Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2022-2023

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	S. Y. B.Tech (Computer Engineering)
Course:	Analysis of Algorithm Laboratory
Course Code:	DJ19CEL404
Experiment No.:	07

AIM: **IMPLEMENT SINGLE SOURCE SHORTEST PATH USING DYNAMIC PROGRAMMING**

THEORY:

BELLMAN-FORD ALGORITHM

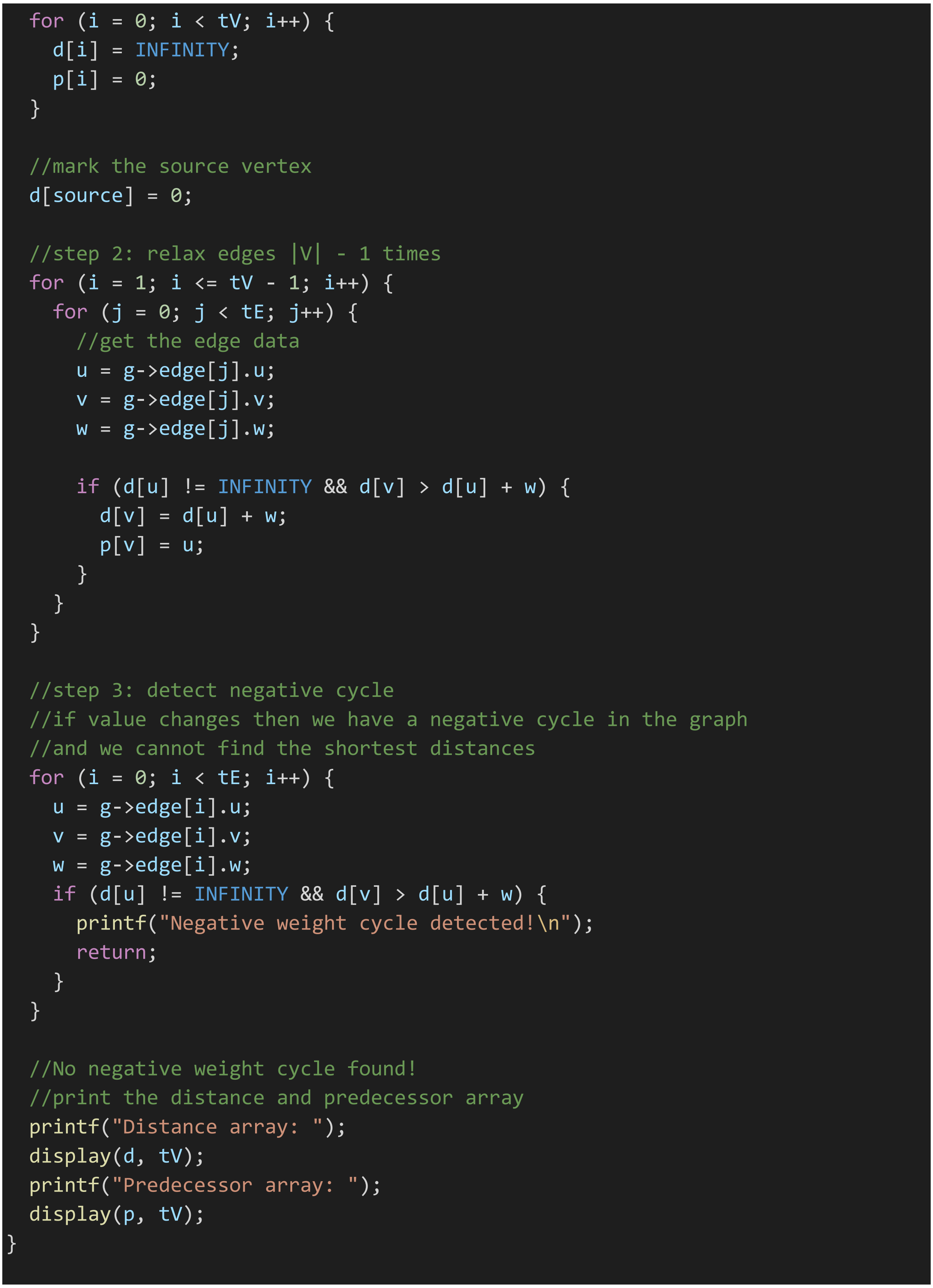
- The single source shortest path algorithm (for arbitrary weight positive or negative) is also known Bellman-Ford algorithm is used to find minimum distance from source vertex to any other vertex.
- The main difference between this algorithm with Dijkstra’s algorithm is, in Dijkstra’s algorithm we cannot handle the negative weight, but here we can handle it easily.
- Bellman-Ford algorithm finds the distance in bottom-up manner.
- At first it finds those distances which have only one edge in the path. After that increase the path length to find all possible solutions.
- **Pseudocode:**

```
function bellmanFord(G, S)
  for each vertex V in G
    distance[V] <- infinite
      previous[V] <- NULL
  distance[S] <- 0

  for each vertex V in G
    for each edge (U,V) in G
      tempDistance <- distance[U] + edge_weight(U, V)
      if tempDistance < distance[V]
        distance[V] <- tempDistance
        previous[V] <- U

  for each edge (U,V) in G
    If distance[U] + edge_weight(U, V) < distance[V}
      Error: Negative Cycle Exists

  return distance[], previous[]
```

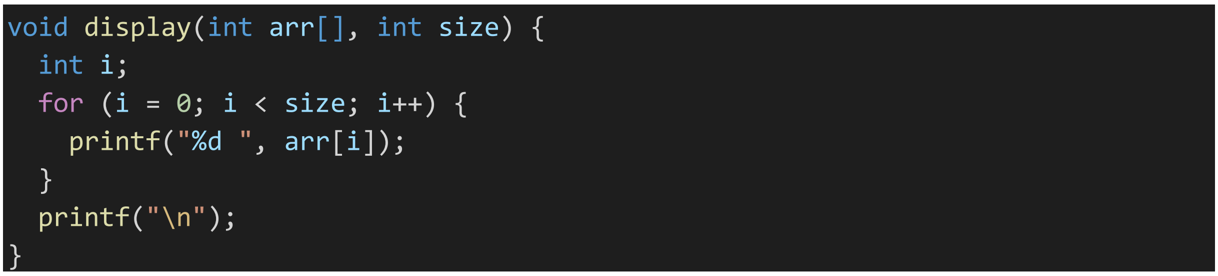
CODE:

```
// Bellman Ford Algorithm in C

#include <stdio.h>
```

```
#include <stdlib.h>

#define INFINITY 99999

//struct for the edges of the graph
struct Edge {
  int u;  //start vertex of the edge
  int v;  //end vertex of the edge
  int w;  //weight of the edge (u,v)
};

//Graph - it consists of edges
struct Graph {
  int V;        //total number of vertices in the graph
  int E;        //total number of edges in the graph
  struct Edge *edge;  //array of edges
};

void bellmanford(struct Graph *g, int source);
void display(int arr[], int size);

int main(void) {
  //create graph
  struct Graph *g = (struct Graph *)malloc(sizeof(struct Graph));
  g->V = 4;  //total vertices
  g->E = 5;  //total edges

  //array of edges for graph
  g->edge = (struct Edge *)malloc(g->E * sizeof(struct Edge));

  //------- adding the edges of the graph
  /*
        edge(u, v)
        where   u = start vertex of the edge (u,v)
                v = end vertex of the edge (u,v)

        w is the weight of the edge (u,v)
    */

  //edge 0 --> 1
  g->edge[0].u = 0;
  g->edge[0].v = 1;
  g->edge[0].w = 5;
  //edge 0 --> 2
  g->edge[1].u = 0;
  g->edge[1].v = 2;
  g->edge[1].w = 4;

  //edge 1 --> 3
  g->edge[2].u = 1;
  g->edge[2].v = 3;
  g->edge[2].w = 3;

  //edge 2 --> 1
  g->edge[3].u = 2;
  g->edge[3].v = 1;
  g->edge[3].w = 6;

  //edge 3 --> 2
  g->edge[4].u = 3;
  g->edge[4].v = 2;
  g->edge[4].w = 2;

  bellmanford(g, 0);  //0 is the source vertex

  return 0;
}

void bellmanford(struct Graph *g, int source) {
  //variables
  int i, j, u, v, w;

  //total vertex in the graph g
  int tV = g->V;

  //total edge in the graph g
  int tE = g->E;

  //distance array
  //size equal to the number of vertices of the graph g
  int d[tV];

  //predecessor array
  //size equal to the number of vertices of the graph g
  int p[tV];

  //step 1: fill the distance array and predecessor array
```

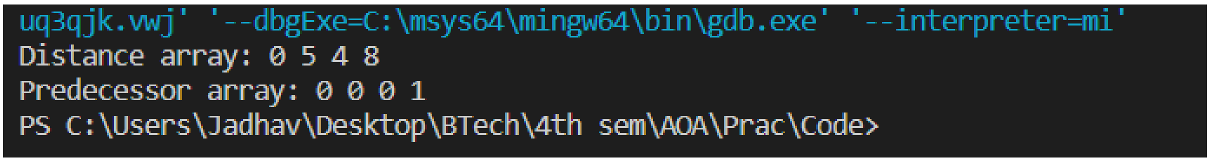


```
  for (i = 0; i < tV; i++) {
    d[i] = INFINITY;
    p[i] = 0;
  }

  //mark the source vertex
  d[source] = 0;

  //step 2: relax edges |V| - 1 times
  for (i = 1; i <= tV - 1; i++) {
    for (j = 0; j < tE; j++) {
      //get the edge data
      u = g->edge[j].u;
      v = g->edge[j].v;
      w = g->edge[j].w;

      if (d[u] != INFINITY && d[v] > d[u] + w) {
        d[v] = d[u] + w;
        p[v] = u;
      }
    }
  }

  //step 3: detect negative cycle
  //if value changes then we have a negative cycle in the graph
  //and we cannot find the shortest distances
  for (i = 0; i < tE; i++) {
    u = g->edge[i].u;
    v = g->edge[i].v;
    w = g->edge[i].w;
    if (d[u] != INFINITY && d[v] > d[u] + w) {
      printf("Negative weight cycle detected!\n");
      return;
    }
  }

  //No negative weight cycle found!
  //print the distance and predecessor array
  printf("Distance array: ");
  display(d, tV);
  printf("Predecessor array: ");
  display(p, tV);
}
```



```
void display(int arr[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}
```

OUTPUT:



```
uq3qjk.vwj' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'
Distance array: 0 5 4 8
Predecessor array: 0 0 0 1
PS C:\Users\Jadhav\Desktop\BTech\4th sem\AOA\Prac\Code>
```

CONCLUSION:

- **Time Complexity**

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

- **Space Complexity**
  The space complexity is O(V).

- Bellman Ford's Algorithm Applications
  - For calculating shortest paths in routing algorithms
  - For finding the shortest path