Module - 3

→ Analyze the amortized cost of the self-balancing operation (such as rotations and color flips) in red-black trees using the aggregate method.

A: Worst - Case Scenarios:

- Each insertion or deletion may propagate rebalancing up the height of the tree.
- A red-black tree with 'n' nodes has a height $O(\log n)$

Total Cost Over 'm' Operations:

- Color flips: Each node can only change colors a few times in a single operation (at most 2 per level).
- Rotations: A rotation occurs at most twice per level in a single operation.

For 'm' insertions or deletions:

- Each insertion or deletion involves $O(\log n)$ rebalancing operations.
- Thus, the total cost over 'm' operations is $O(m \log n)$

Amortized Cost Per Operation:

Using the aggregate method:

- Amortized cost = $\frac{\text{Total Cost}}{\text{Number of Operations}}$

∴ Amortized cost per operation = $\frac{O(m \log n)}{m} = O(\log n)$

2> Analyze the amortized cost of finding negative cycle in bellman ford algorithm.

A: Key Observations:

1. Negative Cycle Detection:

- Negative cycles are detected in the final iteration of edge relaxations.
- This requires O(E) work

2. Amortized over |V| - 1 iterations:

- The final iteration's cost O(E) can be distributed across all |V| - 1 iterations of the algorithm.
- Amortized cost per iteration for detecting negative cycles: $= \frac{O(E)}{|V|-1} = O\left(\frac{E}{V}\right)$.

Total Amortized Cost Per Edge Relaxation:

- Including the O(E · (|V| - 1)) cost for edge relaxations and the final negative cycle detection, the total amortized cost per relaxation step is:

$\therefore$ Amortized cost per relaxation $= \frac{O(E \cdot V)}{E \cdot (|V|-1)} = O(1)$.

3> What is the running time of bellman ford algorithm for a complete graph with 'n' vertices. In the bellman Ford algorithm, if a graph has 5 vertices

and 12 edges, how many iterations are required to guarantee finding the shortest paths from a single source to all other vertices?

A: • Total number of edges in a complete graph is:

$$E = \frac{n(n-1)}{2}$$

Steps in Bellman Ford algorithm.

1. Initialization: $O(V)$, where $V = n$.
2. Relaxation: Relax all edges $|V|-1$ times.
   • In each iteration, all E edges are processed.
   • Total cost of relaxation:

$$O((V-1)\cdot E) = O\left(n\cdot\frac{(n-1)n}{2}\right) = O(n^3)$$

   • Cost: $O(E) = O(n^2)$.

Thus, the total running time of the Bellman-Ford algorithm for a complete graph is:

$$T_{total} = O(V) + O(E\cdot V) + O(E) = O(n^3).$$

To guarantee finding the shortest paths from a single source to all other vertices, $|V|-1$ iterations of edge relaxations are required.

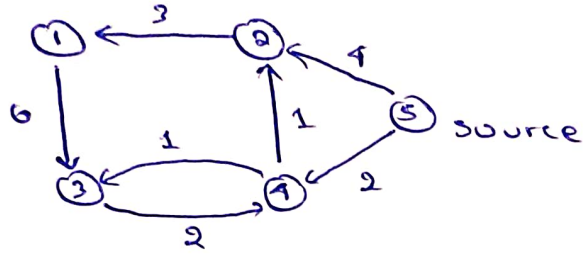
• For a graph with $V = 8$:

Number of iterations: $V-1 = 8-1 = 7$.

Thus, 7 iterations are required to guarantee finding the shortest paths.

Q> Consider the following directed weighted graph and apply the Bellman-Ford algorithm starting from vertex s to compute the shortest paths to all other vertices. Show the result after each iteration of relaxation. Also, detect if any negative-weight cycle exists.

A:

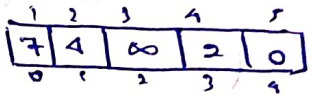
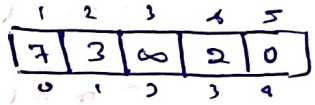
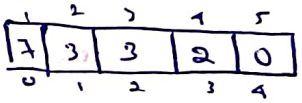


A: initialize distance vector (array) dist[5] = {∞, ∞, ∞, ∞, 0}
0 1 2 3 4

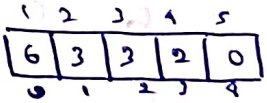
1st iteration:

edges (u,v,w)	dist(v) = min(dist[v], dist[u]+w)	updated dist array
(1,3,6)	= min(∞, ∞+6) = ∞	–
(2,1,3)	= min(∞, ∞+3) = ∞	–
(3,4,2)	= min(∞, ∞+2) = ∞	–
(4,2,1)	= min(∞, ∞+1) = ∞	–
(4,3,1)	= min(∞, ∞+1) = ∞	–
(5,2,4)	= min(∞, 0+4) = 4	1 2 3 4 5 / ∞ 4 ∞ ∞ 0 / 0 1 3 3 4
(5,4,2)	= min(∞, 0+2) = 2	1 2 3 4 5 / ∞ 4 ∞ 2 0 / 0 1 3 3 4

2nd iteration:

edges (u,v,w)	dist[v] = min(dist[v], dist[u]+w)	updated dist array
(1,3,6)	= min(∞, ∞+6) = ∞	–
(0,1,3)	= min(∞, 4+3) = 7	
(3,4,2)	= min(2, ∞+2) = 2	–
(4,2,1)	= min(4, 2+1) = 3	
(4,3,1)	= min(∞, 2+1) = 3	
(5,2,4)	= min(3, 4) = 3	–
(5,4,2)	= min(2, 2) = 2	–

3rd iteration:

edges (u,v,w)	dist[v] = min(dist[v], dist[u]+w)	updated dist array
(1,3,6)	= min(3, 7+6) = 3	–
(0,1,3)	= min(7, 3+3) = 6	
(3,4,2)	= min(2, 3+2) = 2	–
(4,2,1)	= min(3, 2+1) = 3	–
(4,3,1)	= min(3, 2+1) = 3	–
(5,2,4)	= min(3, 4) = 3	–
(5,4,2)	= min(2, 2) = 2	–

4th iterations:

edges (u,v,w)	dist[v] = min(dist[v], dist[u]+w)	Updated dist array
(1,3,6)	= min(3, 6+6) = 3	—
(2,1,3)	= min(6, 3+3) = 6	—
(3,4,2)	= min(2, 3+2) = 2	—
(4,2,1)	= min(3, 2+1) = 3	—
(4,3,1)	= min(3, 2+1) = 3	—
(5,2,4)	= min(3, 4) = 3	—
(5,4,2)	= min(2, 2) = 2	—

∴ After |v|-1 iterations we got dist[] as

1	2	3	4	5
6	3	2	2	0
0	1	2	3	4

5 → 1 = 6 (5 → 4 → 2 → 1)

5 → 2 = 3 (5 → 4 → 2)

5 → 3 = 3 (5 → 4 → 3)

5 → 4 = 2 (5 → 4)

5 → 5 = 0