S.No: 18	Exp. Name: ***Program to find Single source Shortest path using Dijkstra's Algorithm in weighted directed graph***	**Date:**

Aim:

Program to find Single source Shortest path using Dijkstra's Algorithm in weighted directed graph

Source Code:

dijkstrasAlgorithm.c

```
#include<stdio.h>
#include<limits.h>
int n, k;
#define perm 1
#define tent 2
#define infinity INT_MAX
typedef struct nodelabel {
    int predecessor;
     int length;
      int label;
       int number;
}
nodelabel;
void initialize_single_source(nodelabel state[], int s, int n) {
    int i;
     for (i = 1; i <= n; i++) {
       state[i].predecessor = 0;
        state[i].length = infinity;
         state[i].label = tent;
          state[i].number = i;
     }
      state[s].predecessor = 0;
       state[s].length = 0;
        state[s].label = perm;
         state[s].number = s;
}
 int parent(int i) {
    return i / 2;
 }
  int left(int i) {
    return 2 * i;
  }
   int right(int i) {
       return 2 * i + 1;
   }
    void min_heapify(nodelabel q[], int i) {
       struct nodelabel temp;
        int l, r, smallest;
             l = left(i);
              r = right(i);
               if (l <= k && q[l].length < q[i].length)
                smallest = l;
                 else
                  smallest = i;
                   if (r <= k && q[r].length < q[i].length)
                    smallest = r;
                     if (smallest != i) {
                         temp = q[i];
                          q[i] = q[smallest];
                           q[smallest] = temp;
                            min_heapify(q, smallest);
                     }
    }
     void build_min_heap(nodelabel q[], int n) {
       int i;
        for (i = n / 2; i >= 1; i--)
         min_heapify(q, i);
     }
      nodelabel heap_extract_min(nodelabel state[]) {
          nodelabel min, temp;
           min = state[1];
            temp = state[1];
             state[1] = state[k];
              state[k] = temp;
               k = k - 1;
                min_heapify(state, 1);
                 return min;
      }
       void heap_decrease_key(nodelabel state[], int key, int i) {
          nodelabel temp;
           state[i].length = key;
            while (i > 1 && state[parent(i)].length > state[i].length) {
                temp = state[i];
                 state[i] = state[parent(i)];
                  state[parent(i)] = temp;
                   i = parent(i);
            }
       }
        void relax(nodelabel u, int a[10][10], nodelabel state[], int i)
        {
          int key;
           if (state[i].length > (u.length + a[u.number][state[i].number])) {
             state[i].predecessor = u.number;
              key = u.length + a[u.number][state[i].number];
               heap_decrease_key(state, key, i);
           }
        }
         void Dijkstra(int a[][10], int n, int s) {
             nodelabel state[10], min;
              int i, count, j, x, dist = 0;
               int path[10];

                 initialize_single_source(state, s, n);
                  build_min_heap(state, n);

                    while (k != 0) {
                      min = heap_extract_min(state);
                       for (i = 1; i <= k; i++)
                        if (a[min.number][state[i].number] > 0 && state[i].label == ten
t)
                         relax(min, a, state, i);
                          min.label = perm;
                           }

                             for (i = 1; i <= n; i++)
                              if (i != s) {
                                 j = i;
                     dist = 0;
                      count = 0;
                       do {
                         count++;
                          path[count] = j;
                           for (k = 1; k <= n; k++)
                            if (state[k].number == j) {
```

```
                        j = state[k].predecessor;
                        break;
                       }
                        } while (j != 0);
                         for (j = 1; j <= count / 2; j++) {
                            x = path[j];
                             path[j] = path[count - j + 1];
                              path[count - j + 1] = x;
                         }

                           for (j = 1; j < count; j++)
                            dist += a[path[j]][path[j + 1]];

                              printf("Shortest path from %d to %d is :", s, i);
                               if (count != 1)
                                printf("%d", path[1]);
                                 else
                                  printf("No path from %d to %d", s, i);
                                   for (j = 2; j <= count; j++)
                                    printf("-->%d", path[j]);
                                     printf("\nDistance from node %d to %d is : %d",s,
i, dist);
                                      printf("\n");
                                       }
                                        }
                                         int main() {
                                          int a[10][10], i, j, source;

                           printf("Enter the number of nodes :");
                            scanf("%d", & n);
                              for (i = 1; i <= n; i++) {
          printf("Enter node %d connectivity :", i);
          for (j = 1; j <= n; j++)
           scanf("%d", & a[i][j]);
            }
             k = n;
             printf("Enter the source node :");
             scanf("%d", & source);

             Dijkstra(a, n, source);
             return 0;
             }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of nodes : 3
Enter node 1 connectivity : 1 2 0
Enter node 2 connectivity : 0 5 6
Enter node 3 connectivity : 5 3 0
Enter the source node : 1
Shortest path from 1 to 2 is :1-->2
Distance from node 1 to 2 is : 2
Shortest path from 1 to 3 is :1-->2-->3
Distance from node 1 to 3 is : 8