So we have found our SCC1, SCC2, SCC3.

— × — × — × — × ——

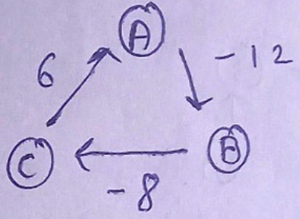
Lecture 101

Bellman Ford algorithm

It is used to find shortest path b/w source to destination but graph may contain negative weights.

Dijkstra's algo also finds shortest path but it does not work in negative weights. because it is a kind of Greedy appraoch.

Bellman ford does not work for -ve cycles. means



Let say we start from C

C→A→B = 6+(-12) = -6

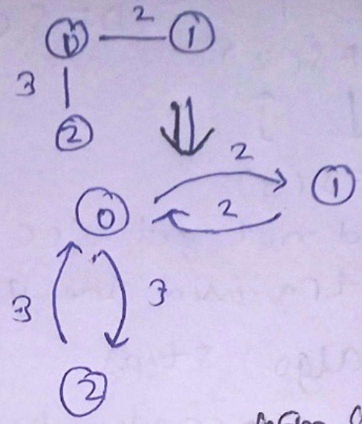
B→C = -6-8 = -14

C→A = -14+6 = -8

A→B = -8-12 = -20

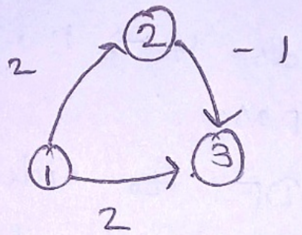
everytime we get new shortest path and we are stuck in a cycle, this is called -ve cycle but using Bellman ford algo we can find whether -ve cycle present in graph or not

We can apply Bellman ford in Directed graph. but we can convert unDirected graph to directed



this way we can apply Bellman ford in undirected graph.

$\boxed{\text{Algo}}$



1) Algo says we need to apply below formula (n-1) times in all edges.

$u \xrightarrow{wt} v$ is an edge

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

we apply it (n-1) times

$1 \to 2$ (wt = 2)

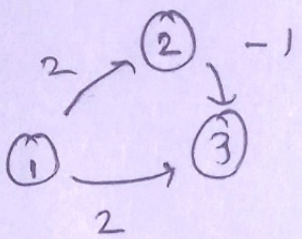
$2 \to 3$ (wt = -1)

$1 \to 3$ (wt = 2)

(2) one more time apply same dist-formula and if any once gets updated means -ve cycle is present, we can't ~~apply~~ find shortest path

(3) Otherwise return the shortest distance

DRY RUN



edges

$1 \to 2$ (2)

$2 \to 3$ (-1)

$1 \to 3$ (2)

n = 3 so we apply formula 2 times

we make a Distance array

Dist	0	~~∞~~ 3	∞
	1	2	3

src node = 1

so dist [1] = 0, rest all ∞

1st time formula

we can start applying from any edge

a) dist [1] + 2 < dist [2]

0 + 2 < ∞ // True

{ Dist [2] = 3

}

b) dist [2] + (-1) < dist [3]

3 - 1 < ∞ // True

{

dist [3] = 2

}

c) dist [1] + 2 < dist [3]

0 + 2 < 2 // false

{

// Do nothing

}

First time application of formula is done

— x — x — x — x —

Second time application of formula

a) dist [1] + 2 < dist [2]

0 + 2 < 3 // True

{ dist [2] = 3

}

b) dist [2] + (-1) < dist [3]

2 - 1 < 3 // True

{

dist [3] = 1

}

c) dist [1] + 2 < dist [3]

0 + 2 < 1 // false

{

// Do nothing

}

After 2 implementation

Dist array is

{ 0 , 2 , 1 }

1 2 3

↑ src = 1

considering

1 → 1 = short dist = 0

1 → 2 = shortest dist = 2

1 → 3 = " " = 1

2) Checking negative cycle

apply formula again

for all edges

a) dist [1] + wt < dist [2]

0 + 2 < 2 // false

b) dist [2] + (-1) < dist [3]

2 + (-1) < 1

1 < 1 // false

c) dist [1] + 2 < dist [3]

0 + 2 < 1

2 < 1 // false

No, -ve cycle present

3) return dist [1]