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.:	04

AIM: IMPLEMENT SINGLE SOURCE SHORTEST PATH USING GREEDY APPROACH

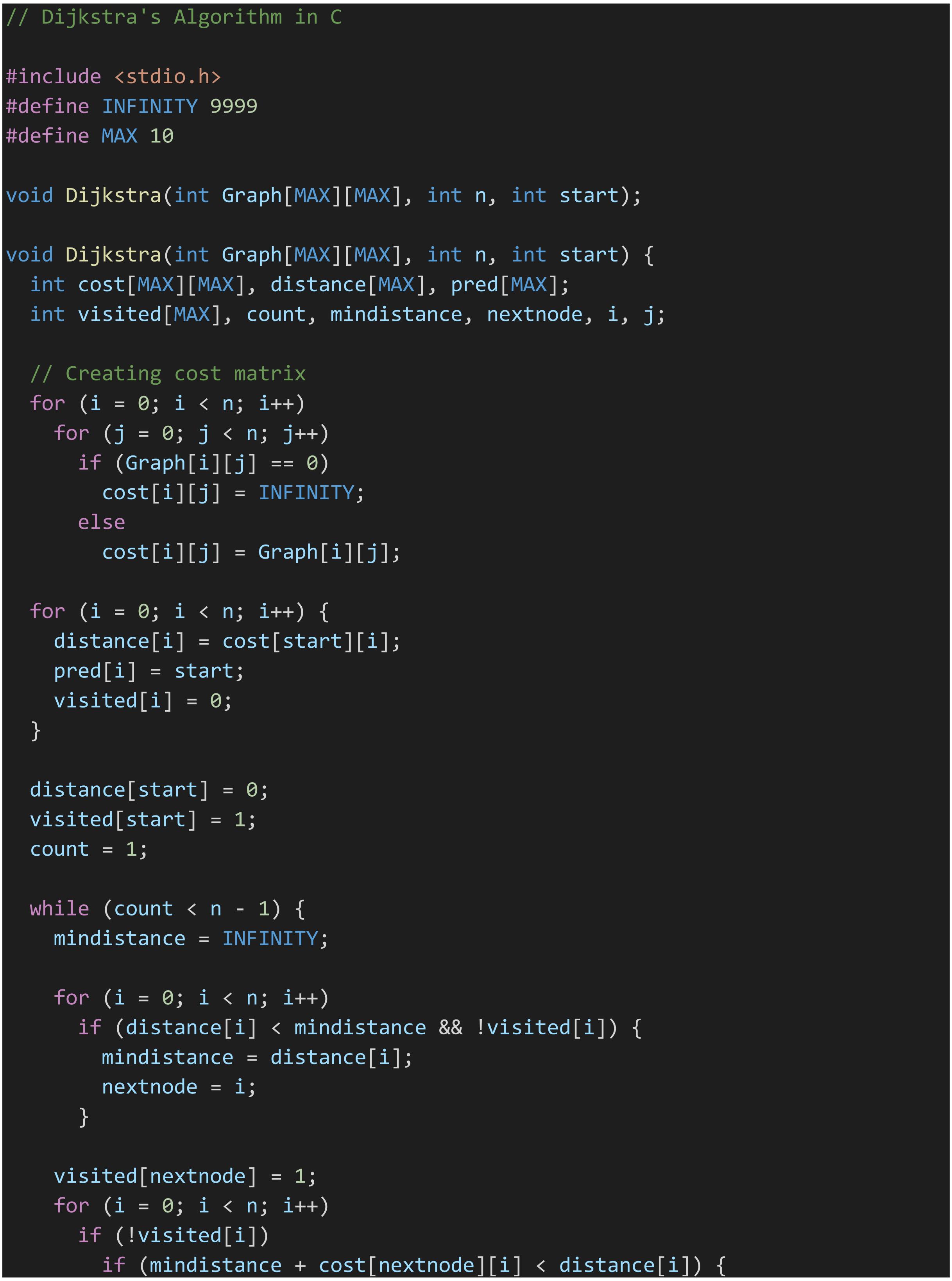
THEORY:

- Dijkstra Algorithm is a very famous greedy algorithm.
- It is used for solving the single source shortest path problem.
- It computes the shortest path from one particular source node to all other remaining nodes of the graph.
- Conditions:
  - Dijkstra algorithm works only for connected graphs.
  - Dijkstra algorithm works only for those graphs that do not contain any negative weight edge.
  - It only provides the value or cost of the shortest paths.
  - By making minor modifications in the actual algorithm, the shortest paths can be easily obtained.
  - Dijkstra algorithm works for directed as well as undirected graphs.
- **Algorithm:**

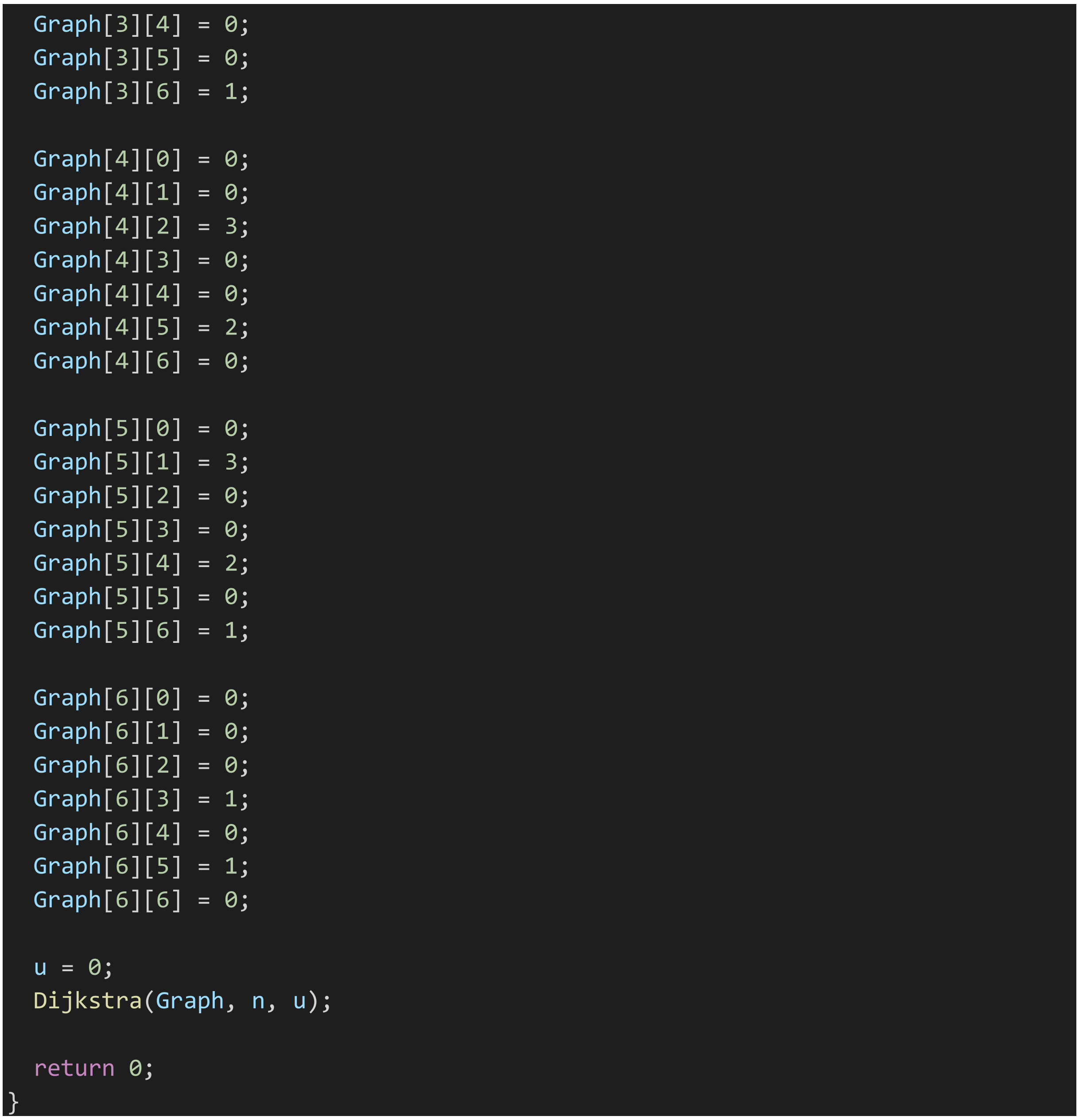
```
Algorithm: Dijkstra’s-Algorithm (G, w, s)
for each vertex v Є G.V
   v.d := ∞
   v.∏ := NIL
s.d := 0
S := Φ
Q := G.V
while Q ≠ Φ
   u := Extract-Min (Q)
   S := S U {u}
   for each vertex v Є G.adj[u]
      if v.d > u.d + w(u, v)
         v.d := u.d + w(u, v)
         v.∏ := u
```

- First for loop does initialization in O(|V|) time. As there are |V| nodes in the graph, size of queue Q would be V, and hence while loop iterates |V| times in worst case. For loop inside while loop run maximum |V| time because a node can have maximum |V| – 1 neighbour. The worst case upper bound running time of this algorithm is described as O(|V2|).

CODE:

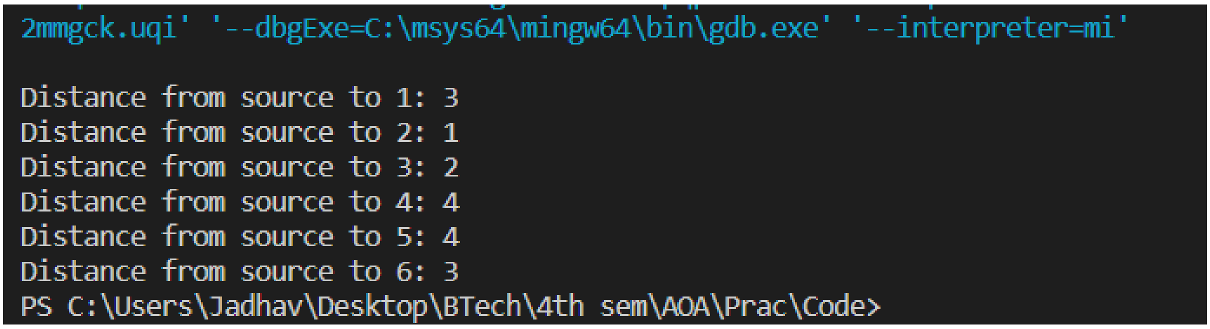


```
// Dijkstra's Algorithm in C

#include <stdio.h>
#define INFINITY 9999
#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {
  int cost[MAX][MAX], distance[MAX], pred[MAX];
  int visited[MAX], count, mindistance, nextnode, i, j;

  // Creating cost matrix
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (Graph[i][j] == 0)
        cost[i][j] = INFINITY;
      else
        cost[i][j] = Graph[i][j];

  for (i = 0; i < n; i++) {
    distance[i] = cost[start][i];
    pred[i] = start;
    visited[i] = 0;
  }

  distance[start] = 0;
  visited[start] = 1;
  count = 1;

  while (count < n - 1) {
    mindistance = INFINITY;

    for (i = 0; i < n; i++)
      if (distance[i] < mindistance && !visited[i]) {
        mindistance = distance[i];
        nextnode = i;
      }

    visited[nextnode] = 1;
    for (i = 0; i < n; i++)
      if (!visited[i])
        if (mindistance + cost[nextnode][i] < distance[i]) {
          distance[i] = mindistance + cost[nextnode][i];
          pred[i] = nextnode;
        }
    count++;
  }

  // Printing the distance
  for (i = 0; i < n; i++)
    if (i != start) {
      printf("\nDistance from source to %d: %d", i, distance[i]);
    }
}
int main() {
  int Graph[MAX][MAX], i, j, n, u;
  n = 7;

  Graph[0][0] = 0;
  Graph[0][1] = 0;
  Graph[0][2] = 1;
  Graph[0][3] = 2;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

  Graph[1][0] = 0;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 3;
  Graph[1][6] = 0;

  Graph[2][0] = 1;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
  Graph[2][3] = 1;
  Graph[2][4] = 3;
  Graph[2][5] = 0;
  Graph[2][6] = 0;

  Graph[3][0] = 2;
  Graph[3][1] = 0;
  Graph[3][2] = 1;
  Graph[3][3] = 0;
```



```
  Graph[3][4] = 0;
  Graph[3][5] = 0;
  Graph[3][6] = 1;

  Graph[4][0] = 0;
  Graph[4][1] = 0;
  Graph[4][2] = 3;
  Graph[4][3] = 0;
  Graph[4][4] = 0;
  Graph[4][5] = 2;
  Graph[4][6] = 0;

  Graph[5][0] = 0;
  Graph[5][1] = 3;
  Graph[5][2] = 0;
  Graph[5][3] = 0;
  Graph[5][4] = 2;
  Graph[5][5] = 0;
  Graph[5][6] = 1;

  Graph[6][0] = 0;
  Graph[6][1] = 0;
  Graph[6][2] = 0;
  Graph[6][3] = 1;
  Graph[6][4] = 0;
  Graph[6][5] = 1;
  Graph[6][6] = 0;

  u = 0;
  Dijkstra(Graph, n, u);

  return 0;
}
```

OUTPUT:



```
2mmgck.uqi' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'

Distance from source to 1: 3
Distance from source to 2: 1
Distance from source to 3: 2
Distance from source to 4: 4
Distance from source to 5: 4
Distance from source to 6: 3
PS C:\Users\Jadhav\Desktop\BTech\4th sem\AOA\Prac\Code>
```

CONCLUSION:

Dijkstra's Algorithm Applications

- To find the shortest path
- In social networking applications
- In a telephone network
- To find the locations in the map