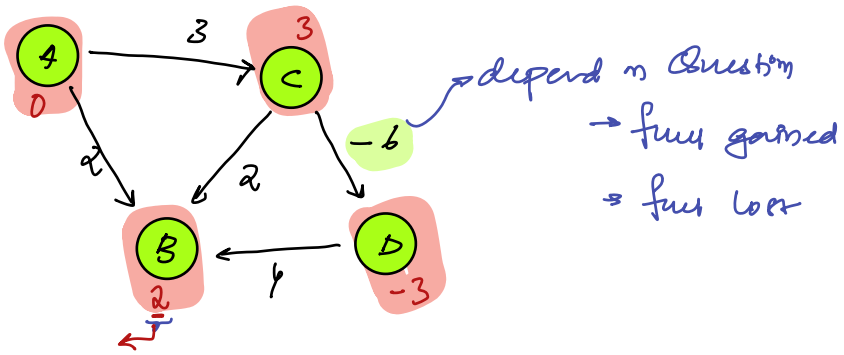
Today's Content:

- Belman Ford ✓ → Inverter
- Floyd Warshal Algorithm
- Prims

Dijkstra's With Negative Edges : source A → { find Shortest Path Nodes }

Eg1:



// Shortest Path A → B : 2 → // According to dijkstra we are getting 2

// A → C → D → B : 1 → correct ans = 1

With Negative Edges Dijkstra's fail

→ Dijkstras Idea

a) Out of all Unblasted Node take node with min value

b) Blast above node & iterate on all its edges & update its adjacent nodes

// graph N Node : length of longest Path : N-1

New Idea : { Bellman - Ford }

a) Using every edge update all nodes

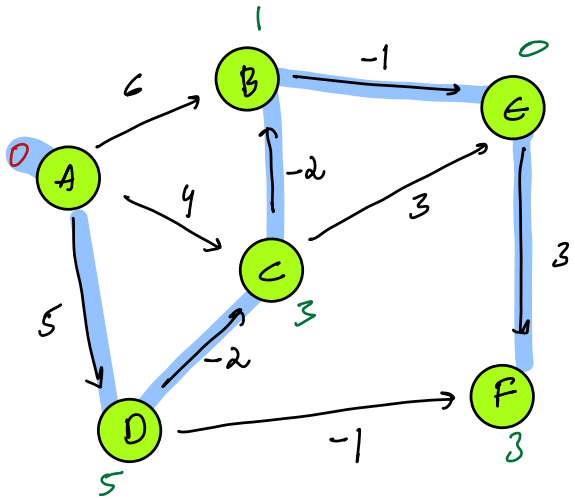
b) Repeat above proces N-1 times

c) Note : { if no change in dist[] in full iteration stop }

TC : { N-1 } x { E }

SC : O(N) → dist array

→ Bellman ford:

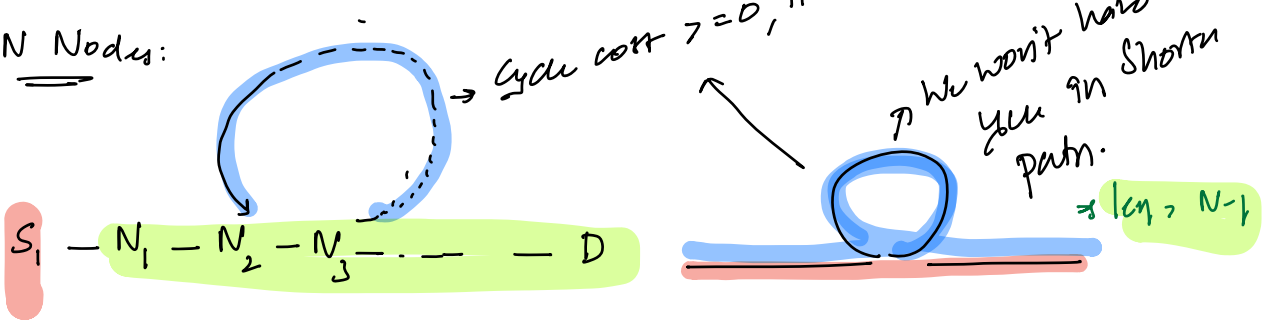


dist:	A	B	C	D	E	F
Init:	0	∞	∞	∞	∞	∞
Itr1:	0	2	3	5	5	4
Itr2:	0	1	3	5	1	4
Itr3:	0	1	3	5	0	3
Itr4:	0	1	3	5	0	3
Itr5:	0	1	3	5	0	3
	✓	✓	✓	✓	✓	

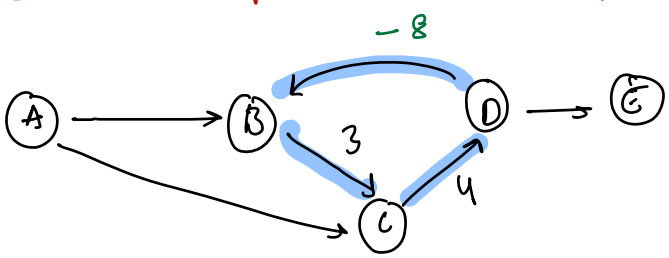
Edges:

A→B, A→C, A→D, B→E, C→B, C→E, D→C, D→F, E→F

// N Nodes:



Note: If graph has Negative cycle, than Bellmanford won't work



→ If graph has -ve cycle we cannot get min Path

```
// Each Edge Should Contain {u, v, w},  u → v  Weight w

                                  { Construct Adj Lists
int BellmanFord( list< pair<int,int> > g[], N, s, D){

    dist[N+1] = {INF}
    dist[s] = 0
    for(i=1; i< N; i++){          { iterating all Edges N-1 times
        bool Flag = false
        // Iterate to all Edges & update all nodes
        for(j=1; j<=N; j++){
            for(k=0; k< g[j].size(); k++){
                pair<int,int> x = g[j][k]
                V= x.first  W= x.second;
                j ---W---> V
                if( dis[V] > dis[j]+W){
                    dist[V] = dis[j]+W
                    flag = True
                }
            }
        }
        if( flag == False){ break}
    }
    return dist[d];
}
```

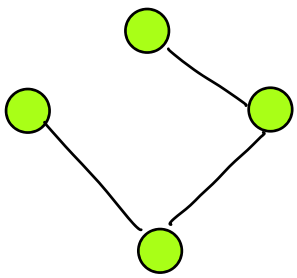
TC: $O(N*E)$ → Best : $O(N^2)$, Worst : $O(N^3)$

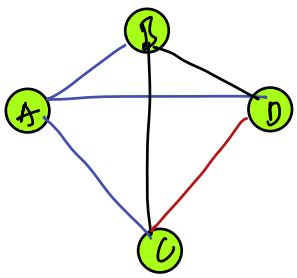
-ve cycle ❌ won't work
1) -ve Edges : ✓
2) +ve Edges : ✓

// graphs:

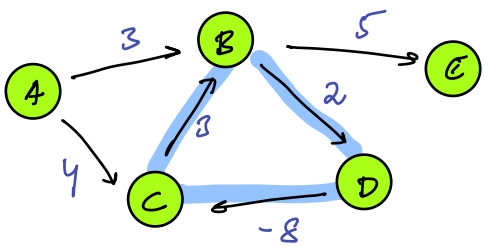
N Nodes, Edges → N-1 Edges for it to be connected : {Sparse graphs}

→ $O(N^2)$ Edges ⇒ {Dense graphs}





// Give a graph check if it contains negative cycle?



→ How many time we need to iterate on all Edge & update 4 times

→ After 4 times we should ideally get shortest path

→ Say we update 5th time, if value are again changing, →

→ Bellman-Ford fails

→ Negative Cycle Exists

Step

1) update N time check dist[] value are getting changed, if value are getting changed. In That case cycle is present

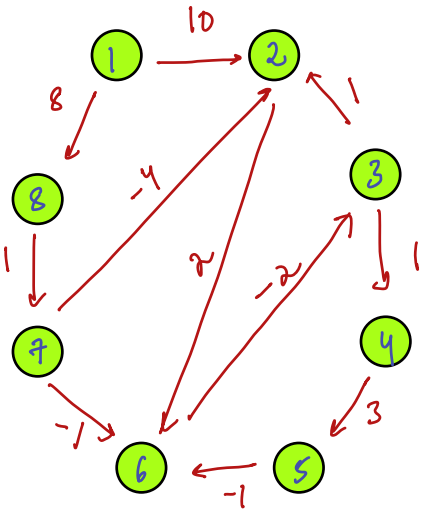
TC: $O(N*E)$

// 10:30 pm

// Floyd - Warshall {All pair shortest Path} → {Dynamic Programming}

{-ve cycle won't work}

↳ We will take every pair of S & D & get shortest path between them.



N Nodes: → N^2 pairs

→ ans [N+1][N+1] ?

→ {For every Node Apply Bellman - Ford}

TC: N * N * E
- Best: N * N * N
- Worst: N^4

// dp [i] [j] [k] = Shortest path from i - j

u v {nodes we can use in between u → v}

// dp [i] [j] [3] = Shortest from i - j, in between nodes (1, 2, 3)

// dp [i] [j] [5] = Shortest from i - j, in between nodes (1, 2, 3, 4, 5)

// dp [i] [j] [k] = Shortest from i - j, in between nodes (1, 2, 3 .. k)

i {1, 2, 3, .. k} j

min

Case - 1

i {1, 2, 3, .. k-1} j

{dp [i] [j] [k-1]

use k^{th} node between

i {1, 2, .. k-1} k^{th} {1, 3 .. k-1} j

dp [i] [k] [k-1] + dp [k] [j] [k-1] }

dp[N+1][N+1][N+1]

Base Condition: k = 0

↳ dp[i][j][0] = min distance between i-j with no node in between

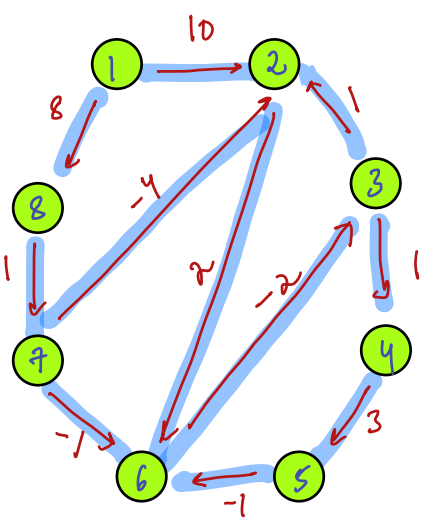
↳ given Edge/ Adjacent Node

```
k=1; k<=N; k++){
    i=1; i<=N; i++){
        j=1; j<=N; j++){
            dp[i][j][k] = min(dp[i][j][k-1], dp[i][k][k-1] + dp[k][j][k-1])
        }
    }
}
```

TC: O(N3) SC: O(N3)

dp[s][d][0] = {using Edge Fill the matrix}

//



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

dp[i][j][1] = { i-j shortest path with node 1 in between }

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

1-2 → ① 1-2

1-1 + 1-2 → 10

0 10

2-3 →

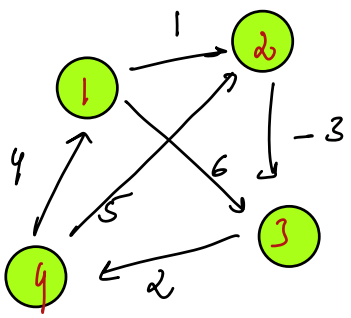
2-1+1-3

dp[i][j][2] = { i-j shortest path with node {1, 2} }

	0	1	2	3	4	5	6	7	8
0									
1	0	10	∞	∞	∞	12	∞	8	
2	∞	0	∞	∞	∞	2	∞	∞	
3	∞	1	0	1	∞	∞	∞	∞	
4	∞	∞	∞	0	3	∞	∞	∞	
5	∞	∞	∞	∞	0	-1	∞	∞	
6	∞	∞	-2	∞	∞	0	∞	∞	
7	∞	-4	∞	∞	∞	-1	0	∞	
8	∞	∞	∞	∞	∞	∞	1	0	

//

Eg:



dp[5][5][0] ⟶ dp[5][5][1] : {Include only node 1 between}

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

	0	1	2	3	4
0					
1	0	1	6	∞	
2	∞	0	-3	∞	
3	∞	∞	0	2	
4	4	5	10	0	

dp[5][5][2] : = {Include {1, 2}}

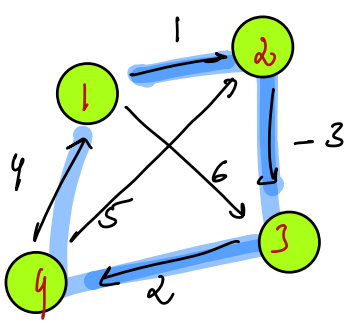
	0	1	2	3	4
0					
1	0	1	-2	∞	
2	∞	0	-3	∞	
3	∞	∞	0	2	
4	4	5	2	0	

dp[5][5][3] = {Include {1, 2, 3}}

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

dp[5][5][4] = {Include {1, 2, 3, 4}}

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



↓

SC: We only need prev matrix data ⇒ $O(N^2)$

- Thursday → proofs
- Bipartite
- Strong Connected Components

⇒ Thursday optional