wt ;
}
```

Note → **Detection of -ve Edge weight cycle**

If there is an updation in N^{th} iteration, that means there exist a path where no. of edges is greater than $N-1$.

⇒ Atleast one of the edge is visited multiple time.

#

```
int dist[] = new int[N+1]
∀ i  dist[i] = ∞
dist[src] = 0
```

```
for (i=1 ; i<N ; i++)        // only helping in doing N-1 iterations
    for (j=0 ; j<=E ; j++){
        int u = graph[j][0]
        int v = graph[j][1]
        int wt = graph[j][2]

        if ( dist[u] + wt < dist[v] ){
            dist[v] = dist[u] + wt;
        }
    }
}
return dist[];
```

TC : O(N * E)

SC : O(1)

* Shortest Path from Every Node to Every other Node

01. All edge wts are +ve

→ Apply Dijkstra considering every node as the source Node

TC : O(N * E log E)

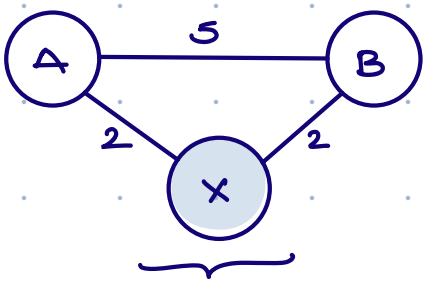
→ Bellman Ford Algo

→ Apply Bellman's ford Algo considering every node as source Node.

TC : $O(N^2 E)$

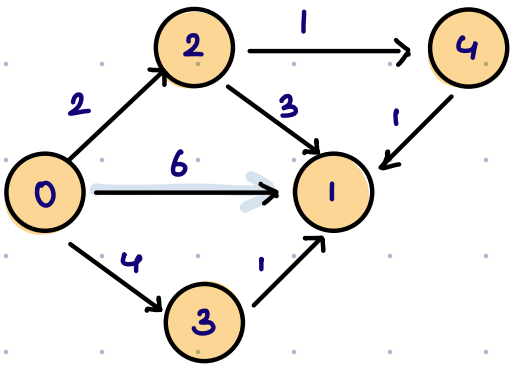
→ FloyD Warshall Algo 10:15 → 10:25

(Multisource shortest Path Algo)



Intermediate Node

∀ nodes, consider them as the intermediate node & relax all the connected edges



d[0][1]

0 → 1 ⇒ 6

(0 → 2) + (2 → 1)

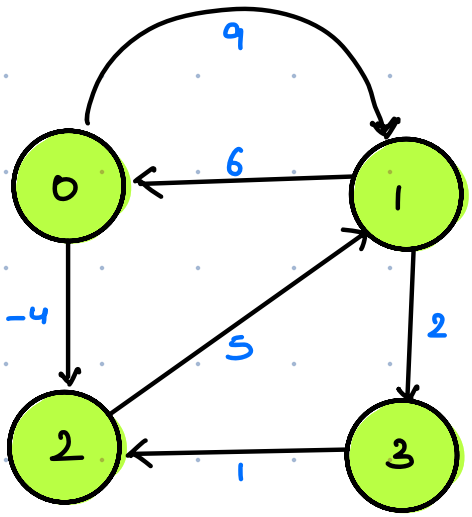
2 + 3 = 5

(0 → 3) + (3 → 1)

4 + 1 = 5

$(0 \to 4) + (4 \to 1)$

$3 + 1 = 4$



Adjacency matrix

	0	1	2	3
0	0	9	-4	∞
1	6	0	∞	2
2	∞	5	0	∞
3	∞	∞	1	0

Node - 0 as intermediate node

	0	1	2	3
0	0	9	-4	∞
1	6	0	2	2
2	∞	5	0	∞
3	∞	∞	1	0

$d[i][0] + d[0][j] < d[i][j]$

$d[1][0] + d[0][2] < d[1][2]$

$d[1][0] + d[0][3] < d[1][3]$

$d[2][0] + d[0][1] < d[2][1]$

$d[2][0] + d[0][3] < d[2][3]$

$d[3][0] + d[0][1] < d[3][1]$

$d[3][0] + d[0][2] < d[3][2]$

Node - 1 as intermediate node

	0	1	2	3
0	0	9	-4	11
1	6	0	2	2
2	11	5	0	7
3	∞	∞	1	0

$d[i][1] + d[1][j] < d[i][j]$

$d[0][1] + d[1][2] < d[0][2]$

$d[0][1] + d[1][3] < d[0][3]$

$d[2][1] + d[1][0] < d[2][0]$

$d[2][1] + d[1][3] < d[2][3]$

$d[3][1] + d[1][3] < d[3][0]$

$d[3][1] + d[1][2] < d[3][2]$

Node - 2 as intermediate node

	0	1	2	3
0	0	1	-4	3
1	6	0	2	2
2	11	5	0	7
3	12	6	1	0

$d[i][2] + d[2][j] < d[i][j]$

Node - 3 as intermediate node

	0	1	2	3
0	0	1	-4	3
1	6	0	2	2
2	11	5	0	7
3	12	6	1	0

d[i][3] + d[3][j] < d[i][j]

```
for (k=0; k<N; k++){
    for (i=0; i<N; i++){
        for (j=0; j<N; j++){
            if (dist[i][k] + dist[k][j] < dist[i][j]){
                d[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}
```

TC : O(N³)

SC : O(1)