Batch 3

CN-LAB

Chirag Swamy
1BM19CS402

Distance Vector Algorithm :

```
Class Graph :
    def_init_ (self, vertices):
        self.V = vertices
        self.graph = []

    ~~def~~ def add_edge (self, s, d, w):
        self.graph.append ([s, d, w])

    def print_solution (self, dist, src, next_hop):
        print ("Routing table for ", src)
        print ("Dest \t cost \t next hop)
        for i in range (self, v):
            print ("{0} \t {1} \t {2}".format (i, dist[i], next_hop[i]))

    def bellman_ford (self, src):
        dist = [99] * self.V
        dist[src] = 0
        next_hop = {src : src}
        for _ in range (self.V - 1):
            for s, d, w in ~~self~~ self.graph:
                if dist[s] != 99 and dist[s] + w < dist[d]:
                    dist[d] = dist[s] + w
                    if s == src :
                        next_hop[d] = d
                    elif s in next_hop:
                        next_hop[d] = next_hop[s]

        for s, d, w in self.graph:
            if dist[s] != 99 and dist[s] + w < dist[d]:
                print ("Graph contains negative weight cycle")
                return.
```

Chirag Swamy
1BM19CS402

```
def main():
    matrix = []
    print("Enter the no of routers:")
    n = int(input())
    print("Enter the adjacency matrix : Enter 99 for infinity")
    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):
            g.add_edge(i, j, matrix[i][j])

        for k in range(0, n):
            g.bellman_ford(k)

main()
```

Scanned with CamScanner