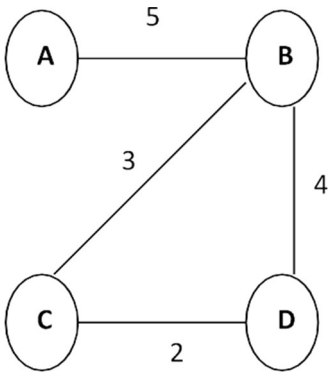
2.Write a program to find the shortest path between vertices using bellman-ford algorithm.

```
import java.util.Scanner;
public class BellmanFord {
   private int[] D;  // distance array
   private int num_ver;
   public static final int MAX_VALUE = 999;
   public BellmanFord(int num_ver) {
      this.num_ver = num_ver;
      D = new int[num_ver + 1]; // 1-based indexing
   }

   public void BellmanFordEvaluation(int source, int[][] A) {
      // Step 1: Initialize distances
      for (int node = 1; node <= num_ver; node++) {
         D[node] = MAX_VALUE;
      }
      D[source] = 0;

      // Step 2: Relax edges repeatedly (num_ver - 1 times)
      for (int k = 1; k <= num_ver - 1; k++) {
         for (int sn = 1; sn <= num_ver; sn++) {
            for (int dn = 1; dn <= num_ver; dn++) {
               if (A[sn][dn] != MAX_VALUE) {
                  if (D[sn] != MAX_VALUE && D[dn] > D[sn] + A[sn][dn]) {
                     D[dn] = D[sn] + A[sn][dn];
                  }
               }
            }
         }
      }

      // Step 3: Check for negative weight cycles
      for (int sn = 1; sn <= num_ver; sn++) {
         for (int dn = 1; dn <= num_ver; dn++) {
            if (A[sn][dn] != MAX_VALUE) {
               if (D[sn] != MAX_VALUE && D[dn] > D[sn] + A[sn][dn]) {
                  System.out.println("The Graph contains a negative edge cycle!");
                  return;
               }
            }
         }
```

```
        }

        // Step 4: Print result
        for (int vertex = 1; vertex <= num_ver; vertex++) {
            System.out.println("Distance of source " + source + " to " + vertex + " is "
+ D[vertex]);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of vertices:");
        int num_ver = scanner.nextInt();

        int[][] A = new int[num_ver + 1][num_ver + 1];

        System.out.println("Enter the adjacency matrix:");
        for (int sn = 1; sn <= num_ver; sn++) {
            for (int dn = 1; dn <= num_ver; dn++) {
                A[sn][dn] = scanner.nextInt();

                if (sn == dn) {
                    A[sn][dn] = 0; // Distance to self is 0
                } else if (A[sn][dn] == 0) {
                    A[sn][dn] = MAX_VALUE; // No edge => Infinity
                }
            }
        }

        System.out.println("Enter the source vertex:");
        int source = scanner.nextInt();

        BellmanFord b = new BellmanFord(num_ver);
        b.BellmanFordEvaluation(source, A);

        scanner.close();
    }
}
```

Output:

Input Graph



Enter the number of vertices:

4

Enter the adjacency matrix:

0 5 0 0

5 0 3 4

0 3 0 2

0 4 2 0

Enter the source vertex:

2

Distance of source 2 to 1 is 5

Distance of source 2 to 2 is 0

Distance of source 2 to 3 is 3

Distance of source 2 to 4 is 4

Case 2:

Enter the number of vertices:

6

Enter the adjacency matrix:

0 6 4 5 0 0

0 0 0 0 -1 0

0 -2 0 0 3 0

0 0 -2 0 0 -1

0 0 0 0 0 3

0 0 0 0 0 0

Enter the source vertex:

1

Distance of source 1 to 1 is 0

Distance of source 1 to 2 is 1

Distance of source 1 to 3 is 3

Distance of source 1 to 4 is 5

Distance of source 1 to 5 is 0

Distance of source 1 to 6 is 3