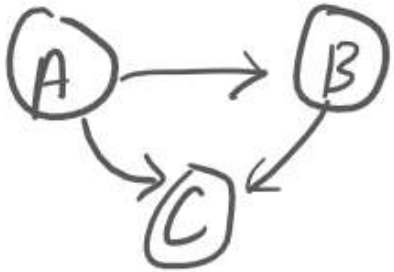
BELLMAN FORD ALGORITHM (GRAPH)



LinkedIn @manojofficialmj

📂 2. Bellman Ford Algorithm

🚀 *It is also used to find SSSP from source to all nodes*

🚀 ***SSSP:*** *SINGLE SOURCE SHORTEST PATH*

What is the Bellman-Ford Algorithm:

This loop-based algorithm is used to find the shortest path from a single source node to all other destination nodes and detect the negative cycle in the graph.

Note 1: *It is also used to find all shortest path from source to all other nodes in a negative and positive weighted, directed or undirected graph.*

Note 2: *It is also used to when a graph have the negative cycle.*

Step 1: to find the SSSP

Relaxation N-1 times to get the all shortest path from single node.

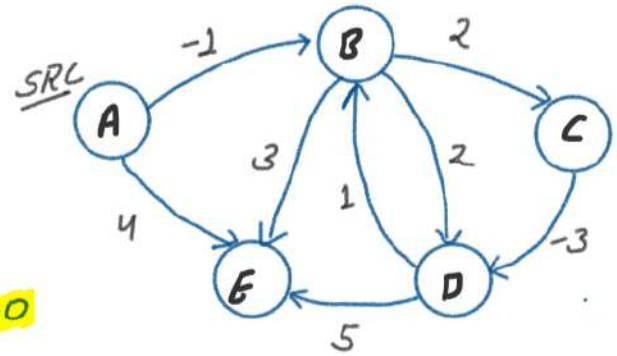
→ (Dist[Node] + weight < Dist[Node])

Step 2: to detect the negative cycle

One more relaxation after N-1 times to detect the negative cycle

↳ Flag = True / False

Example 1:



Initial state
dist[src]=0

dist	0	∞	∞	∞	∞
	0 A	1 B	2 C	3 D	4 E

RELAX Edges →

$A \xrightarrow{-1} B$

$B \xrightarrow{2} C$

$C \xrightarrow{-3} D$

$D \xrightarrow{5} E$

$A \xrightarrow{4} E$

$B \xrightarrow{3} E$

$B \xrightarrow{2} D$

$D \xrightarrow{1} B$

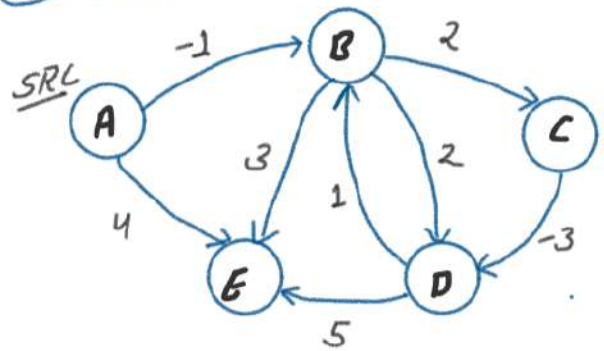
Key	value
A	{{B, -1}, {E, 4}}
B	{{C, 2}, {D, 2}, {E, 3}}
C	{{D, -3}}
D	{{E, 5}, {B, 1}}
E	×

Adj list

DRY RUN

N = Total Node = 5
Relaxation = N-1 = 4

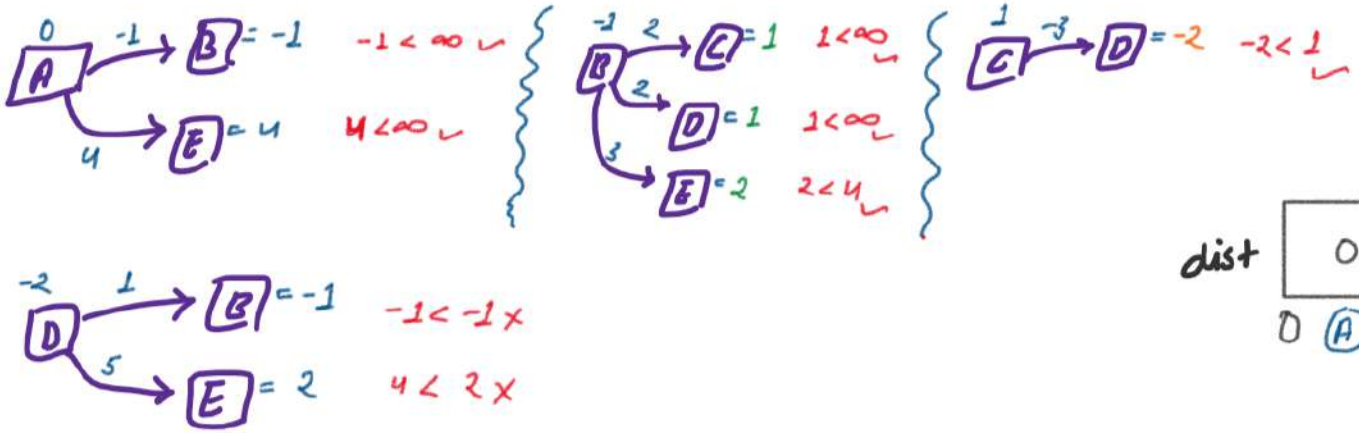
Relaxation 1:



Step 1: to find the SSSP

Key	value
A	{{B, -1}, {E, 4}}
B	{{C, 2}, {D, 2}, {E, 3}}
C	{{D, -3}}
D	{{E, 5}, {B, 1}}
E	×

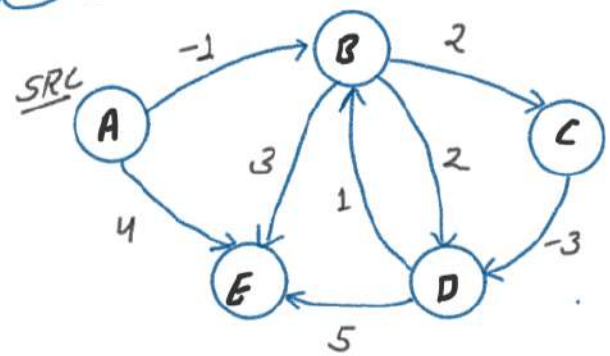
Adj list



dist	0	-1 ~~∞~~	~~∞~~ 1	~~∞~~ ~~1~~ -2	~~4~~ 2 ~~∞~~
	0 A	1 B	2 C	3 D	4 E

Relaxation 2:

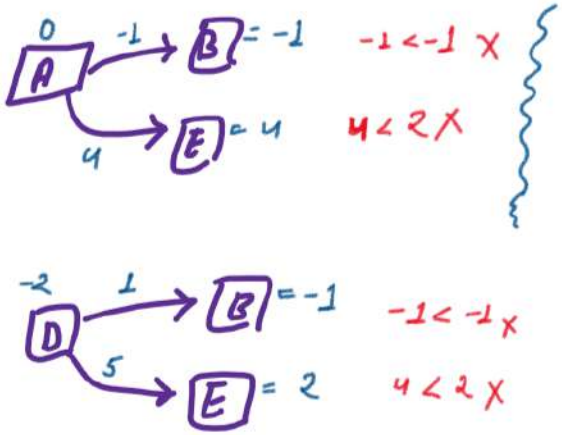
N = Total Node = 5
Relaxation = N-1 = 4

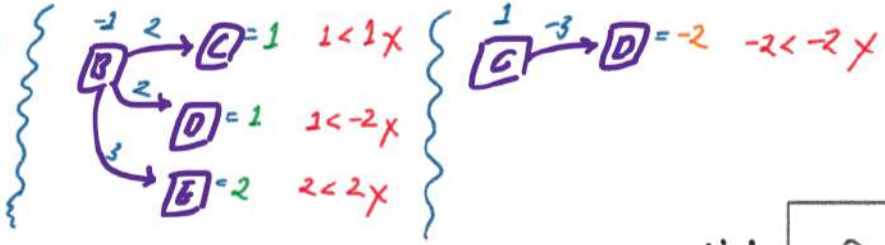


No updation while Relaxation 2

Key	value
A	{{B, -1}, {E, 4}}
B	{{C, 2}, {D, 2}, {E, 3}}
C	{{D, -3}}
D	{{E, 5}, {B, 1}}
E	×

Adj List

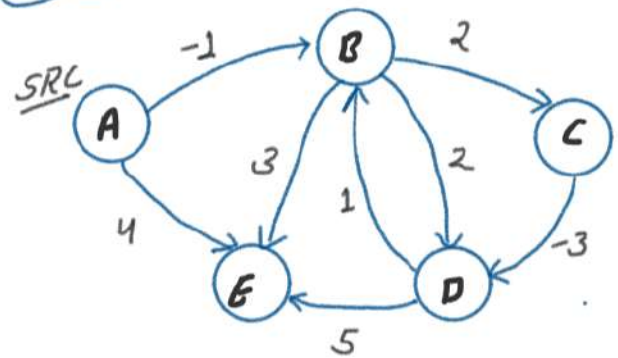




dist	0	-1	1	-2	2
	0 A	1 B	2 C	3 D	4 E

Relaxation 3:

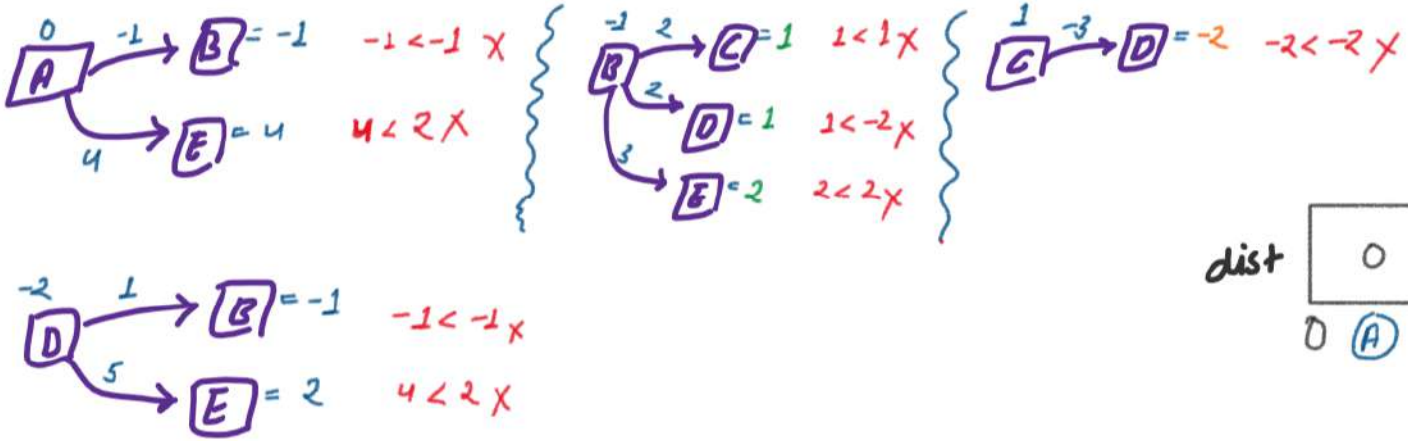
N = Total Node = 5
Relaxation = N-1 = 4



No updation while Relaxation 3

Key	value
A	{{B, -1}, {E, 4}}
B	{{C, 2}, {D, 2}, {E, 3}}
C	{{D, -3}}
D	{{E, 5}, {B, 1}}
E	×

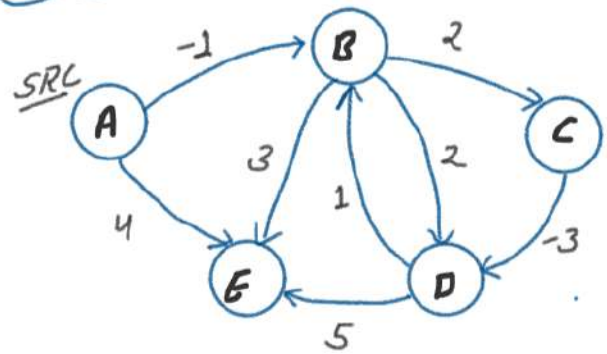
Adj list



dist	0	-1	1	-2	2
	0 A	1 B	2 C	3 D	4 E

N = Total Nodes = 5
Relaxation = N-1 = 4

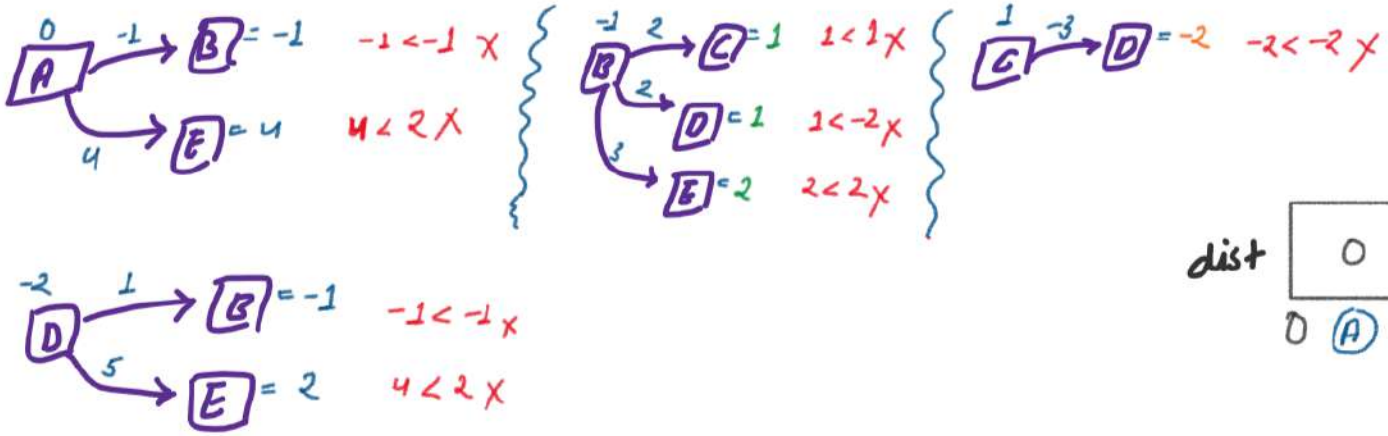
Relaxation 4:



No updation until Relaxation 4

Key	value
A	{{B, -1}, {E, 4}}
B	{{C, 2}, {D, 2}, {E, 3}}
C	{{D, -3}}
D	{{E, 5}, {B, 1}}
E	x

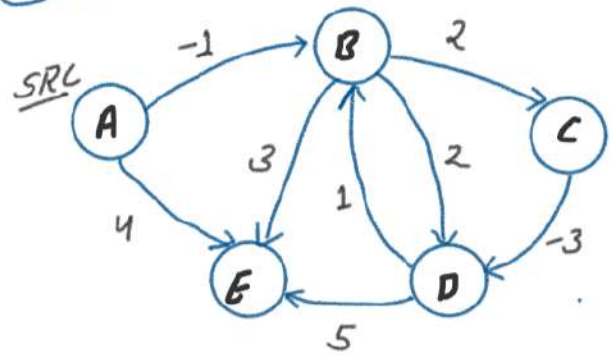
Adj List



dist	0	-1	1	-2	2
	0 A	1 B	2 C	3 D	4 E

Relaxation 5:

N = Total Node = 5
Relaxation = N-1 = 4



STOP The loop due to Relaxation is 5 - 0 = 4

Key	value
A	{{B, -1}, {E, 4}}
B	{{C, 2}, {D, 2}, {E, 3}}
C	{{D, -3}}
D	{{E, 5}, {B, 1}}
E	×

Adjlist

All shortest path from SRC

dist	0		-1		1		-2		2
	0	A	1	B	2	C	3	D	4	E

```cpp
#include<iostream>
#include<vector>
#include<unordered_map>
#include<limits.h>
#include<list>

using namespace std;

class Graph
{
    public:
        unordered_map<char, list<pair<char, int>>> adjList;

        void addEdges(char u, char v, int wt, int direction){
            if(direction == 1){
                // Directed Graph
                adjList[u].push_back({v,wt});
            }
            else{
                // Undirected Graph
                adjList[u].push_back({v,wt});
                adjList[v].push_back({u,wt});
            }
        }

        void bellManFord(int n, char src) {...}
};

int main(){
    Graph g;
    g.addEdges('A','B',-1,1);
    g.addEdges('A','E',4,1);
    g.addEdges('B','C',2,1);
    g.addEdges('B','D',2,1);
    g.addEdges('B','E',3,1);
    g.addEdges('C','D',-3,1);
    g.addEdges('D','E',5,1);
    g.addEdges('D','B',1,1);
    g.printAdjList();

    int n = 5;
    char src = 'A';
    g.bellManFord(n, src);

    return 0;
}
```

```cpp
void bellManFord(int n, char src) {
    // Initial state
    vector<int> dist(n, INT_MAX);
    dist[src-'A'] = 0;

    // Step 1: Relaxation N-1 times to get the all shortest path from single node.
    for(int i=1; i<n; i++){
        // Traverse on entire edge list
        for(auto a: adjList){
            for(auto b: a.second){
                char u = a.first;
                char v = b.first;
                int wt = b.second;
                if(dist[u-'A'] != INT_MAX && dist[u-'A'] + wt < dist[v-'A']){
                    dist[v-'A'] = dist[u-'A'] + wt;
                }
            }
        }
    }

    // Step 2: One more relaxation after N-1 times to detect the negative cycle
    // 1 time relaxation
    bool anyUpdate = false;
    for(auto a: adjList){
        for(auto b: a.second){
            char u = a.first;
            char v = b.first;
            int wt = b.second;
            if(dist[u-'A'] != INT_MAX && dist[u-'A'] + wt < dist[v-'A']){
                anyUpdate = true;
                break;
                dist[v-'A'] = dist[u-'A'] + wt;
            }
        }
    }

    // Expected Output
    if(anyUpdate == true){
        cout << "Negative Cycle Present" << endl;
    }
    else{
        cout << "No Negative Cycle Present in Graph" << endl;
        cout << "Printing Distance Array: ";
        for(auto i: dist){
            cout << i << ", ";
        }
        cout << endl;
    }
}
```

T.C. = ?

S.C. = ?