BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
YELAHANKA, BENGALURU - 560064

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Report on Bellman Ford Algorithm

Name	Raghavendra K M
USN	1BY18IS093
Semester/Section	5B
Course Code	18CSL57
Course Name	Computer Network Laboratory
Faculty	Prof. Gireesh babu C N
Title	Bellman Ford Algorithm
Date	16-10-2020

Signature of a Student **Signature of a Faculty**

Aim of the experiment:

Write a program to find the shortest path between vertices using bellman-ford algorithm.

Bellman Ford algorithm:

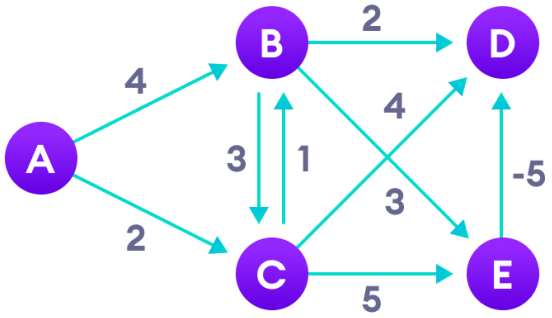
Bellman Ford algorithm helps us find the shortest path from a vertex to all other vertices of a weighted graph. It is similar to Dijkstra's algorithm but it can work with graphs in which edges can have negative weights.

How Bellman Ford's algorithm works

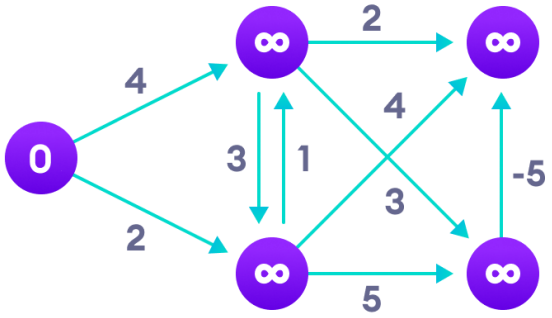
Bellman Ford algorithm works by overestimating the length of the path from the starting vertex to all other vertices. Then it iteratively relaxes those estimates by finding new paths that are shorter than the previously overestimated paths.

By doing this repeatedly for all vertices, we can guarantee that the result is optimized.

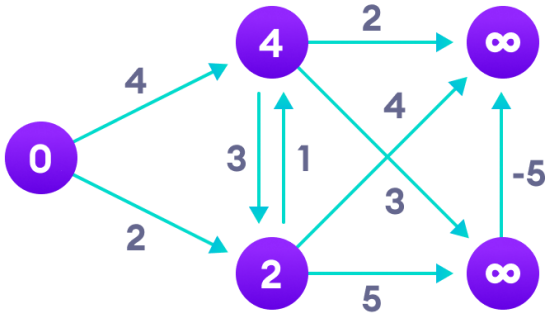
Step 1: Start with the weighted graph



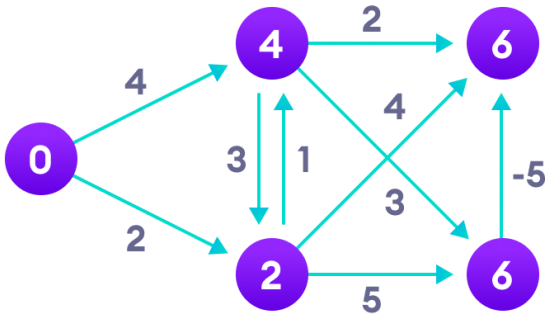
Step 2: Choose a starting vertex and assign infinity path values to all other vertices



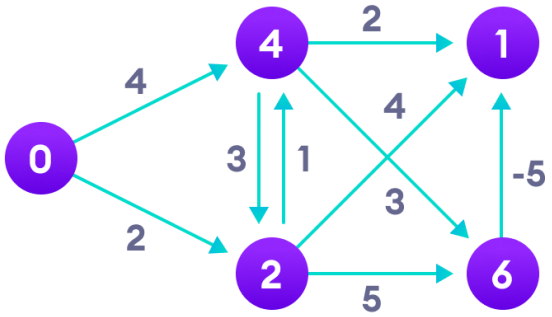
Step 3: Visit each edge and relax the path distances if they are inaccurate



Step 4: We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times



Step 5: Notice how the vertex at the top right corner had its path length adjusted



Step 6: After all the vertices have their path lengths, we check if a negative cycle is present

	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	6
0	3	2	1	6
0	3	2	1	6

Bellman Ford's Complexity

Time Complexity

Best Case Complexity	O(E)
Average Case Complexity	O(VE)
Worst Case Complexity	O(VE)

Space Complexity

And, the space complexity is `O(V)`.

Bellman Ford's Algorithm Applications

1. For calculating shortest paths in routing algorithms
2. For finding the shortest path

Source code:

BellmanFord.java

```
import java.util.Scanner;

public class BellmanFord {
    int[] d;
    int[][] a;
    int n, start;
    final int max = 999;

    public void read() {
        Scanner scan = new Scanner(System.in);

        System.out.println("Enter the number of vertices");
        n = scan.nextInt();
        d = new int[n + 1];
        a = new int[n + 1][n + 1];

        System.out.println("Enter the adjacency matrix: (Enter -- for Infinity)");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                String str = scan.next();
                if (str.equals("--"))
                    a[i][j] = max;
                else a[i][j] = Integer.parseInt(str);
            }
        }

        System.out.println("Enter the start vertex");
        start = scan.nextInt();

        if (start > n || start < 1) {
            System.out.println("Invalid start vertex.");
            System.exit(0);
        }
        scan.close();
    }

    public void calculate() {
        for (int i = 1; i <= n; i++)
            d[i] = max;
        d[start] = 0;

        for (int k = 1; k <= n - 1; k++)
            for (int i = 1; i <= n; i++)
                for (int j = 1; j <= n; j++)
                    if (a[i][j] != max)
                        if (d[j] > d[i] + a[i][j])
                            d[j] = d[i] + a[i][j];
    }

    public void display() {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (a[i][j] != max && d[j] > d[i] + a[i][j]) {
                    System.out.println("The Graph contains negative edge cycle");
```

```
                    return;
                }
            }
        }
        System.out.println("Distance from Source Vertex " + start + " to:");
        for (int i = 1; i <= n; i++)
            System.out.println("Vertex " + i + " is " + d[i]);
    }

    public static void main(String[] args) {
        BellmanFord obj = new BellmanFord();
        obj.read();
        obj.calculate();
        obj.display();
    }
}
```

Outputs:

Case 1:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\Bellman-ford Algorithm>javac BellmanFord.java

D:\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\Bellman-ford Algorithm>java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix: (Enter -- for Infinity)
-- 4 5 --
-- -- -- 10
-- 7 -- --
-- -- 3 --
Enter the start vertex
1
Distance from Source Vertex 1 to:
Vertex 1 is 0
Vertex 2 is 4
Vertex 3 is 5
Vertex 4 is 14

D:\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\Bellman-ford Algorithm>
```

Case 2:

```
C:\Windows\System32\cmd.exe

D:\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\Bellman-ford Algorithm>javac BellmanFord.java

D:\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\Bellman-ford Algorithm>java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix: (Enter -- for Infinity)
-- 5 4 --
-- -- -- 7
-- 7 -- --
-- -- -15 --
Enter the start vertex
1
The Graph contains negative edge cycle

D:\1by18is093\ISE V SEM\Subjects\18CSL57 - Computer Network Laboratory\Assignment\Bellman-ford Algorithm>
```