Distance Vector Algorithm to find suitable path for transmission

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    def add_edge(self, s, d, w):
        self.graph.append([s, d, w])
    def print_solution(self, dist, src):
        print("Vertex Distance from", ~~source~~ src)
        for i in range(self.V):
            print("{0}\t\t{1}".format
                        (i, dist[i]))
    def bellman_ford(self, src):
        dist = [float("Inf")] * self.V
        dist[src] = 0
        for _ in range(self.V - 1):
            for s, d, w in self.graph:
                if dist[s] != float("Inf")
                    and dist[s] + w < dist[d]
                        dist[d] = dist[s] + w

        for s, d, w in self.graph:
            if dist[s] != float("Inf") and
                dist[s] + w < dist[d]:
                print("Graph contains negative
                       weight cycle")
                    return
self.print_solution (dist, src )
def main ( ):
      matrix = [ ]
      print ("Enter the no. of nodes")
      n = int (input ( ) )
      print ("Enter the adjacency matrix")
      for i in range (0, n):
            a = list (map (int, input ( ) . split (" ")))
            matrix. append (a)

      g = graph (n)
      for i in range (0, n):
          for j in range ( 0, n):
               if matrix [i] [j] = = 1
                     g.add_edge ( i, j, 1 )

      for k in range (0, n ):
            print ("For router", k)
            g. bellman_ford (k)

main( )
```