Lab Report-3

Bellman Ford: **Bellman–Ford** is a shortest path algorithm. It is used to find the **minimum cost/distance** from one starting node to all other nodes in a **weighted graph**.

Works on:

- Weighted graph
- Positive weight
- Negative weight

Does NOT work if:

- There is a **negative cycle** (distance keeps decreasing forever)

Negative weight: If any edge has a **negative value**, like `-2`, `-5`, `-10`,that is called a **negative weight**.

Example: A → B = -3 (this is a negative weight)

Negative cycle: A **negative cycle** is a cycle (loop) where the **total sum of weights is negative**.

Meaning:
If you keep going around the cycle, the cost becomes **smaller and smaller forever**.

Example: A → B = 2, B → C = -4 & C → A = -3

Total weight of the cycle =`2 + (-4) + (-3) = -5`

This is **negative** → so this is a **negative cycle**.

Why is this a problem?

Because if we can go around:
A → B → C → A → B → C → A → …
and every time our distance becomes **more negative**.

So **Bellman–Ford cannot give a final shortest path**, because the distance keeps decreasing forever.

Graph Search Example:

In Bellman–Ford, we relax all edges **(n − 1) times**, where **n is the number of vertices**. However, if at any point—**before completing all (n − 1) iterations**—we see that **no value changes between two consecutive iterations**, then we can stop early.

Because if an iteration makes **no updates**, it means all shortest distances are already found, and the **remaining iterations will not change anything**.

Relaxation Rule: It's the mechanism that checks if a shorter path to a neighbor has been found.

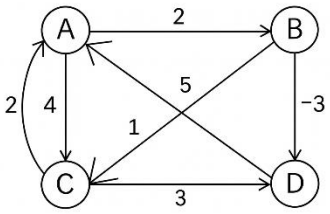
Formula: if(d[u]+c(u,v)<d[v]) ⟹ d[v]=d[u]+c(u,v)

Meaning of Each Symbol

- **u** : The *current node* (the node we are exploring now).
- **v** : A *neighbor node* of uuu.
- **d[u]** : The shortest distance from the start node to u (known so far).
- **d[v]** : The shortest distance from the start node to v (recorded so far).

c(u,v) : The cost/weight of the edge from u to v.

Graph:



Find the path from A to D, where A is the source Node.

Iteration-1:

Step-1: Initialzation

	A	**B**	**C**	**D**
A	0	∞	∞	∞

Node	A	B	C	D
Parent	-1	-1	-1	-1

Step-2:

	A	**B**	**C**	**D**
A	0	∞	∞	∞
B	0	2	4	∞

Node	A	B	C	D
Parent	-1	A	A	-1

Check Condition for B:

$d[u] + c(u,v) < d[v] \Rightarrow 0 + 2 < \infty$
Condition is **true**, so we **relax**:
$d[v] = d[u] + c(u,v) = 0+2 = 2$

Check Condition for C:

$d[u] + c(u,v) < d[v] \Rightarrow 0 + 4 < \infty$
Condition is **true**, so we **relax**:
$d[v] = d[u] + c(u,v) = 0+4 = 4$

Step-3:

	A	**B**	**C**	**D**
A	0	∞	∞	∞
B	0	2	4	∞
C	0	2	3	-1

Node	A	B	C	D
Parent	-1	A	B	B

Check Condition for C:

$d[u] + c(u,v) < d[v] \Rightarrow 2+1 < 4$
Condition is **true**, so we **relax**:
$d[v] = d[u] + c(u,v) = 2+1 = 3$

Check Condition for D:

$d[u] + c(u,v) < d[v] \Rightarrow 2-3 < \infty$
Condition is **true**, so we **relax**:
$d[v] = d[u] + c(u,v) = 2-3 = -1$

Step-4:

	A	**B**	**C**	**D**
A	0	∞	∞	∞
B	0	2	4	∞
C	0	2	3	-1
D	0	2	3	-1

Node	A	B	C	D
Parent	-1	A	B	B

Step-5:

	A	**B**	**C**	**D**
A	0	∞	∞	∞
B	0	2	4	∞
C	0	2	3	-1
D	0	2	3	-1
	0	2	3	-1

Node	A	B	C	D
Parent	-1	A	B	B

Iteration-2:

Step-1:

	A	**B**	**C**	**D**
A	0	2	3	-1

Node	A	B	C	D
Parent	-1	A	B	B

Step-2:

	A	**B**	**C**	**D**
A	0	2	3	-1
B	0	2	3	-1

Node	A	B	C	D
Parent	-1	A	B	B

Step-3:

	A	**B**	**C**	**D**
A	0	2	3	-1
B	0	2	3	-1
C	0	2	3	-1

Node	A	B	C	D
Parent	-1	A	B	B

Step-4:

	A	**B**	**C**	**D**
A	0	∞	∞	∞
B	0	2	4	∞
C	0	2	3	-1
D	0	2	3	-1

Node	A	B	C	D
Parent	-1	A	B	B

Step-5:

	A	**B**	**C**	**D**
A	0	∞	∞	∞
B	0	2	4	∞
C	0	2	3	-1
D	0	2	3	-1
	0	2	3	-1

Node	A	B	C	D
Parent	-1	A	B	B

- **Since the value of iteration remains same so the the shortest path from A to D Node:**

  $A \rightarrow B \rightarrow D$

Pseudocode:

```
BELLMAN-FORD(G, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    // Computation Phase: Relax all edges |V| - 1 times
    for i ← 1 to |V| - 1 do
        for each edge (u, v) ∈ G.E do
            RELAX(u, v)
    // Check Phase: Detect negative-weight cycles
    for each edge (u, v) ∈ G.E do
        if d[v] > d[u] + w(u, v) then
            return FALSE  // Negative cycle detected
    return TRUE  // No negative cycle; shortest paths found
```