Design and Analysis of Algorithms

Dr. Roshan Fernandes
Associate Professor
Dept. of CS&E
NMAMIT, Nitte

Single-Source Shortest Paths

-1-

Graph Algorithms

Roshan Fernandes
NMAMIT, NITTE

In a shortest-paths problem, we are given a weighted, directed graph $G = (V, E)$, with weight function $w: E \rightarrow R$ mapping edges to real-valued weights.

The weight of path $p = \langle v_0, v_1, \ldots v_k \rangle$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

We define the shortest-path weight from u to v by:

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

A shortest-path from vertex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$.

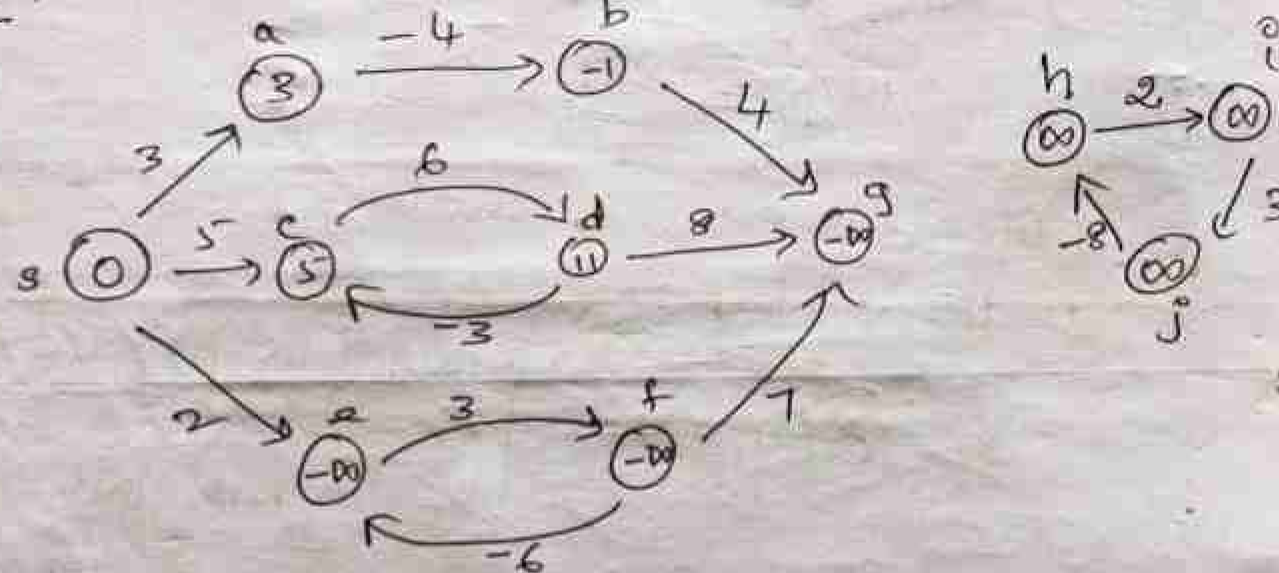
Negative-weight edges:

In some instances of the single-source shortest paths problem, there may be edges whose weights are negative. If the graph $G = (V, E)$ contains no negative weight cycles reachable from the source s, then for all $v \in V$, the shortest path weight $\delta(s, v)$ remains well defined, even if it has a negative value.

If there is a negative weight cycle reachable from s, however, shortest path weights are not well defined.

If there is a negative-weight cycle on some path from s to v, we define $\delta(s, v) = -\infty$.

Example:



- Because there is only one path from s to a (the path: ⟨s,a⟩), $\delta(s,a) = w(s,a) = 3$.
- There is only one path from s to b, and so $\delta(s,b) = w(s,a) + w(a,b) = 3+(-4) = -1$.
- There are infinitely many paths from s to c: ⟨s,c⟩, ⟨s,c,d,c⟩, ⟨s,c,d,c,d,c⟩, and so on. ~~Because~~ If we take path ⟨s,c⟩ it is 5; If we take path ⟨s,c,d,c⟩ it will be $5+(6-3) = 8$. $(5<8)$ so $\delta(s,c) = w(s,c) = 5$.

- There are infinitely many paths from s to e: $\langle s,e \rangle$, $\langle s,e,f,e \rangle$, $\langle s,e,f,e,f,e \rangle$ and so on. Since the cycle $\langle e,f,e \rangle$ has weight $3+(-6) = -3 < 0$, however, there is no shortest path from s to e.

By traversing the negative-weight cycle $\langle e,f,e \rangle$ arbitrarily many times, we can find paths from s to e with arbitrarily large negative weights, and so $\delta(s,e) = -\infty$.

- Similarly, $\delta(s,f) = -\infty$
- Similarly, $\delta(s,g) = -\infty$.
- Vertices h, i, and j also form a negative-weight cycle. They are not reachable from s, however, and so $\delta(s,h) = \delta(s,i) = \delta(s,j) = \infty$.

Note: 1. Some shortest-paths algorithms, such as Dijkstra's algorithm, assume that all edge weights in the input graph are non negative.

2. Others, like Bellman-Ford algorithm, allow negative-weight edges in the input graph and produce a correct answer as long as no negative-weight cycles are reachable from the source. If there is such a negative-weight cycle, the algorithm can detect and report its existence.

Note: Can a shortest path contain a cycle?

It cannot contain a negative-weight cycle nor it can contain a positive-weight cycle.

Representing shortest paths:

We often wish to compute not only shortest-path weights, but the vertices on shortest paths as well.

Given a graph $G = (V, E)$, we maintain for each vertex $v \in V$ a predecessor $\pi[v]$ that is either another vertex or NIL.

Relaxation :-

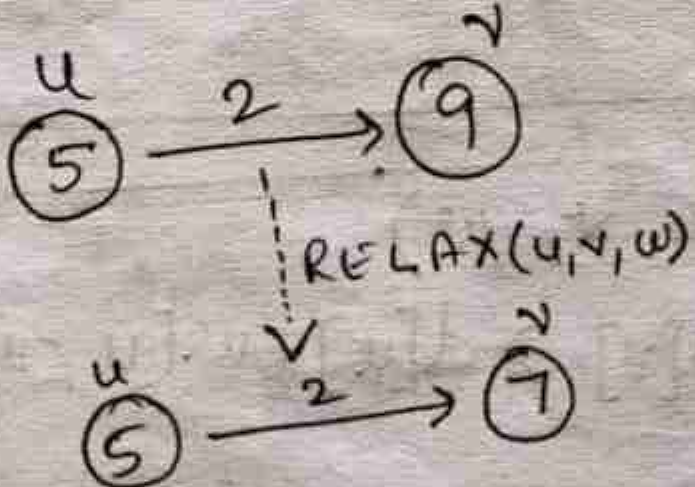
For each vertex $v \in V$, we maintain an attribute $d[v]$, which is an upper bound on the weight of a shortest path from source s to v.

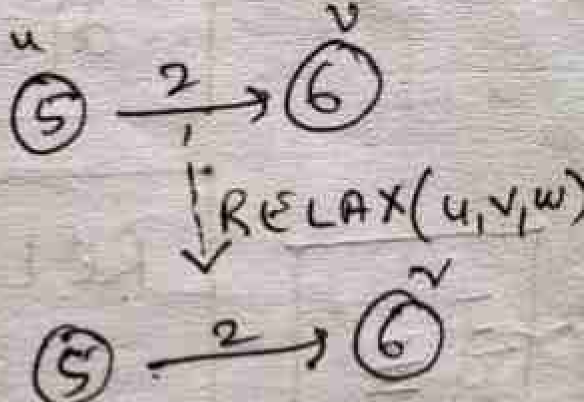
We call $d[v]$ a <u>shortest-path estimate</u>

The process of relaxing an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating $d[v]$ and $\pi[v]$.

A relaxation step may decrease the value of the shortest-path estimate d[v] and update v's predecessor field π[v].

Example:





The Bellman-Ford Algorithm:

- The Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which edge weights may be negative.
- Given a weighted, directed graph $G=(V,E)$ with source s and weight function $w: E \to R$, the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a -ve weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths & their weights.

The complete Algorithm is given in next sheet.

INITIALIZE-SINGLE-SOURCE (G, s)

for each vertex $v \in V[G]$

do $d[v] \leftarrow \infty$

$\pi[v] \leftarrow NIL$

$d[s] \leftarrow 0.$

RELAX (u, v, w)

if $d[v] > d[u] + w(u, v)$

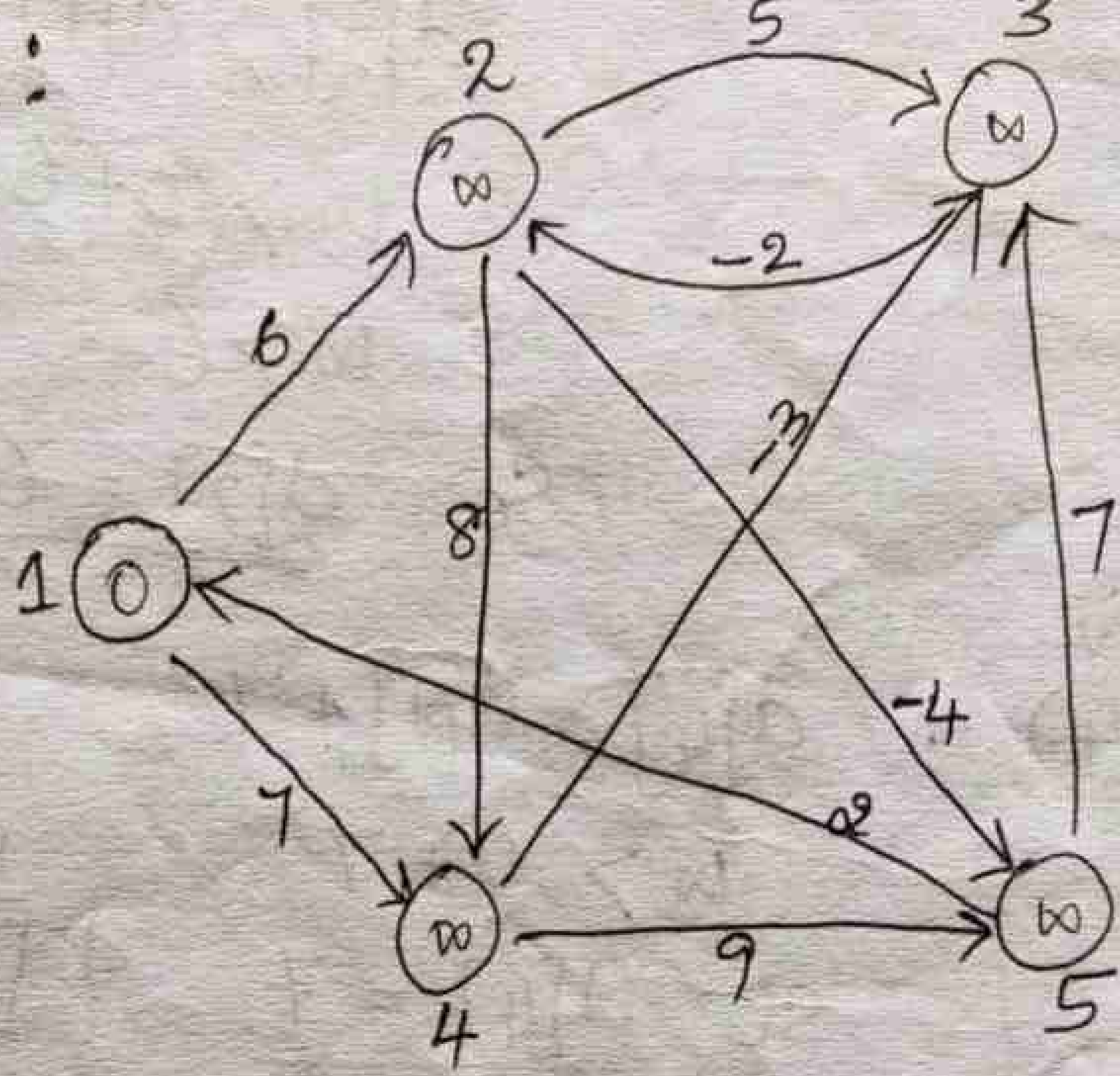
then $d[v] \leftarrow d[u] + w(u, v)$

$\pi[v] \leftarrow u$

```
PRINT-PATH (G, s, v)
if v = s
    then print s
    else if π[v] = NIL
        then print "no path from" s "to" v
                                   "exists"
    else
        PRINT-PATH(G, s, π[v])
            print v.
```

```
BELLMAN-FORD (G, w, s)
INITIALIZE-SINGLE_SOURCE (G, s)
for i ← 1 TO |V[G]| - 1
  do
    for each edge (u, v) ∈ E[G]
    do
        RELAX (u, v, w)
for each edge (u, v) ∈ E[G]
do
    if d[v] > d[u] + w(u, v)
    then
        return FALSE   // there is -ve weight cycle
return TRUE.  // No -ve weight cycle.
```

Example 1:



The data structure used to represent info about each edge:

```
struct Edge
{ int u;
  int v;
  int w;
};
Edge edges[50];
```

Initially:

d:

1	2	3	4	5
0	∞	∞	∞	∞

[since vertex 1 is source].

edges:

	1	2	3	4	5	6	7	8	9	10
u	1	1	2	2	2	3	4	4	5	5
v	2	4	3	4	5	2	3	5	1	3
w	6	7	5	8	-4	-2	-3	9	2	7

ITERATION 1: edge $\overset{u,v}{(1,2)}$: $d[v] > d[u] + w[u,v]$

$d[2] > d[1] + 6$

$\infty > 0 + 6$

$\infty > 6$

So $d[2] = 6$.

d:

1	2	3	4	5
0	6	∞	∞	∞

edge $\overset{u,v}{(1,4)}$: $d[4] > d[1] + 7$

$\infty > 0 + 7$

So $d[4] = 7$.

d:

1	2	3	4	5
0	6	∞	7	∞

edge $\overset{u,v}{(2,3)}$: $\infty > 6 + 5$ So $d[3] = 11$.

d:

1	2	3	4	5
0	6	11	7	∞

edge (2,4) : $7 > 6+8$

So no change

d:

1	2	3	4	5
0	6	11	7	∞

edge (2,5) : $\infty > 6+(-4)$

So $d[5] = 2$

d:

1	2	3	4	5
0	6	11	7	2

edge (3,2) : $6 > 11+(-2)$

So no change.

d:

1	2	3	4	5
0	6	11	7	2

edge (4,3) : $11 > 7+(-3)$

So $d[3] = 4$

d:

1	2	3	4	5
0	6	4	7	2

edge (4,5) : $2 > 7+9$

So no change.

d:

1	2	3	4	5
0	6	4	7	2

edge (5,1) : $0 > 2+2$

So no change.

d:

1	2	3	4	5
0	6	4	7	2

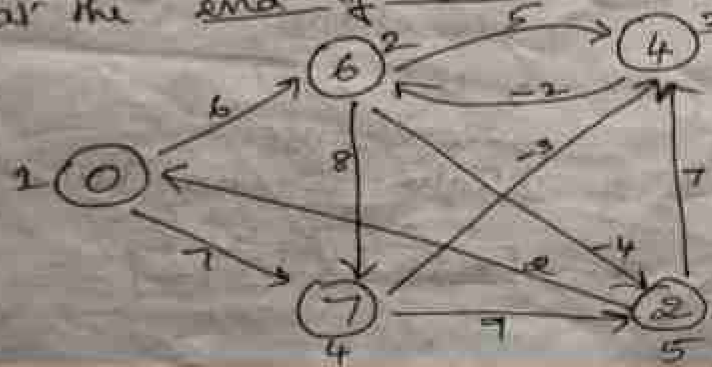
edge (5,3) : $4 > 2+7$

So no change.

d:

1	2	3	4	5
0	6	4	7	2

So at the end of iteration 1 we have:



Iteration 2; Now d:

1	2	3	4	5
0	6	4	7	2

u v d[v] d[u] + w(u, v)

edge (1, 2) : 6 > 0 + 6
No change.

edge (1, 4) : 7 > 0 + 7
No change.

edge (2, 3) : 4 > 6 + 5
No change.

edge (2, 4) : 7 > 6 + 8
No change.

edge (2, 5) : 2 > 6 − 4
No change.

edge (3, 2) : 6 > 4 − 2
So d[2] = 2.

d:

1	2	3	4	5
0	2	4	7	2

edge (4, 3) : 4 > 7 − 3
No change.

edge (4, 5) : 2 > 7 + 9
No change.

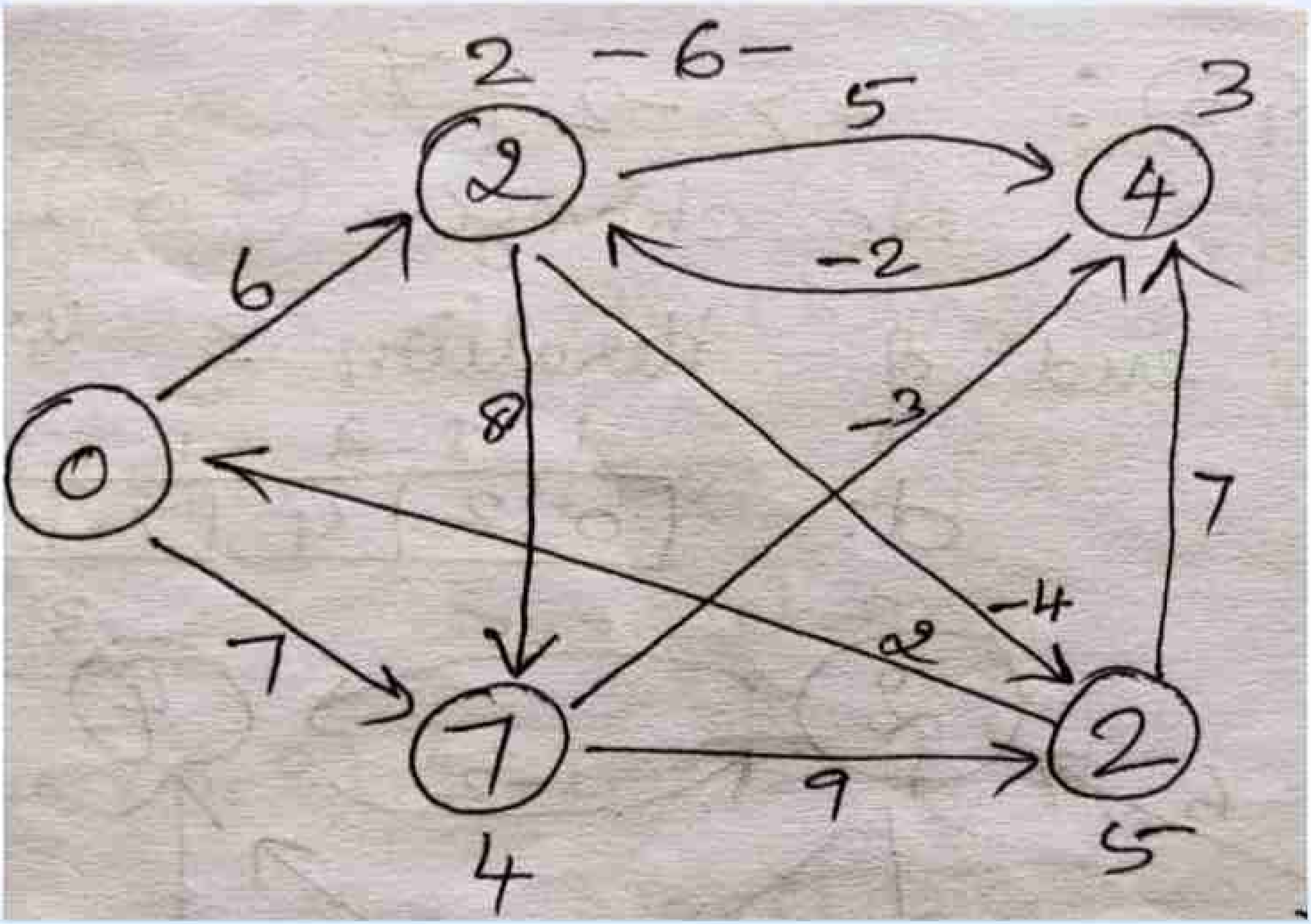
edge (5, 1) : 0 > 2 + 2
No change.

edge (5, 3) : 4 > 2 + 7
No change.

At the end of iteration 2 we have

d:

1	2	3	4	5
0	2	4	7	2


2
-6-
5
3
2
4
-2
6
8
-3
0
7
-4
2
7
7
2
9
4
5

Iteration 3:

d:

1	2	3	4	5
0	2	4	7	2

u, v

edge (1,2) : $2 > 0+6$
No change.

edge (1,4) : $7 > 0+7$
No change.

edge (2,3) : $4 > 2+5$
No change.

edge (2,4) : $7 > 2+8$
No change.

edge (2,5) : $2 > 2-4$
$d[5] = -2$.

d:

1	2	3	4	5
0	2	4	7	-2

edge (3,2) : $2 > 4-2$
No change.

edge (4,3) : $4 > 7-3$
No change.

edge (4,5) : $-2 > 7+9$
No change.

edge (5,1) : $0 > -2+2$
No change.

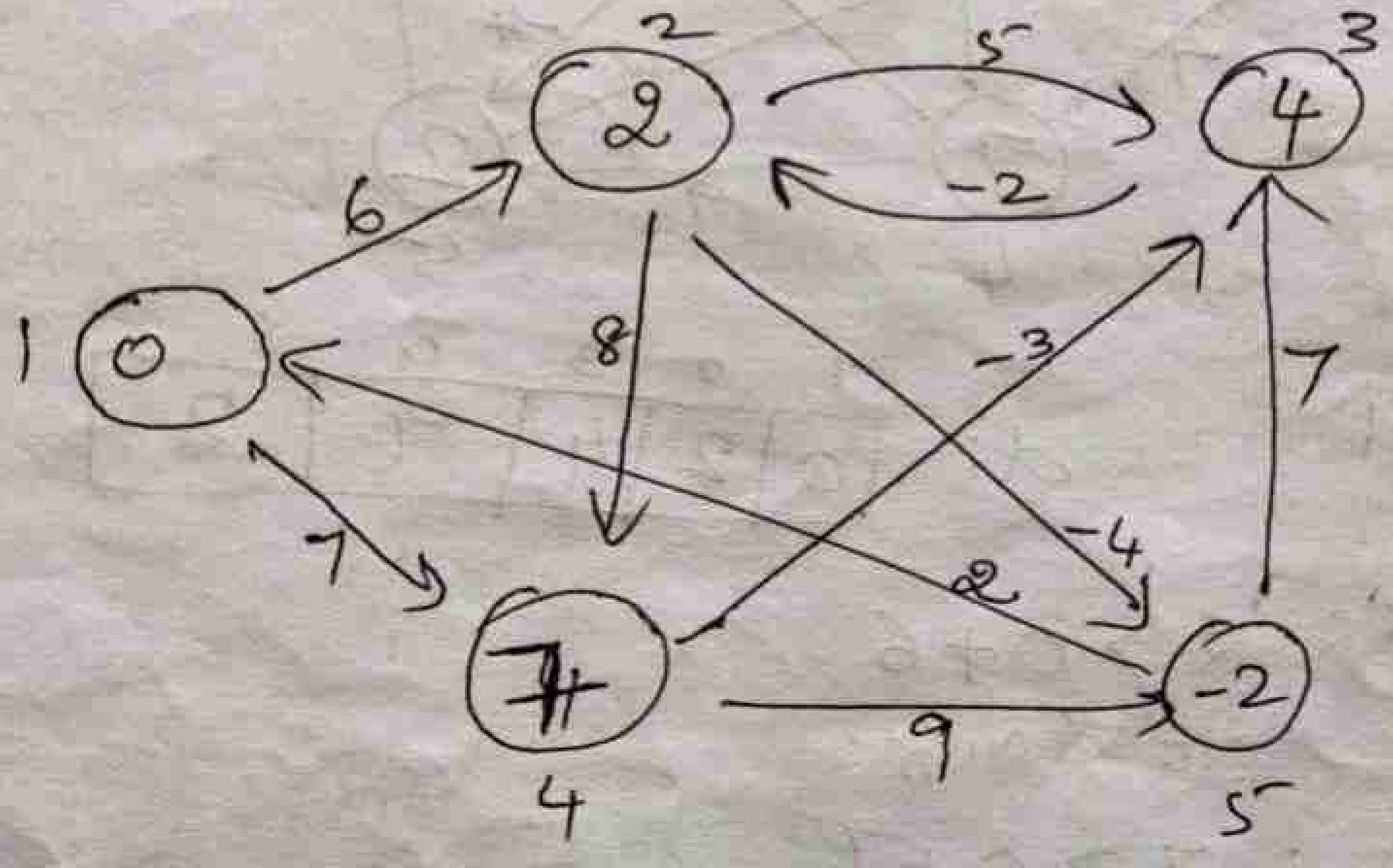
edge (5,3): $4 > -2 + 7$

No change.

At the end of iteration 3 we have:

d:

1	2	3	4	5
0	2	4	7	-2



Iteration 4:

$$d: \begin{array}{|c|c|c|c|c|} \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} & \multicolumn{1}{c}{5} \\ \hline 0 & 2 & 4 & 7 & -2 \\ \hline \end{array}$$

edge $(1,2)$: $2 > 0+6$
No change.

edge $(1,4)$: $7 > 0+7$
No change.

edge $(2,3)$: $4 > 2+5$
No change.

edge $(2,4)$: $7 > 2+8$
No change.

edge $(2,5)$: $-2 > 2-4$
No change.

edge $(3,2)$: $2 > 4-2$
No change.

edge $(4,3)$: $4 > 7-3$
No change.

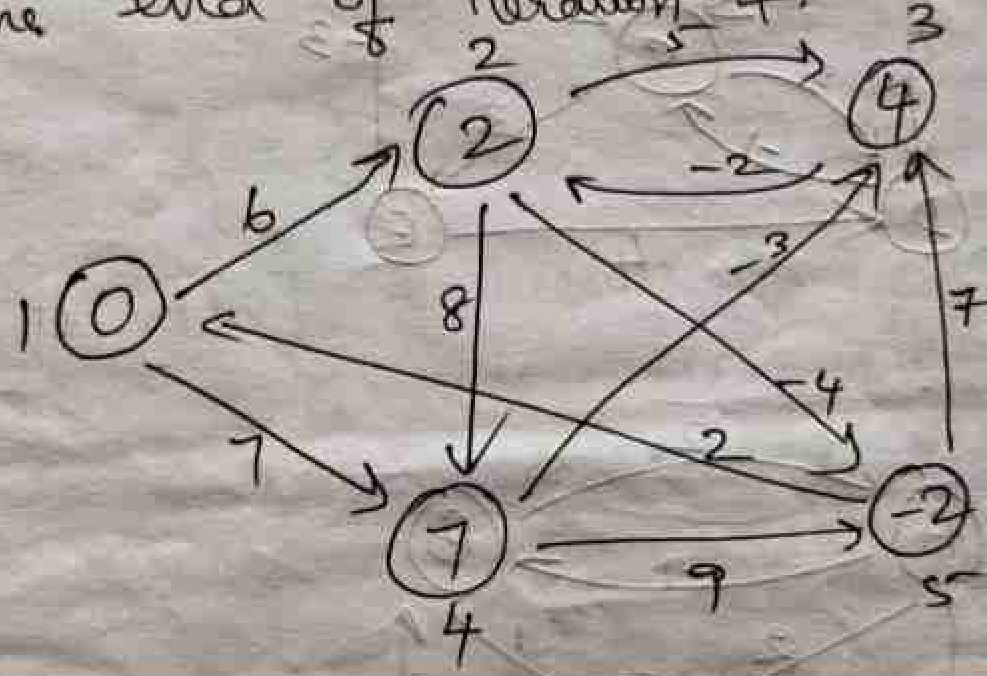
Roshan Fernandes
NMAMIT, Nitte

edge (4,5) : $-2 > 3+9$
No change.

edge (5,1) : $0 > -2+2$
No change.

edge (5,3) : $4 > -2+7$
No change.

At the end of iteration 4:



final Answer.

The Bellman-Ford algorithm runs in $O(V \cdot E)$;

~~si~~ What is the difference between Dijkstra and Bellman-Ford?

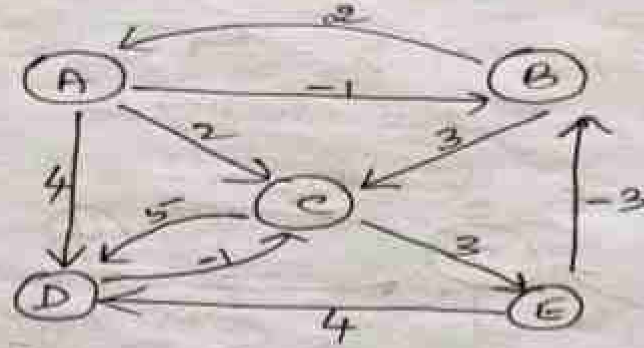
Basically the Dijkstra's method marks each node as 'visited' once it relaxes all edges from that node, and will not try to relax any edges leading back to that node after that point.

Hence Dijkstra's algorithm completes much quicker than Bellman-Ford's and scales better with a larger no. of nodes.

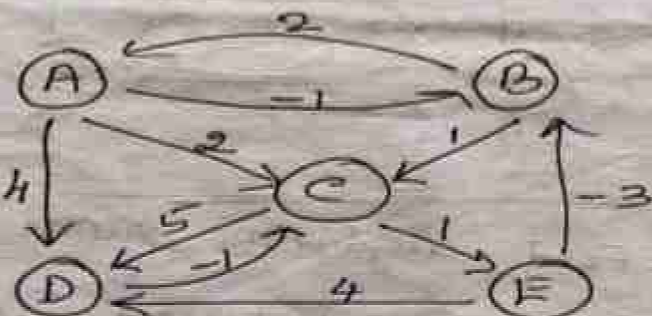
The main drawback of Dijkstra's method is that it will not work with negative edge weights.

Assignments :

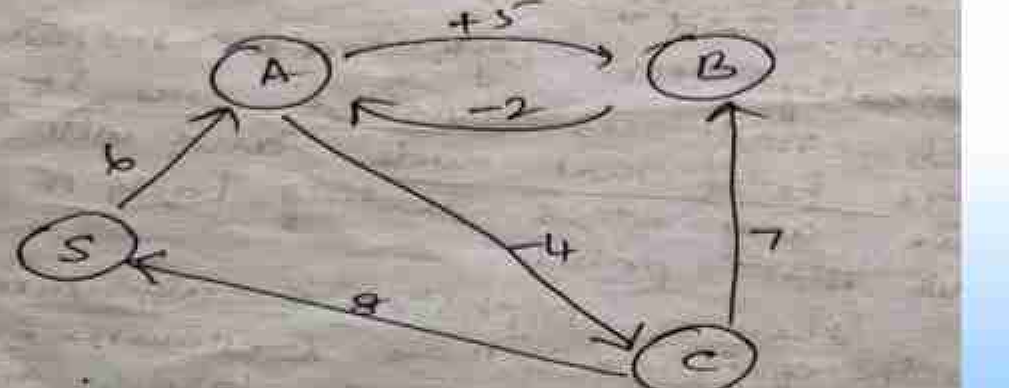
1)



2]



3]



Thank You!!!