浙江大學

ZHEJIANG UNIVERSITY

FDS 2025 春夏

project3

Transportation Hub

2025-05-04

Contents

1 Introduction

In this project, we need to write a program to find the transportation hubs of the shortest paths from start to end according to a given weighted graph.And this report will analyze the algorithm and performance of my program.

(1). In Chapter 2,I will introduce the the main algorithm of process used in the program.

(2). In Chapter 3,I present my test data,including input,output,and some analysis for the result.

(3). In Chapter 4,I conduct an analysis of my program.

(4). In Chapter 5,I include the complete code in the appendix.

2 Algorithm Specification

The following are the pseudocodes and explanations for main algorithm of process used in the program.

2.1 Dijkstra Algorithm

```
function Dijkstra(table, start, n):
dist ← array of size n, initialized to ∞
visited ← array of size n, initialized to false
dist[start] ← 0

for i from 0 to n-1:
    min_dist ← ∞
    temp ← -1
    for j from 0 to n-1:
        if not visited[j] and dist[j] < min_dist:
            min_dist ← dist[j]
            temp ← j

    visited[temp] ← true

    for j from 0 to n-1:
        if table[temp][j]   0 and dist[temp] + table[temp][j]   dist[j]:
            dist[j] ← dist[temp] + table[temp][j]

    return dist
```

In Dijkstra Algorithm,we mainly need to find the shortest path from a start node to any other node.I implied a greedy algorithm to achieve this and made it to find the shortest path.

2.2 Transportation hub check

```
function CheckTransportationHubs(table, start, end, n, dist, count, k):

dis_e ← array of size n
his_con ← array of size n

for i from 0 to n-1:
    dis_e[i] ← Dijkstra(i, end, table, n, count, 0, k)
    his_con[i] ← 0

for i from 0 to n-1:
    if i   start and i   end:
        con_o ← 0
        con_i ← 0
        for j from 0 to n-1:
            if table[i][j]   0 and dis_e[i] == dis_e[j] + table[i][j] and dis_e[i] + dist[i] == dist[end]:
                con_o ← con_o + 1
            else if table[i][j]   0 and dis_e[j] == dis_e[i] + table[i][j] and dis_e[j] + dist[j] == dist[end]:
                con_i ← con_i + 1
            if table[i][j]   0 and dis_e[j] == dis_e[i] + table[i][j] and count[j] == 1 and his_con[j] == 1:
                con_i ← k

        if con_o   k or con_i   k:
            his_con[i] ← con_o
            count[i] ← 1
```

Then we use Dijkstra Algorithm to find the min distance from all the node to the end of route.And we check

(1). The distance from start to node and node to end(compared to distance from start to end)-> If it is on a shortest path.

(2). The in and out situation for each node-> If it is a transportation hub.

(3). If the former node is a hub,and the current one is the only next node to choose in the shortest path.

3 Testing Data

To analyze the performance of the Transportation Hub found program , I try to employed different graph and testing path,and compare the output with expected answer.

3.1 Graph 1

```
10 16 2
1 2 1
1 3 1
1 4 2
2 4 1
2 5 2
3 4 1
3 0 1
4 5 1
4 6 2
5 6 1
7 3 2
7 8 1
7 0 3
8 9 1
9 0 2
0 6 2
```

This is the input format for the first graph,and I use graph editor to draw the graph as below:

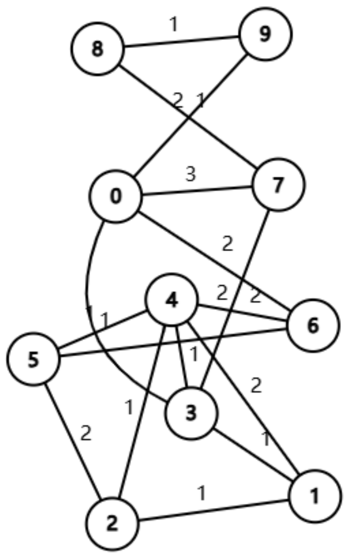

Figure 1: Graph 1-visualized

According to this graph,we input the following test cases;

```
3
1 6
7 0
5 5
```

For each case,I will analyze in detail:

(1). For 1 6,the shortest paths are as below

(a) 1->2->4->6

(b) 1->2->5->6

(c) 1->3->4->6

(d) 1->3->4->->5->6

(e) 1->4->5

Therefore the transportation hub should be 2 3 4 5,and the true output is :

```
2 3 4 5
```

which is correct.

(2). For 7 0,the shortest paths are as below

(a) 0->7

(b) 0->3->7

Therefore the transportation hub should be 2 3 4 5,and the true output is :

```
None
```

which is correct.

(3). For 5 5,there is no shortest path,so the output should be None,,and the true output is :

```
None
```

which is correct.

3.2 Graph 2

```
12 21 2
0 1 1
0 2 1
0 3 2
1 3 1
1 4 2
2 3 1
2 5 1
3 4 1
3 5 2
3 6 1
4 5 1
4 7 2
5 6 1
5 8 2
6 7 1
6 8 1
7 8 1
7 9 2
8 9 1
9 10 1
10 11 1
```

This is the input format for the first graph,and I use graph editor to draw the graph as below:

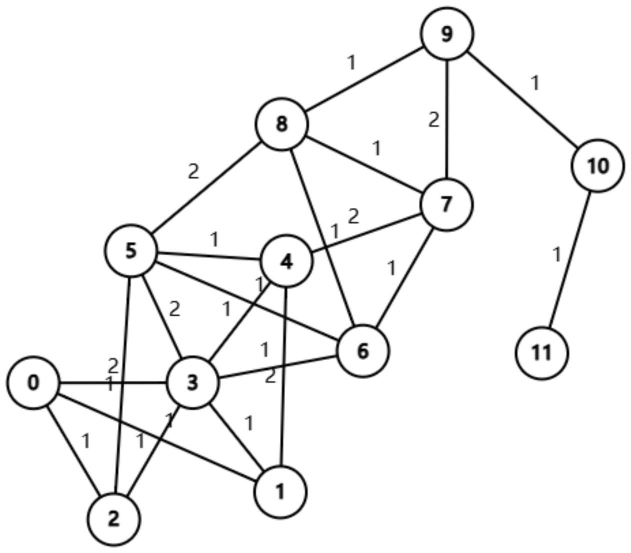

Figure 2: Graph 1-visualized

According to this graph,we input the following test cases;

```
3
0 10
7 11
2 8
```

For each case,I will analyze in detail:

(1). For 0 10,the shortest paths are as below

(a) 0->1->3->6->8->9->10

(b) 0->2->5->6->8->9->10

(c) 0->2->5->8->9->10

(d) 0->3->6->8->9->10

Therefore the transportation hub should be 2 3 5 6 8 9,and the true output is :

```
2 3 5 6 8 9
```

which is correct.

(2). For 7 11,the shortest paths are as below

(a) 7->9->10->11

(b) 7->8->9->10->11

Therefore the transportation hub should be 9 10,and the true output is :

```
9 10
```

which is correct.

(3). For 2 8,the shortest paths are as below

(a) 2->5->8

(b) 2->5->6->8

(c) 2->3->6->8

Therefore the transportation hub should be 5 6,and the true output is :

```
5 6
```

which is correct.

4 Analysis and Comments

From those cases above I can conclude that the program has made it to find the transportation hubs of a given weighted graph,now I'd like to further analyze the time and space complexity of the program.

4.1 Time complexity

For the first step of reading,it will take $O(M)$ time.And in every find function,it needs to go through every node and use dijkstra function to calculate the distance,this will take $O(N \times N^2) = O(N^3)$time.Therefore the total time complexity of the program is $O(N^3 + M)$.

4.2 Space complexity

I use a two-dimensional array table to save the data of vertexes and edges,which takes $O(N)$ space.And in every find function,I will use Dijkstra function for n times,and each time will create several arrays with the size of n,so the total space is $O(N)$.Therefore the total space complexity of the program is $O(N^2)$.

4.3 Comment

The program performs quite well in the target of finding transportation hubs.But just as the analysis implies,the program takes a quite huge time and space complexity.This will cost a lot of time and space waste when N is quite large.So there's still need further optimizations to analyze large graphs

5 Appendix

```c
# include<stdio.h>
# include<stdlib.h>

void find(int start,int end,int k,int** table,int n);
int djkstra(int start,int end,int** table,int n,int* count,int flag,int k
   );

int main(void){
    int n,m,k;                                //initialize n,m,k to save
        the para of graph
    scanf("%d %d %d",&n,&m,&k);               //scan the input
    int** table = (int**)malloc(n*sizeof(int*)); //create the table for
        the graph
    for(int i = 0;i < n;i++){
        table[i] = (int*)malloc(n*sizeof(int));
        for(int j = 0;j < n;j++){
            table[i][j] = 0;                  //initalize the graph,mark
                the unconnected as distance of 0
        }
    }
    for(int i = 0;i < m;i++){
        int c1,c2,dis;
        scanf("%d %d %d",&c1,&c2,&dis);        //scan the input of the vertexes and edges
        table[c1][c2] = dis;                   //save the edge information
        table[c2][c1] = dis;
    }
    int num_of_test;                           //scan the test cases
    scanf("%d",&num_of_test);
    for(int i = 0;i < num_of_test;i++){
        int c1,c2;
        scanf("%d %d",&c1,&c2);                //scan the start and end for the test case
        find(c1,c2,k,table,n);                 //find the solution for transportation hub
    }
    free(table);                               //free the space
    return 0;
}
void find(int start,int end,int k,int** table,int n){//function to find the transportation hub
    int* count = (int*)malloc(n*sizeof(int)); //create count as mark of the transportaion hub
    for(int i = 0;i < n;i++){
        count[i] = 0;                          //initialize count as 0
    }
    int min_dis = djkstra(start,end,table,n,count,1,k);//use the djkstra algorithm to find the minimal distance
    // printf("%d\n",min_dis);
    int check = 0;                             //mark if there is any transportaion hub
    for(int i = 0;i < n;i++){
        if(count[i] == 1){                     //if count[i] is marked as 1,then it is a transportaion hub
            printf("%d ",i);                   //print out transportaion hub
            check = 1;                         //mark there exist at least 1 hub
        }
    }
    if(check == 0){                                //hub not found,print "None
        "
        printf("None");
    }
    printf("\n");
    free(count);                                   //free count
}


int djkstra(int start,int end,int** table,int n,int* count,int flag,int k
    ){//function to calculate the shortest distance from start to end
    int* dist = (int*)malloc(n*sizeof(int));  //create dist to save the
        shortest distance from start to i node
    int* visted = (int*)malloc(n*sizeof(int)); //create visited to mark
        whether a node is visited in djkstra
    for(int i = 0;i < n;i++){
        dist[i] = 1000000000;                      //initialize dist big
            enough
        visted[i] = 0;                             //initialize all node
            unvisited
    }
    dist[start] = 0;                               //initialize start as
        distance 0
    for(int i = 0;i < n;i++){
        int temp;                                  //create temp for finding
            nodes
        int min_dist = 1000000000;                 //initialize min_dist big
            enough
        for(int j = 0;j < n;j++){
            if(!visted[j] && dist[j] < min_dist){ //find the unvisted node
                 with shortest distance
                min_dist = dist[j];                //mark min_dist for search
                temp = j;
            }
        }
        visted[temp] = 1;                          //mark the found node as
            visited
        for(int j = 0;j < n;j++){
            if(table[temp][j] != 0 && dist[temp] + table[temp][j] <= dist[j]){//go through all node connected to the temp node
                dist[j] = dist[temp] + table[temp][j]; //if the path temp->j is shorter than previous found path or j is not connected to found node yet
            }
        }
    }
    if(flag){                                       //flag to mark if the function is to simply calculate the distance or find the hubs
        int* dis_e = (int*)malloc(n*sizeof(int)); //create dis_e to save the distance from i to end
        int* his_con = (int*)malloc(n*sizeof(int)); //to mark every node's out data(in shortest path)
        for(int i = 0;i < n;i++){
            dis_e[i] = djkstra(i,end,table,n,count,0,k);//initialize the dis_e by using flag = 0 mode of function djkstra
            his_con[i] = 0;                         //initialize out data as 0
        }

        for(int i = 0;i < n;i++){                   //check every node except start and end
            if(i != start && i != end){
                int con_o = 0;                      //count out data
                int con_i = 0;                      //count in data
                for(int j = 0;j < n;j++){           //table[i][j] != 0 indicate that there is an edge between i and j
                    if(table[i][j] != 0 && dis_e[i] == dis_e[j] + table[i][j] && dis_e[i] + dist[i] == dist[end]){
                        con_o++;                    //dis_e[i] == dis_e[j] + table[i][j] indicate that j is the next node in shortest path contains i
                    }else if(table[i][j] != 0 && dis_e[j] == dis_e[i] + table[i][j] && dis_e[j] + dist[j] == dist[end]){
                        con_i++;                    //dis_e[j] == dis_e[i] + table[i][j] indicate that i is the next node in shortest path contains j
                    }                               //dis_e[i] + dist[i] ==
                        dist[end] indicate that i is on a shortest path
                    if(table[i][j] != 0 &&dis_e[j] == dis_e[i] + table[i][j] && count[j] == 1 && his_con[j] == 1){
                        con_i = k;              //count[j] == 1 && his_con[j] == 1 indicate that j is a transportaion hub and j is the only possible next node for i
                    }
                }
                if(con_o >= k || con_i >= k){
                    his_con[i] = con_o;         //mark for the con_o = 1 situation
                    count[i] = 1;               //mark i as transportation hub
                }
            }
        }
    }
    return dist[end];                           //return the min_dist from start to end
}
```

6 Declaration

I hereby declare that all the work done in this project titled “Project 3 : Transportation Hub ” is of my independent effort.