CRACK A HACK

DIJKSTRA's ALGORITHM

Course: ALGORITHMIC PROBLEM SOLVING

Course code: 17ECSE309

by:

Himanshu Goyal

USN: 01FE15BCS076

1. Introduction

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants. Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

2. Example

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infinity	infinity
1	AD	2,A	4,D		2,D	infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					

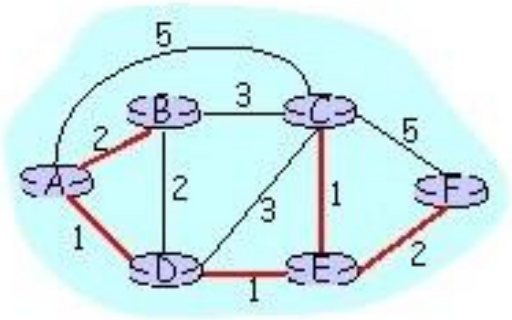

Figure 2.1: Dijksta's Algorithm example

3. Algorithm

```
1  function Dijkstra(Graph, origin, destination):
2      for each vertex v in Graph:           // Initializations
3          dist[v] := infinity,    previous[v] := undefined
4      dist[origin] := 0                     // Distance from origin to origin is 0
5      Q := the set of all nodes in Graph // All nodes in the graph are unoptimized - thus are in Q

6      while Q is not empty:                 // The main loop
7          u := vertex in Q with smallest dist[]
8          if dist[u] = infinity:     break                    // There is no route from origin to destination
9          if u = destination:        break                    // reach the destination
10         remove u from Q
11         for each neighbor v of u:         // where v has not yet been removed from Q.
12             alt := dist[u] + cost_between(u, v)
13             if alt < dist[v]:             //If the distance is less than the previously recorded distance
14                 dist[v] := alt,       previous[v] := u

15     //read the shortest path
16     S := empty sequence
17     u := destination
18     while previous[u] is defined:
19         insert u at the beginning of S
20         u := previous[u]

21     return S
```

Figure 3.1: Dijksta's Algorithm Pseudocode

4. Code (in C++)

```
#include <map>
#include <set>
#include <list>
#include <cmath>
#include <ctime>
#include <deque>
#include <queue>
#include <stack>
#include <string>
#include <bitset>
#include <cstdio>
#include <limits>
#include <vector>
#include <climits>
#include <cstring>
#include <cstdlib>
#include <fstream>
#include <numeric>
#include <sstream>
#include <iostream>
#include <algorithm>
#include <unordered_map>
using namespace std;

#define INF INT_MAX/2

int mini(int a, int b){
   if(a<b)
      return a;
   return b;
}

void calculatedistance(int nodes, vector<vector<int>> mat, int start){
   int i,j,k,temp,u,min;
   int dist[nodes+1];
   for(i=1;i<nodes+1;i++){     //distance array intialization
      dist[i]=mat[start][i];
   }
   int path[nodes+1];
   for(i=1;i<nodes+1;i++){     //path array intialization
      path[i]=start;
   }
   set<int> R;
   set<int>::iterator itr;
   for(i=1;i<nodes+1;i++){
```

```
        R.insert(i);
    }
    itr=R.find(start);
    R.erase(itr);
    while(!R.empty()){
        min=INF+1;
        for(itr=R.begin();itr!=R.end();itr++){
            if(dist[*itr]<min){
                min=dist[*itr];
                u=*itr;
            }
        }
        itr=R.find(u);
        R.erase(itr);

        min=INF;
        for(itr=R.begin();itr!=R.end();itr++){
            temp=mini(dist[*itr],dist[u]+mat[u][*itr]);
            if(temp<dist[*itr]){
                dist[*itr]=temp;
                path[*itr]=u;
            }
        }
    }

    for(i=1;i<nodes+1;i++){
        if(i!=start && dist[i]!=INF)
            cout<<dist[i]<<" ";
        else if(dist[i]==INF)
            cout<<"-1 ";
    }
    cout<<endl;
}

int main(){
    int t;
    cin>>t;
    int i,j,k,a,b,c;
    for(i = 0; i < t; i++){
        int nodes;
        int edges;
        cin >>nodes>>edges;
        vector<vector<int>> mat(nodes+1,vector<int>(nodes+1,INF));
        for(j=1;j<nodes+1;j++){
            mat[j][j]=0;
        }
        for(j=0;j<edges;j++){
            cin>>a>>b>>c;
            if(mat[a][b]==INF || mat[a][b]>c){  //there might be  multiple edges between same 2 nodes
                mat[a][b]=c;
```

```
            mat[b][a]=c;
        }

    }
    int s;
    cin>>s;
    calculatedistance(nodes,mat,s);
    mat.clear();
  }
  return 0;
}
```

5. Time Complexity

The time complexity of the above code is O(ElogV), Where ‘E’ is the number of edges and ‘V’ is the number of vertices.

6. Applications

- It is used in geographical Maps.
  - To find locations of Map which refers to vertices of graph.
  - Distance between the location refers to edges.
- It is used in IP routing to find Open shortest Path First.
- It is used in the telephone network.

7. References

1) https://www.researchgate.net
2) https://www.cs.umd.edu