The 2nd-shortest Path

Yichi Zhang

2022-11-29

1. Introduction

Holiday is coming. Lisa wants to go home by train. But this time, Lisa does not want to go home directly – she has decided to take the second-shortest rather than the shortest path. Your job is to help her find the path.

There are M stations and N unidirectional railway between these stations. For simplicity Lisa starts at No.1 station and her home is at No.M station.

The second-shortest path can backtrack, i.e., use the same road more than once. The second-shortest path is the shortest path whose length is longer than the shortest path(s) (i.e. if two or more shortest paths exist, the second-shortest path is the one whose length is longer than those but no longer than any other path).

2. Algorithm Specification

Algorithm: 2nd short path

Input: Each input file contains one test case. For each case, the first line gives two positive integers M and N which are specified in the above description. Then N lines follow, each contains three space-separated integers: A, B, and D that describe a road from A to B and has length D.

Output: For each test case, print in one line the length of the second-shortest path between node 1 and node M, then followed by the nodes' indices in order.

Main Idea: The implementation of the program focuses on calculating the minimum path in a graph by Dijkstra's algorithm. The second shortest path is calculated by deleting the sub-paths in the shortest path separately to find the shortest path between two nodes. This process only requires modification of the graph and still relies on Dijkstra's algorithm to find the shortest path.

Pseudo Code:

```
int Find2ndMin(){
    while(Loop each edge in the shortest path){
```

```
        Delete this edge;
        Use the Dijkstra algorithm to find the shortest path for the new graph;
        If the second shortest path is generated, save it;
        Restore the graph;
    }
    Output second shortest path;
}
```

3. Testing Results

Table 1 shows some typical test cases for verifying the *Voting Tree* implementation and capturing potential bugs.

All the Test Cases are included in the Appendix B.

Test Cases	Design Purpose	result	status
Case 1	Two shapes similar to example	Result 1	*pass*
Case 2	The smallest graph	Result 2	*pass*
Case 3	random graph	Result 3	*pass*
Case 4	Two shortest path graph	Result 4	*pass*
Case 5	Two 2nd shortest path graph	Result 5	*pass*
...	...	...	...

4. Analysis and Comments

For the runtime of *Find2ndMin*

- Loop edge on the shortest path most using $O(N)$ time units;
- Operation on the graph using $O(1)$ time units;
- Search for shortest side using $O(M^2)$ time units;

Therefore, the worst time complexity can be computed as:

$$T(n) = O(N * M^2).$$

Since most of the graph are simple, the actual time spent will be lower.

For the space requirement, the most we need is a M*M array, the space complexity is $\Theta(M^2)$ and should be optimal.

Appendix A: Source Code (in C)

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 1000

struct Point{
    int number;
    int distance;
    int known;
    int path;
};
//Define the data type of the node, including the distance and the previous node

int M,N;
//Define the number of nodes and the number of edges

int output[MAX]={0};
//Array for storing results

int G[MAX][MAX]={0};
//Define the connection state of the graph

struct Point station0[MAX];
struct Point station1[MAX];
//Array for storing paths

int FindMin(struct Point station[M]);
int Find2ndMin();

int main(){

    scanf("%d %d",&M,&N);
    //Input M,N

    for(int i=0;i<N;i++){
        int x,y,d;
        scanf("%d %d %d",&x,&y,&d);
        G[x-1][y-1]=d;
        //Unidirectional path
    }
    //Save the state of the graph to array

    FindMin(station0);
    //Finding the minimum path
```

```
    Find2ndMin();
    //Finding the 2nd minimum path

}

int FindMin(struct Point station[M]){

    struct Point temp;

    for(int i=0;i<MAX;i++){
        station[i].number=i;
        station[i].distance=10000;
        station[i].known=0;
        station[i].path=-1;
    }
    station[0].distance=0;
    //Initialization

    while(1){

        int Min=10000,MinPos;
        for(int i=0;i<M;i++){
            if(station[i].distance<Min && station[i].known==0){
                temp=station[i];
                MinPos=i;
                Min=station[i].distance;
            }
        }
        //Find the unoperated node with the shortest distance

        if(Min==10000)  break;
        //If not, end the loop

        station[MinPos].known=1;
        for(int i=0;i<M;i++){
            if(G[temp.number][i]>0 && station[i].known==0 && station[i].distance>temp.distar
                station[i].distance=temp.distance+G[temp.number][i];
                station[i].path=temp.number;
                //Update the distance of this point if the condition is met
            }
        }

    }

}
```

```
int Find2ndMin(){
    struct Point temp = station0[M-1];
    int Result[M],Min=station0[M-1].distance,Min2=99999999;


    while(temp.number!=0){

        int TempDis=G[temp.path][temp.number];
        G[temp.path][temp.number]=0;
        //Removing an edge from the shortest path

        FindMin(station1);
        //Finding the shortest path in a new graph

        if(station1[M-1].distance<Min2 && station1[M-1].distance>Min){

            Min2=station1[M-1].distance;

            for(int i=0;i<M;i++){
                Result[i]=-1;
            }

            int i=M-1;
            while(station1[i].path!=-1){
                Result[i]=station1[i].path;
                i=station1[i].path;
            }
        }
        //If the condition is met, save this path as the second shortest path

        G[temp.path][temp.number]=TempDis;
        temp=station0[temp.path];
        //Restore the removed edge

    }

    int i=M-1,j=0;
    printf("%d ",Min2);

    while(i!=-1){
        output[j]=i+1;
        i=Result[i];
        j++;
    }

    for(int k=j-1;k>=0;k--){
```

```
        printf("%d ",output[k]);
    }
    //Output second shortest path

}
```

Appendix B: Test Case

Case1
5 6 1 2 50 2 3 100 2 4 150 3 4 130 3 5 70 4 5 40

Result1 240 1 2 4 5

Case2 3 3 1 2 1 1 3 3 2 3 1

Result2 3 1 3

Case3 4 4 1 2 100 2 4 200 2 3 250 3 4 100

Result3 450 1 2 3 4

Case4
5 6 1 2 1 1 3 2 1 4 4 2 3 1 3 4 1 4 5 1

Result4
5 1 4 5

Case5
5 6 1 2 1 1 3 3 1 4 4 2 3 1 3 4 1 4 5 1

Result5 5 1 4 5

Declaration

I hereby declare that all the work done in this project is of my independent effort.