**Normal Ambulance Dispatch**

**Date:2021-11-28**

**Chapter 1: Introduction**

After the study of graph, queue, and all other basic algorithms, we are here allow to solve the problem of ambulance dispatch.

If there is a map of the city, all ambulance dispatch centers and all pick-up points are marked. Your mission is to write a program to deal with all kinds of emergencies. Suppose the caller is waiting at a location. You must inform the nearest dispatch center, which must have at least one ambulance available.

# Chapter 2: Algorithm Specification

Use Dijkstra algorithm to calculate the shortest path (shortest time) between hospitals and other vertices, and save the path between hospitals and other vertices.

When the top point v sends a rescue request, for the rescue center, find the path with the least number of streets from the top point v to the center. Then compare Na ambulance center with the ambulance center that meets the meaning of the problem (if the time is the least, find the ambulance center with the least time; if the time is the same, find the ambulance center with more vehicles; if the time is the least and the number of ambulances is the same, find the ambulance center with the least number of streets). Of course, only one ambulance center is considered.

Use pre array save the points that involved in the path of each calling, and then use DFS to traverse and update to find the corresponding path.

In addition, one thing to note in this problem is that the rescue point is from A-1 to A10, which means that the processing of two numbers should be considered when inputting data. In addition, when outputting data, it should be noted that rescue points may be accessed in order to output A-I (0 < = I < = Na) and other data. In short, I should pay special attention to input and output.

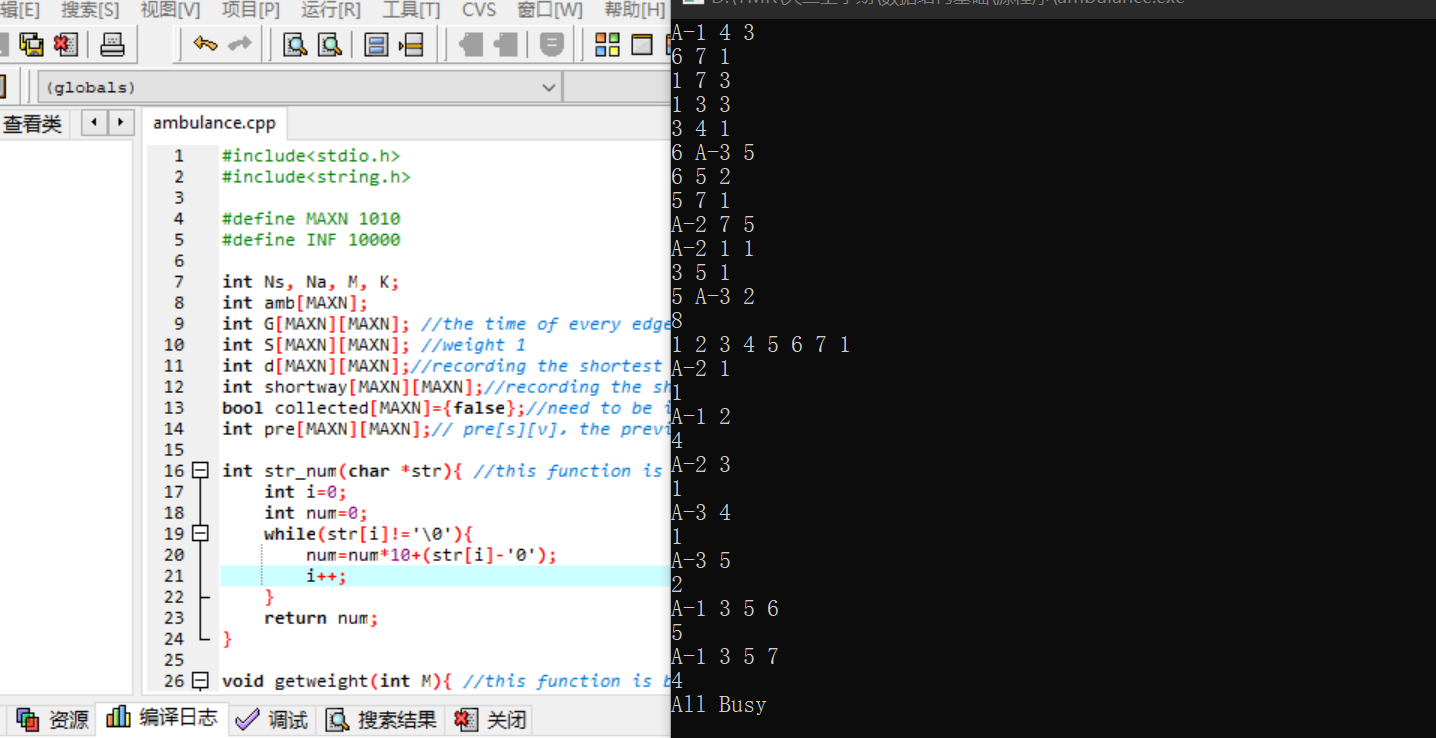
Finally we use the structure of queue to output the path of each calling. Besides, we need to write some functions to deal with the input data, such as transform the ‘A-1’ into the first hospital. And we also need to output with the same rules.

# Chapter 3: Testing Results

As you can see the program perfectly display the results, the input are from the PTA, and the output well match to the standard output.

Besides my program also considered the situation that the path need to pass by the dispatch center.

Here is another test result.



# Chapter 4: Analysis and Comments

The main cost of time and space came from the Dijkstra algorithm.

T=O(V2+E); For you need to search the minimum dist for each start point, this was in a for loop, which is the basic time complexity of Dijkstra algorithm.

S=O(V+E); For you need to store the graph, and the main cost came from the edges and nodes with some coefficient. So the space complexity is a linear relationship to V and E.

# Appendix: Source Code (in C)

#include<stdio.h>

#include<string.h>

#define MAXN 1010

#define INF 10000

int Ns, Na, M, K;

int amb[MAXN];

int G[MAXN][MAXN]; //the time of every edge

int S[MAXN][MAXN]; //weight 1

int d[MAXN][MAXN];//recording the shortest time(dist)

int shortway[MAXN][MAXN];//recording the shortest way

bool collected[MAXN]={false};//need to be initialized

int pre[MAXN][MAXN];// pre[s][v]，the previous vector of v when the source is s

int str\_num(char \*str){ //this function is built to transform the string into a integer

int i=0;

int num=0;

while(str[i]!='\0'){

num=num\*10+(str[i]-'0');

i++;

}

return num;

}

void getweight(int M){ //this function is built to establish a graph

for(int i=0;i<=Na+Ns;i++){

for(int j=0;j<=Na+Ns;j++){

G[i][j]=INF;

}

}

for(int i=0; i<M; i++){

char str[10];

int u,v;

scanf("%s",str);

if(str[0]=='A'){ //use sscanf to get the proper char and transform into an integer

sscanf(str, "A-%d", &u);

u+=Ns;

}

else{

u = str\_num(str);

}

scanf("%s",str);

if(str[0]=='A'){

sscanf(str, "A-%d", &v);

v+=Ns;

}

else{

v = str\_num(str);

}

scanf("%d",&G[u][v]);

G[v][u] = G[u][v];

S[u][v] = S[v][u] = 1;

}

}

void Dijkstra(int s) //use dijkstra algorithm to find out the shortest way

{

collected[s]=true;

for(int i=0;i<=Ns+Na;i++){ //initialize the dist near by the source

if(G[s][i]!=0){

d[s][i]=G[s][i];

//printf("%d-%d=%d\n", s, i, d[s][i]); //check if initialized successfully

pre[s][i]=s;

shortway[s][i]=1;

}

else{

d[s][i]=INF;

shortway[s][i]=INF;

}

}

d[s][s]=shortway[s][s]=0;

pre[s][s] = -1;

while(1){

int u=-1, min=INF;

for(int j=1; j<=Ns+Na; j++){

if(collected[j]==false&&d[s][j]<min){

u = j;

min = d[s][j];

}

}

if(u==-1) break;

collected[u]=true;

for(int v=1; v<=Ns+Na; v++){

if(collected[v]==false&&G[u][v]!=INF){

if((d[s][u]+G[u][v])<d[s][v]){

d[s][v] = d[s][u]+G[u][v];

shortway[s][v] = shortway[s][u]+S[u][v];

pre[s][v] = u;

}

else if(((d[s][u]+G[u][v])==d[s][v])&&((shortway[s][u]+S[u][v])<=shortway[s][v])){

shortway[s][v] = shortway[s][u]+S[u][v];

pre[s][v] = u;

}

}

}

}

}

int main()

{

scanf("%d%d", &Ns, &Na);

for(int i=Ns+1; i<=Ns+Na; i++){

scanf("%d",&amb[i]);

}

scanf("%d",&M);

getweight(M);

/\*for(int i=0;i<=Ns+Na;i++){ //check if the graph was initialized successfully

for(int j=0;j<=Ns+Na;j++){

printf("%d-%d=%d \n",i ,j , G[i][j]);

}

}\*/

/\*for(int i=0;i<=Ns+Na;i++){ //check if the graph was initialized successfully

for(int j=0;j<=Ns+Na;j++){

printf("%d-%d=%d \n",i ,j , S[i][j]);

}

}\*/

for(int i=Ns+1; i<=Ns+Na; i++){ //apply the dijkstra to every hospital

Dijkstra(i);

for(int j=0;j<=Ns+Na; j++){

collected[j]=false;

}

}

scanf("%d",&K);

for(int i=0; i<K; i++){

int s;

scanf("%d",&s);

int minTime=INF, maxAmbu=-1, minStreets=INF, index=-1;

for(int j=Ns+1; j<=Ns+Na; j++){

if(amb[j]>0){

if(d[j][s]<minTime){ //first compare the mintime

index = j;

minTime = d[j][s];

maxAmbu = amb[j];

minStreets = shortway[j][s];

}

else if(d[j][s]==minTime&&amb[j]>maxAmbu){ //then compare the number of ambulance

index = j;

maxAmbu = amb[j];

minStreets = shortway[j][s];

}

else if(d[j][s]==minTime&&amb[j]==maxAmbu&&shortway[j][s]<minStreets){ //finally compare the minstreet

index = j;

minStreets = shortway[j][s];

}

}

}

if(index==-1) printf("All Busy\n");

else{

amb[index]--; //use queue to output the results

int queue[MAXN];

int k=0;

while(s!=-1){

queue[k++]=s;

s=pre[index][s];

}

k--;

printf("A-%d",queue[k]-Ns);

k--;

while(k!=-1){

if(queue[k]>Ns) printf(" A-%d",queue[k]-Ns); //its possible to meet hospital when output the rout

else printf(" %d",queue[k]);

k--;

}

printf("\n");

printf("%d\n",minTime);

}

}

return 0;

}

# Declaration

***I hereby declare that all the work done in this project titled "*Normal Ambulance Dispatch*" is of my independent effort.***