Vruddhi Shah C-186
60004220215

CN Experiment 6

Aim: To implement single source shortest path algorithms.

Theory:

1. Bellman-Ford: This algorithm finds the shortest path between a source, and all other nodes in a graph if the graph does not contain negative-weight cycles.

```
BELLMANFORD (V, E, s, w)
    $\forall v \in V$, $d[v] \leftarrow \infty$
     $\pi[v] \leftarrow$ nil
     $d[s] \leftarrow 0$
     for i from $1 \rightarrow n-1$ do
         for $(u, v) \in E$ do
             $d[v] \leftarrow \min(d[v], d[u] + w(u,v))$
             if $d[v]$ changes, then $\pi[v] \leftarrow u$.
     for $(u, v) \in E$ do
         if $d[v] > d[u] + w(u, v)$
             return "Negative cycle"
     return d
```

The time complexity of this algorithm is $O(V * E)$.

2. Dijkstra's algorithm: This is also a single source shortest path algorithm. It works if weights of the edges are non-negative.

```
DIJKSTA(v, E, w, s)
    S = $\phi$
    $\forall v \in V$, $d[v] \leftarrow \infty$. $\pi[v] \leftarrow nil$
    $d[s] \leftarrow 0$
    $Q = V$
    while $Q \neq \phi$
        $u$ = EXTRACT-MIN(Q)
        $S = S \cup \{u\}$
        for each vertex $v$ s.t. $(u, v) \in E$, do
            $d[v] = \min(d[v], d[u] + w(u, v))$
            $\pi[v] \leftarrow u$
```

If we use a priority queue, the time complexity is $O(V + E \log V)$.

Conclusion: We have successfully studied and implemented Dijkstra and Bellman-Ford algorithms.

Department of Computer Engineering

Class: T.Y. B.Tech. **Semester: V**

Course Code: DJS22CEL502 **Course Name: Computer Networks Lab**

Name: Vruddhi Shah	**SAP ID: 60004220215**
Date of Performance: 4 .09.24	**Date of Submission: 11.09.24**

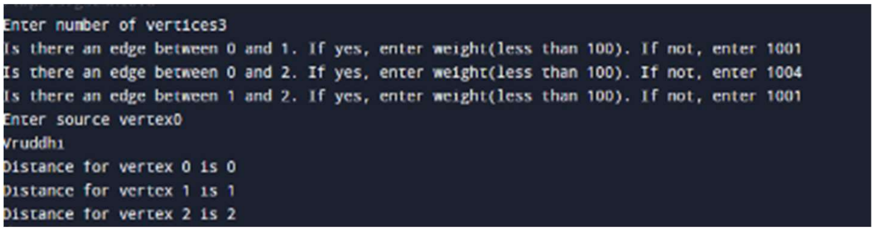
Experiment No: 6

Aim: To find shortest path using Dijikstra's and

Bellman Ford algorithms.

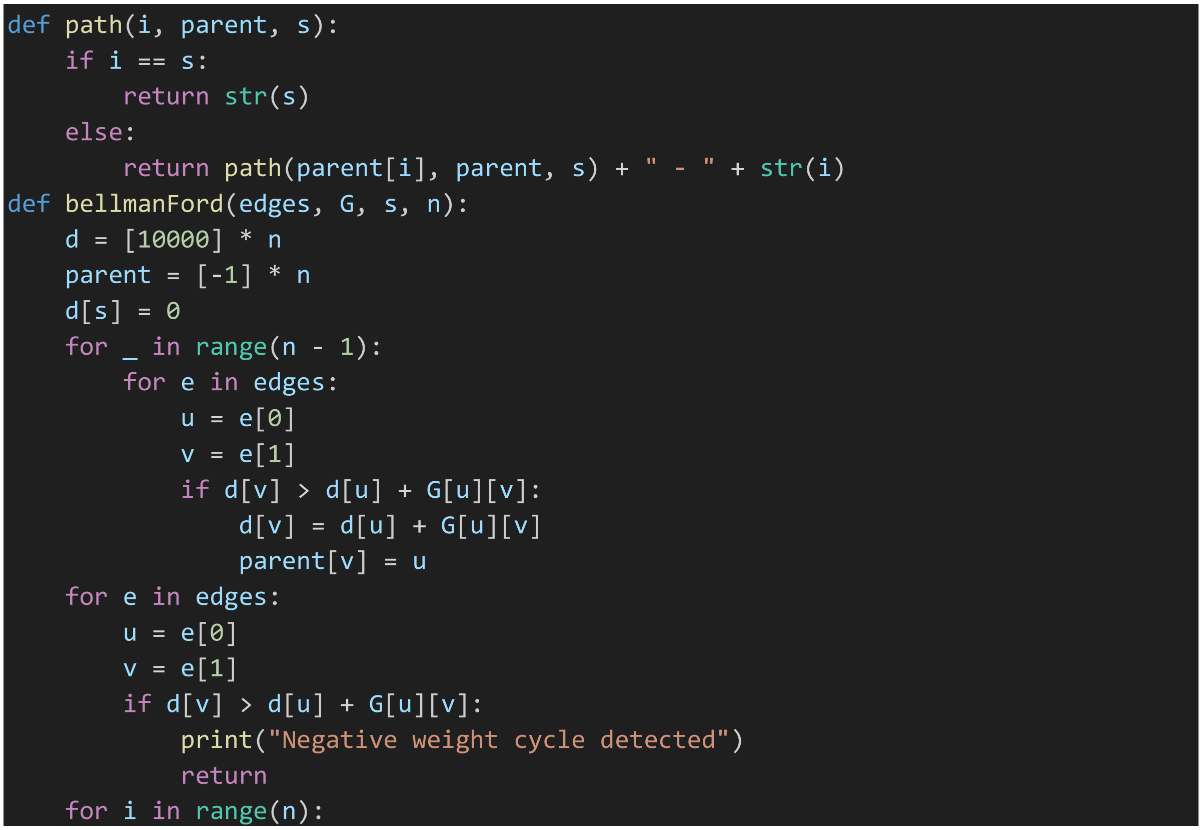
Dijkstra:

```
#include <stdio.h>

int adjMat[100][100];
int visited[100];
int distance[100];
 int main() {
   int n, w, i, j, s, now, k, min;
   printf("Enter number of vertices");
   scanf("%d", &n);
   for (i=0;i<n;i++){
      visited[i] = 0;
   }
   for (i=0;i<n;i++){
      adjMat[i][i] = 0;
      for (j=i+1;j<n;j++){
         printf("Is there an edge between %d and %d. If yes, enter weight(less than 100). If not, enter
100", i, j);
         scanf("%d", &w);
         adjMat[i][j] = w;
         adjMat[j][i] = w;
      } }
  printf("Enter source vertex");
   scanf("%d", &s);
    visited[s] = 1;
   now = s;
   for (i=0;i<n;i++){
      distance[i] = adjMat[s][i];
   }

 for (k=1; k<n; k++){
   min = 100;
   for (i=0;i<n;i++){
      if(distance[i] < min && visited[i] == 0){
         now = i;
         min = distance[i];
      }
   if (visited[now] == 0){
      visited[now] = 1;
      for(j=0; j<n; j++){
         if (distance[j] > distance[now] + adjMat[now][j] + distance[now]){
            distance[j] = distance[now] + adjMat[now][j];
         }
      } }
   } }
   printf("Vruddhi\n");
   for (i=0;i<n;i++){
   printf("Distance for vertex %d is %d\n",i,distance[i]);
   }

   return -1;
}
```

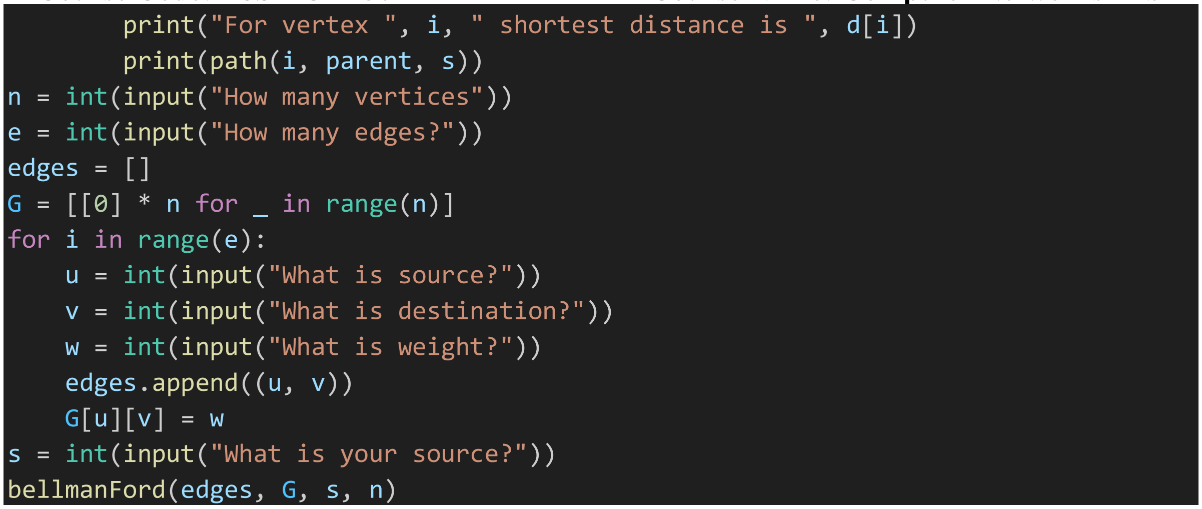


```
Enter number of vertices3
Is there an edge between 0 and 1. If yes, enter weight(less than 100). If not, enter 1001
Is there an edge between 0 and 2. If yes, enter weight(less than 100). If not, enter 1004
Is there an edge between 1 and 2. If yes, enter weight(less than 100). If not, enter 1001
Enter source vertex0
Vruddhi
Distance for vertex 0 is 0
Distance for vertex 1 is 1
Distance for vertex 2 is 2
```

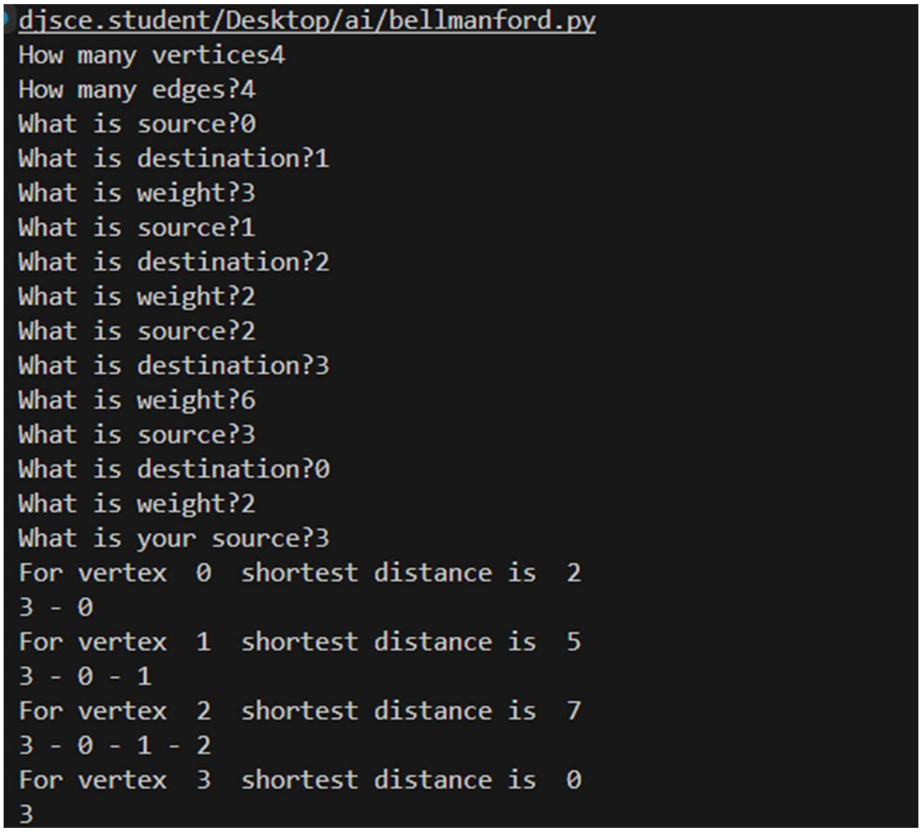
Bellman-Ford:



```python
def path(i, parent, s):
    if i == s:
        return str(s)
    else:
        return path(parent[i], parent, s) + " - " + str(i)
def bellmanFord(edges, G, s, n):
    d = [10000] * n
    parent = [-1] * n
    d[s] = 0
    for _ in range(n - 1):
        for e in edges:
            u = e[0]
            v = e[1]
            if d[v] > d[u] + G[u][v]:
                d[v] = d[u] + G[u][v]
                parent[v] = u
    for e in edges:
        u = e[0]
        v = e[1]
        if d[v] > d[u] + G[u][v]:
            print("Negative weight cycle detected")
            return
    for i in range(n):
```



```
        print("For vertex ", i, " shortest distance is ", d[i])
        print(path(i, parent, s))
n = int(input("How many vertices"))
e = int(input("How many edges?"))
edges = []
G = [[0] * n for _ in range(n)]
for i in range(e):
    u = int(input("What is source?"))
    v = int(input("What is destination?"))
    w = int(input("What is weight?"))
    edges.append((u, v))
    G[u][v] = w
s = int(input("What is your source?"))
bellmanFord(edges, G, s, n)
```



```
djsce.student/Desktop/ai/bellmanford.py
How many vertices4
How many edges?4
What is source?0
What is destination?1
What is weight?3
What is source?1
What is destination?2
What is weight?2
What is source?2
What is destination?3
What is weight?6
What is source?3
What is destination?0
What is weight?2
What is your source?3
For vertex  0  shortest distance is  2
3 - 0
For vertex  1  shortest distance is  5
3 - 0 - 1
For vertex  2  shortest distance is  7
3 - 0 - 1 - 2
For vertex  3  shortest distance is  0
3
```

Conclusion:

Thus, we have studied and implemented shortest path algorithms.