FDS Project

Dijkstra Sequence

Date 2024-4-26

1 Introduction

What's Dijikstra Sequence: The Dijkstra algorithm is an algorithm used to find the shortest path between two vertices in a graph. The algorithm works by starting at the source vertex and then iteratively adding the vertex with the smallest distance to the set of vertices that have been visited.

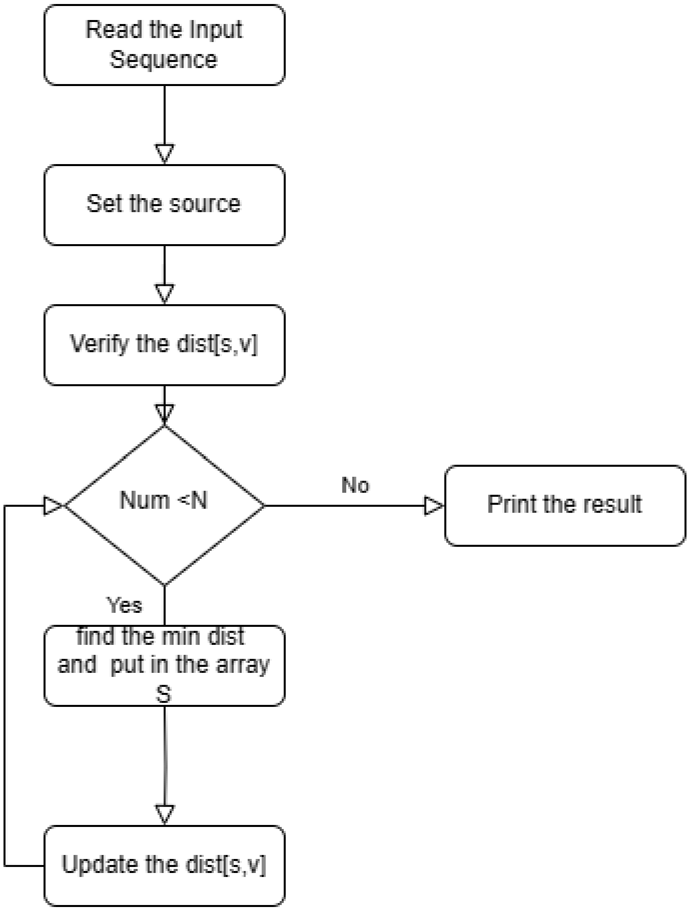
What to be done: According to the data given,generating a graph with N_v vertices and N_e eddges.And then verify whether a given sequence adheres to the properties of a Dijkstra sequence

2 Algorithm Specification

2.1 Data Structure

The adjacency matrix is used to store the graph. The value of the matrix at position (i, j) is the weight of the edge between vertex i and vertex j. If there is no edge between the two vertices, the value is 0.

2.2 Flowchart



2.3 pseudo-code

```
Algorithm :Dijkstra

Input :Directed graph G=(V, E, W) with weight

Output :All the shortest paths from the source vertex s to every

1:S←{s}

2:dist[s,s]<0

3 :for v in V-{s} do:dist[s,v]<-w(s,v)
    (when v not found, dist[s,v]<∞)
```

```
whileV-S≠Ø do:
    find min dist[s, v] from the set V-S

    S←S + v
    for v in V-S do
        if dist[s, v] + w < dist[s,v] then
            dist[s,v]<-dist[s,v] + w
```

3 Testing Results

3.1 Input Specification

The input file contains the following information:

- The first line contains two integers N_v and N_e, representing the number of vertices and edges in the graph, respectively.
- The next N_e lines contain three integers v_1, v_2, and w, representing the vertices v_1 and v_2 and the weight of the edge between them.
- The next line contains an integer k, representing the number of sequences to be verified.
- The next k lines contain N_v integers, representing the sequence to be verified.

3.2 Test cases and results

These test cases may cover almost every situation.

Index	Specification	status
1	Empty graph	pass
2	Sample Input from PTA	pass
3	5 Vertices 7 Edges	pass
4	7 Vertices 11 Edges	pass
5	20 Vertices 50 Edges	pass
6	100 Vertices 300 Edges	pass
7	Extreme Cases with Nv=1000 Ne=10^4	pass
8	Extreme Cases with Nv=1000,Ne=10^5	pass

4 Analysis and Comments

- **TimeComplexity:**$O(N_v^2)$

  The time complexity of the provided code is primarily determined by the ifDijkstra function, which implements the logic to check if a given sequence is a Dijkstra sequence.

The while loop iterates until cnt reaches numv, performing operations such as finding the minimum distance, updating distances, and checking conditions. In the worst case, this loop iterates numv times.

  - Inside the while loop, finding the minimum distance involves iterating over all vertices, taking O(numv) time.
  - Updating distances involves iterating over all vertices, taking O(numv) time.
    Thus, the overall time complexity of the ifDijkstra function is O(numv^2).

- **SpaceComplexity:**$O(N_v^2)$

  The space complexity of the provided code is primarily determined by the adjacency matrix and the input array.

  - The adjacency matrix is of size numv x numv, taking O(numv^2) space.
  - The input array is of size k x numv, taking O(k*numv) space.

- **Further possible comments**

  - The code could be optimized by using a priority queue to find the minimum distance vertex efficiently.
  - The graph could be represented using an adjacency list instead of an adjacency matrix to save space.

5 Appendix

```
#include<stdio.h>
#include<stdlib.h>
#define maxv 1005
#define maxe 100005

int matrix[maxv][maxv]; //adjacency matrix
int input[maxv][maxv]; //used to store the input sequence

int ifDijkstra(int numv,int index){
    /* create a hash_set S to check
    whether the vertices have been visited*/
    int s[numv+1];
    // init the hash_set
    for(int i = 1;i <= numv;i++) s[i] = 0;
    // get the source
    int source = input[index][0];
    s[source] = 1;
    // init the dist
    int dist[numv+1];
    for(int i = 1;i<=numv;i++){
        if(i == source) dist[i] = 0; //if it is the source
        //this means there is no path betwenn source and i
        else if(matrix[source][i] == 0) dist[i] = INT_MAX;
        else dist[i] = matrix[source][i];
    }
    // get the sequence
    int cnt = 1;
    // if the number of vertices is less than 2
    while(cnt < numv){
        // find the min dist
        int mindist = 0;
        for(int i = 1;i<=numv;i++){
            if(s[i] == 1) continue; //if it has been visited
            // if it is the first non-visited vertex
            if(dist[i] != 0 && mindist == 0) mindist = dist[i];
            // if the new dist is smaller than the current min dist
            if(dist[i] > 0 && dist[i] < mindist) mindist = dist[i];
        }
        //check whether the input is min dist
        int newv = input[index][cnt];
        if(dist[newv] != mindist) return 0;
        else s[newv] = 1; //mark it as visited
        cnt++;
        for(int i = 1;i<=numv;i++) //update the dist
                /* if there is a new path between source and i and
                the new path is shorter*/
                if(s[i]==0&&matrix[newv][i] != 0 &&
                dist[newv] + matrix[newv][i] < dist[i])
                    dist[i] = dist[newv] + matrix[newv][i];
    }
    return 1;
}
```

```
int main(int argc,char *argv[]){
    // The basic operation according to the question
    char in[30] = ".\\data\\data1.in";
    char out[30] = ".\\data\\data1.out";
    // chaning dirs according to the system
    if(argv[1][1] == 'l') in[1]=in[6]=out[1]=out[6]='/';
    in[11] = out[11] = argv[2][1];
    // represent the input file and output file
    printf("in:%s\n",in);
    printf("out:%s\n",out);
    FILE *fp = fopen(in,"r"); //open the input file
    FILE *fpp = fopen(out,"w+"); //open the output file
    // get the number of vertices and edges
    int numv,nume;
    fscanf(fp,"%d %d",&numv,&nume);
    //if there is no vertices or edges
    if(numv == 0 || nume == 0) return 0;
    // init the matrix
    for(int i = 0;i<nume;i++){
        int v1,v2,w;
        fscanf(fp,"%d %d %d",&v1,&v2,&w);
        // store the weight of the edge between v1 and v2
        matrix[v1][v2] = matrix[v2][v1] = w;
    }
    int k; // get the input sequence
    fscanf(fp,"%d",&k);
    for(int i = 0;i<k;i++){
        for(int j = 0;j<numv;j++) fscanf(fp,"%d",&input[i][j]);
    }
    // check whether the input sequence is the Dijkstra sequence
    for(int i = 0;i<k;i++){
        if(ifDijkstra(numv,i)) fprintf(fpp,"Yes\n");
        else fprintf(fpp,"No\n"); //if it is not the Dijkstra sequence
    }
}
```

6 Declaration

I hereby declare that all the work done in this project titled "Dijkstra Sequence" is of my independent effort