Getting Closer to Desired Destination

Bachelor of Technology
Computer Science and Engineering

Submitted by

ANURAG GANGULY (13000117**125**)
ANUPAM CHAKRABORTY (13000117**126**)
ANIKET DAS (13000117**130**)
ANANYA PAUL (13000117**131**)

NOVEMBER 2019

Techno Main
EM-4/1, Sector-V, Salt Lake
Kolkata- 700091
West Bengal
India

Contents

Abstract

Our project statement dictates us to reach a particular destination from our current location. So, we start to find a place that will take us closer to our destination and we move there. Now, our location changes and again, we repeat the process.
We stop, once we reach our destination. The objective is to find an optimal solution for the problem.

Introduction

Project Statement

You want to go to a place, say IIT Madras. You have no idea how IITM is connected to your current location, but you do know that going to the airport will get you closer to your destination. So, you go to the airport. Your current location just changed. You now look for another place to go to which will get you closer to IITM. The airplane seat seems like a good place. So, for any current location, we try to find a place X such that it gets us closer to our destination. We do this till we end up at our destination. Apply a design technique to solve the problem.

Suppose, one wants to visit USA from Kolkata. Now there is no flight which directly takes him/her to the United States. He/she has to reach Delhi, then take a second flight to Qatar from where he/she can take a direct flight to USA. However, he/she can also skip Delhi and directly avail a flight to Qatar in which case the time duration will be much more as compared to the first scenario. Our problem statement is a fictional note on many such examples, and we have decided to implement the Dijkstra's algorithm for this problem.

Dijkstra's algorithm is used to find the shortest path from a single source vertex to all other vertices in the given graph, and we will use this algorithm to find out our required solution.

A correct algorithm will not only solve the problem, but also, reduce the time complexity. The technique which will solve the problem efficiently in the minimum amount of time will be chosen as our ideal algorithm for solving the problem.

Body

a) Pseudo Code

V contains the total number of vertices.

Initialize minval to 0; temp to 9999; single source dist[src] to 0 and rest of its element to the maximum value and sptSet[] to false; elen[] contains all the shortest distances from every vertex to destination; nds[] will contain the path.

u stores the current node.

Initialise min to clen[u].

for each u not equal to V-1 do

Initialise sptSet[u] to true; nds[i] to u; increment i by 1; temp to 9999.

for each v = 0 to V-1 do

if v not equal to destination then

if graph[u][v] is true and !sptSet[v] then

Initialise minval to cen[v].

if minval<min then

Initialise min to minval and temp to v.

Initialise u to temp.

if temp equals to 9999 then

Initialise sptSet[destination] to true; nds[i] to destination and break out from the external loop.

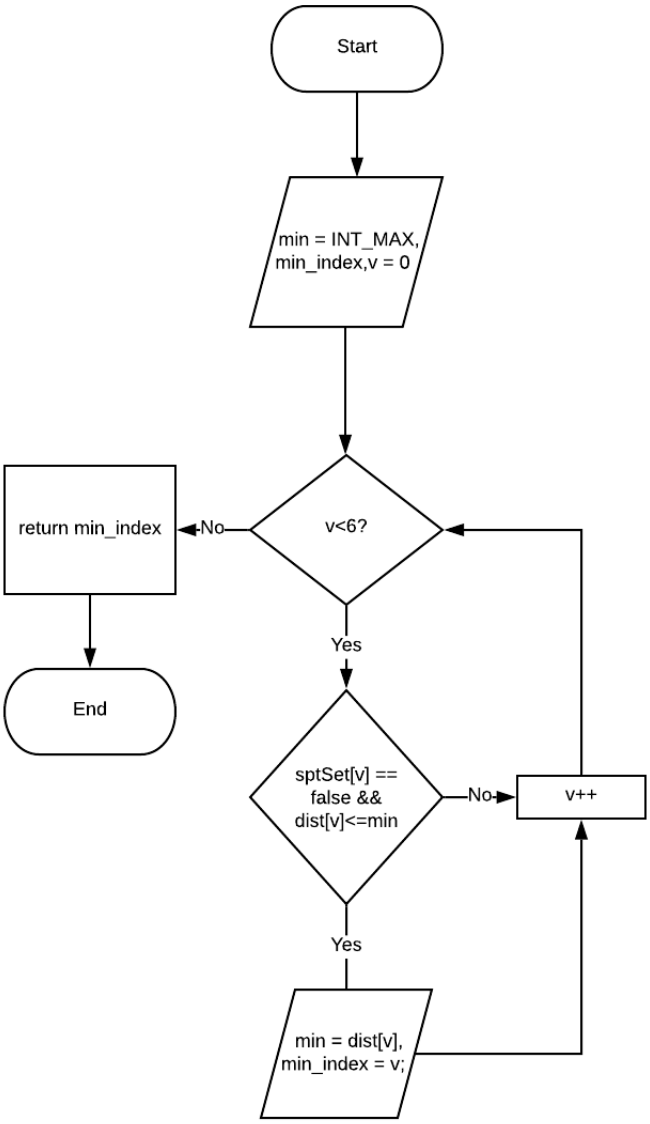
for j = 0 to i do

print nds[j]

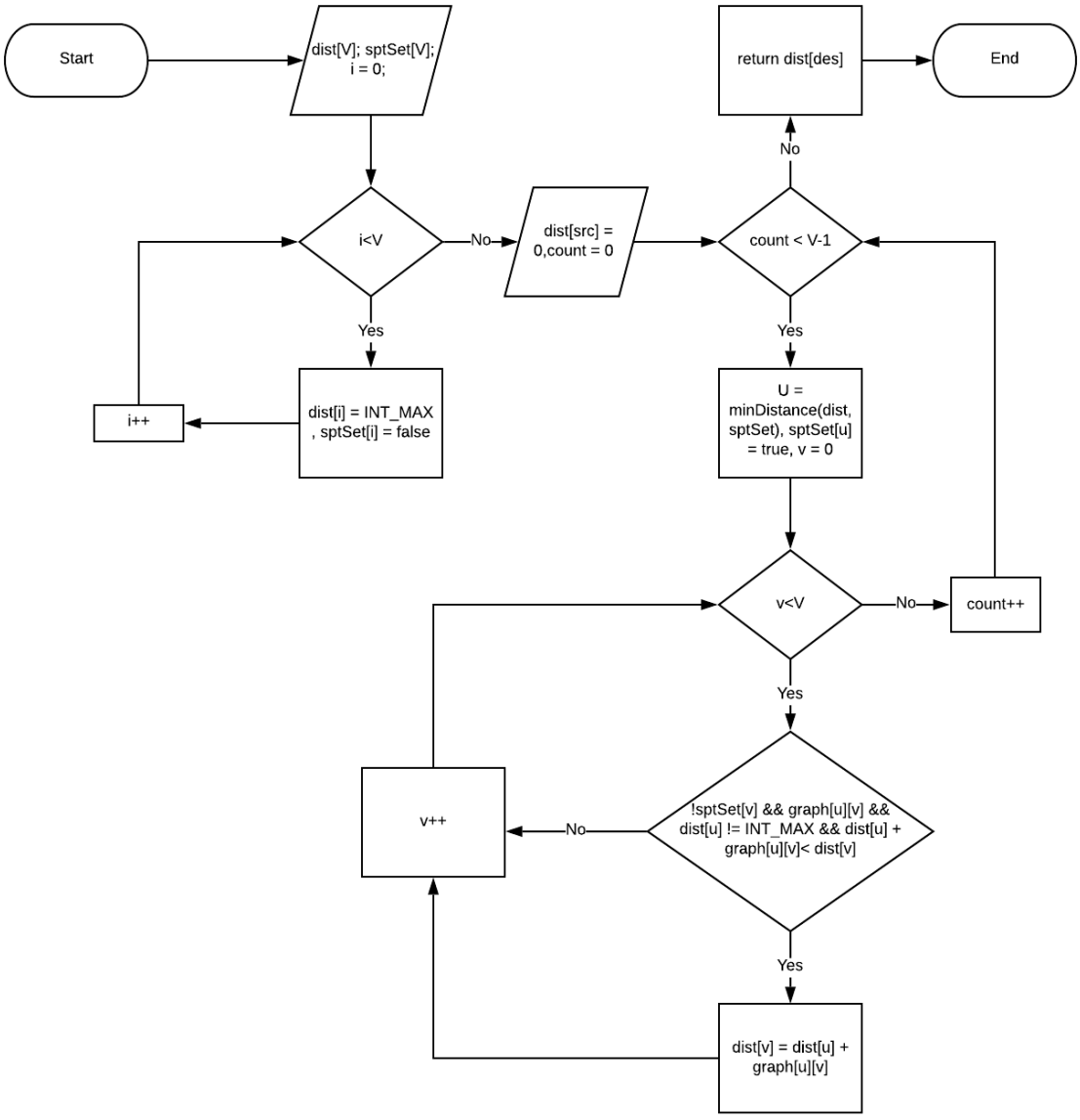
End

b) <u>Design of the solution using flowchart</u>

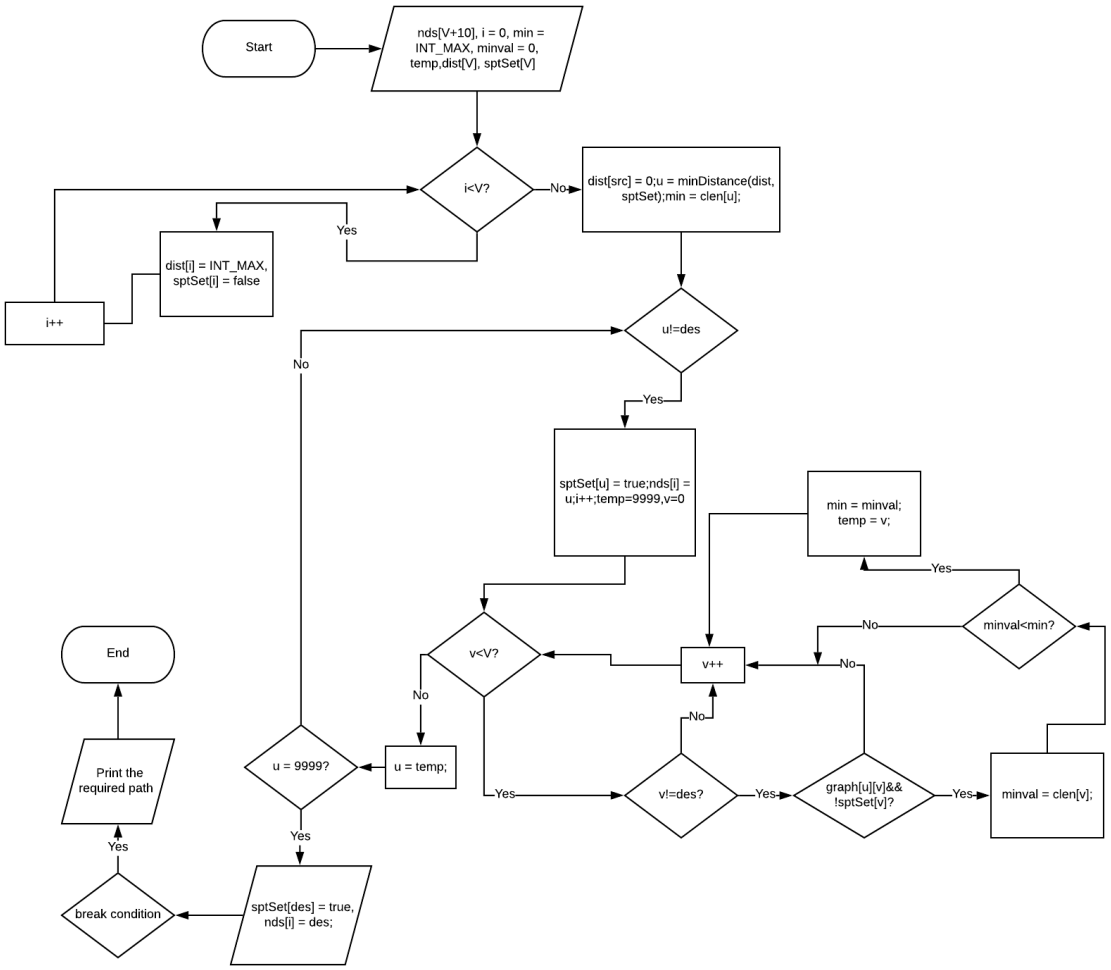
int minDistance(int dist[], bool sptSet[])



int dijkstra(int graph[V][V], int src, int des)



void getSolution (int graph[V][V], int src, int des, int clen[V])



c) Solution (Mathematically) and its complexity analysis

The time complexity for Dijkstra's algorithm is $O(V^2)$ where V is the number of vertices.
Considering our program and the logic that has been applied, the total worst case time complexity is $O(V^3)$.

d) Implementation using C

```
#include <limits.h>

#include <stdio.h>

//Total number of vertex in the graph

#define V 6

//Method to calculate the minimum distance between source and other vertices

int minDistance(int dist[], bool sptSet[])

{

        int min = INT_MAX, min_index;

        for (int v = 0; v < V; v++)

                if (sptSet[v] == false && dist[v] <= min)

                        min = dist[v], min_index = v;

        return min_index;

}


/*Method to calculate the shortest distances from source to all other vertices and returning the shortest

distance from source to the destination*/

int dijkstra(int graph[V][V], int src, int des)

{

        int dist[V];
```

```
        bool sptSet[V];
        for (int i = 0; i < V; i++)
                dist[i] = INT_MAX, sptSet[i] = false;
        dist[src] = 0;
        for (int count = 0; count < V - 1; count++) {
                int u = minDistance(dist, sptSet);
                sptSet[u] = true;
                for (int v = 0; v < V; v++)
    {
                        if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                                && dist[u] + graph[u][v] < dist[v])
                                dist[v] = dist[u] + graph[u][v];
    }
        }
        return dist[des];
}
```

//Method to calculate the node which is closer to the destination and thus printing the correct sequence

```
void getSolution(int graph[V][V], int src, int des, int clen[V])
{
   int nds[V+10],i=0;
   int min = INT_MAX, minval = 0, temp;
        int dist[V];
        bool sptSet[V];
```

```
    for (int i = 0; i < V; i++)
            dist[i] = INT_MAX, sptSet[i] = false;
    dist[src] = 0;
    int u = minDistance(dist, sptSet);
    min = clen[u];
    while(u!=des)
    {
        sptSet[u] = true;
        nds[i] = u;
        i++;
        temp = 9999;

for(int v = 0; v < V; v++)
{
    if(v!=des)
    {
        if(graph[u][v] && !sptSet[v])
        {
            minval = clen[v];

            if(minval<min)
            {
                min = minval;
                temp=v;
            }
```

```
            }
        }
    }
    u = temp;
    if(u == 9999)
    {
        sptSet[des] = true;
        nds[i] = des;
        break;
    }
        }
        printf("The path is: \n");

  for(int j = 0; j <= i; j++)
  {
    if(j==i)
        printf("%d",nds[j]);
    else
        printf("%d ---> ",nds[j]);
  }
  printf("\n");
}
//Driver method
int main()
{
```

```
FILE *fp;
fp = fopen("input.txt", "r");      //The file which contains the adjacency matrix of the graph


if(fp == NULL)
{
   printf("File failed to open");
   return 0;
}
int graph[V][V];
for(int i = 0; i < V; i++)          //Reading the values into the graph[][] array
{
   for(int j = 0; j < V; j++)
   {
      fscanf(fp, "%d", &graph[i][j]);
   }
}
```

int clen[V]; //clen[] array to store the shortest distance from vertices(sparing source and destination) to the destination

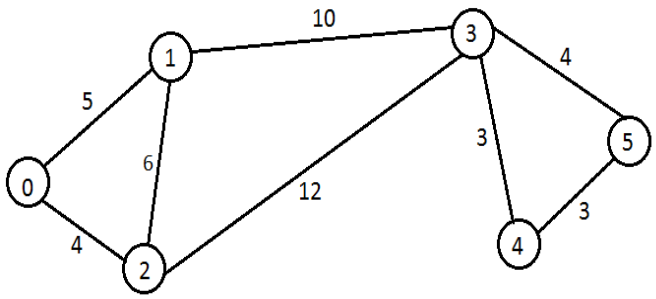
```
for(int i = 0; i < V; i++)
{
   clen[i] = 0;
}
int src,des;
printf("***** Choose from 0 to %d *****\n",(V-1));
printf("Enter the source ---> ");
```

```
scanf("%d",&src);
if(src>(V-1) || src<0)
{
   printf("Source does not  exist!!!!\n");
   return 0;
}
printf("\n");
printf("Enter the destination ---> ");
scanf("%d",&des);
if(des>(V-1) || des<0)
{
   printf("Destination does not  exist!!!!\n");
   return 0;
}
printf("\n");
if(src == des)
{
   printf("No routes found!!");
   printf("\n");
   return 0;
}
for(int i = 0; i < V; i++)
{
   if(i!=des)
   clen[i] = dijkstra(graph,i,des);
```

```
    }
    getSolution(graph,src,des,clen);
    fclose(fp);
        return 0;
}
```

e) Testing

Input



Output - 1

Enter the source : 0

Enter the destination : 5

The path is :: 0-->1-->2-->3-->4-->5

Output - 2

Enter the source : 2

Enter the destination : 4

The path is :: 2-->3-->4

f) Advantages & Disadvantages

The main advantage of our algorithm is that, for every iteration, we find a place which is closer to our destination. Thus, we move towards our target destination at each step.

Referring to our graph, one might point out a particular disadvantage where, if someone is standing at vertex "3", the destination is directly connected to it and cost is '4'.

But, we choose to go to vertex "4", as it is closer to our destination, though our cost increases.

Conclusion

We found out an optimal solution to the given problem which could also be implemented in many real-life problems resembling our project. The project helped us explore the advantages and disadvantages of the Dijkstra's algorithm, how it could be implemented to our benefits but at the cost of some limitations. In our daily lives, we may unknowingly implement one of the algorithms to solve a problem but may not realize it explicitly. The project helped us develop some practical ideas which might further nurture our brains to find out something innovative which could benefit a lot of people.

The time complexity can be improved by using a Fibonacci heap which are optimal but those are very complicated to implement and memory intensive.

References

i. https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/

ii. https://www.youtube.com/watch?v=XB4MIexjvY0 - Dijkstra Algorithm - Single Source Shortest Path – Greedy Method

iii. https:// //www.codingame.com/playgrounds/1608/shortest-paths-with-dijkstra's algorithm/Dijkstra's algorithm

iv. Fundamentals of Computer Algorithms by Ellis Horowitz