Ambulance Dispatch

Date: November 19th,2021

author:韩恺荣

Chapter One: Introduction

In this project, we try to solve how to dispatch ambulance reasonably, in order to optimize the distribution.

In the theory of graph, we can represent cities by vertex. Under this background, Given the map of a city, with all the ambulance dispatch centers and all the pick-up spots marked. We are able to write a program to process the emergency calls using tools and concept in graph. And the work done below is the steps i try to complement this function.

To verify the cases, some explanation are as follow. To begin with, There are two integers, Na and Ns, representing the number of pick-up spots and hospitals respectively.And edges, also considered as streets connecting spots and hospitals, are given with distance. Every hospital have a certain number of ambulances and can dispatch ambulances only if ambulance' s number is bigger than zero. When a call is coming, we should action to the call. First, we select nearest hospital, which remain has ambulances. If there exist one more cases, we consider number of ambulances. The hospital have more ambulances should dispatch with high priority. Finally, if there remain exists more than one cases, we take streets'

number we crossed into consideration. We select a route crossing less streets.

one possible input is as follow(comments is in parenthesis):

2(number of spots) 2(number of hospital)

1(ambulances in first hospital) 3(ambulances in second)

4(number of edges)

A-1 1 2(A-1 is connected to 1 and distance is 2)

1 A-2 3(same meaning as above)

A-2 2 4(same meaning as above)

2 A-1 1(same meaning as above)

3(calls number)

1 1 2(spots 1 1 2 are called for help as follow)

and output corresponding of this input is:

A-1 1(route)

2(distance)

A-2 1

3

A-2 2

4

Chapter 2: Algorithm Specification

function 1: main function

```c
int main(){
    int i,j;
    scanf("%d %d",&Ns,&Na);//input the number of spots and hospitals
    for(i=1;i<=Na;i++){
        H[i] = (struct hospital*)malloc(sizeof(struct hospital));//apply for space
        scanf("%d",&H[i]->Am);//number of ambulances
    }
    scanf("%d",&edge);//number of edges
    Init_Adj();//initial the adj matrixm, inwhich adj[i][i] is 0 and others are INF
    for(i=1;i<=edge;i++){
        InputRoute();//input route
    }
    for(i=1;i<=Na;i++)Dij(i);//Dijkstra for every hosptial,record path message and distance message
    scanf("%d",&call);//number of calls
    for(i=1;i<=call;i++){
        scanf("%d",&k);
        Print(k);//arange every route for every call
    }
}
```

By using module design ways, I break question into pieces. We will deal with input and output in turn.

First, we read in Na , Ns and Am, which are integers representing number of spots and hospital and number of ambulances in every hospital respectively. Then read in streets' number and create a graph by function "InputRoute". Next step is to use Disjkstra algorithm for every hospital. Finally, we output result according Disjkstra algorithm.

function 2: Init_Adj()

```c
void Init_Adj(){//initial the adj matrixm, in which adj[i][i] is 0 and others are INF
    int i,j;
    for(i=1;i<=Na+Ns;i++){
        for(j=1;j<=Na+Ns;j++){
            if(i==j){
                Adj[i][j]=0;//the distance from a verterx to itself is zero
            }
            else {
                Adj[i][j]=INF;//the distance from a verterx to others are infinite
            }
        }
    }
}
```

All the useful messages are store in adjacency matrix, and initial it is essential. We assign zero to Adj[i][i],and assign INF, which means distance is infinite, to other vertexes.

function3: InputRoute()

```
void InputRoute(){
    int ve1,ve2,rank;
    char ch[10],ch1[10];
    scanf("%s",ch);//input as a string
    scanf("%s",ch1);
    scanf("%d",&rank);//the rank of every edge
    if(ch[0]!='A'){//when inputting is not a hospital
        ve1 = TurnToInt(ch);//convert string to integer
    }else{
        ve1 = TurnToInt(ch+2)+Ns;
    }
    if(ch1[0]!='A'){//when inputting is not a hospital
        ve2 = TurnToInt(ch1);//convert string to integer
    }else{
        ve2 = TurnToInt(ch1+2)+Ns;
    }
    Adj[ve1][ve2] = rank;//assign to the adjacency matrix
    Adj[ve2][ve1] = rank;//assign to the adjacency matrix
}
```

We read data as strings and invert strings to integer. Then we assign to adjacency matrix. After executing this function, we can get a graph represented by adjacency matrix.

function 4:TurnToInt(char *ch)

```
int TurnToInt(char *ch){//convert string to integer
    int sum=0,i=0;
    while(ch[i]!='\0'){//stop when read '\0'
        sum = sum*10 + ch[i]+0-'0';
        i++;
    }
    return sum; //return sum
}
```

This function is to convert a string to integer by a simple loop.

function 5: CopyAdj()

```
void CopyAdj(){//copy adjacency matrix
    int i,j;
    for(i=1;i<=Na+Ns;i++){
        for(j=1;j<=Na+Ns;j++){
            Adj_f[i][j]=Adj[i][j];//assign to every element
        }
    }
}
```

Using this function is to copy a matrix same as adjacency matrix.

function 6:Dij(int num)

```
void Dij(int num){//Dijkstra algorithm
    int j,i,ve;
    CopyAdj();//copy matrix
    for(i=1;i<=Na+Ns;i++){//initial hospital's message
        H[num]->dist[i]=Adj_f[num+Ns][i];
        if(H[num]->dist[i]<INF)//discuss different cases
            H[num]->path[i] = num+Ns;
        else
            H[num]->path[i] = -1;
        H[num]->know[i]=0;//when not be collected, assign 0
        H[num]->street[i]=0;//when cross streets are unnessary
    }
    while(1){
        ve = FindMin(H[num]->dist,H[num]->know);//find minterm in dist[]
        if(ve==ERROR) break;//if the graph is all scaned
        H[num]->dist[ve]=Adj_f[num+Ns][ve];//assign the value to dist[]
        H[num]->know[ve] = 1;//ve is what we find as a nearest verterx
        for(i=1;i<=Na+Ns;i++){
            if(i != num+Ns){//except for hospital itself
                if(H[num]->know[i]==0&&Adj_f[ve][i]<INF){//if it's not be dealt
                    if(Adj_f[num+Ns][i]>Adj_f[num+Ns][ve]+Adj_f[ve][i])//update distance's message
                    {
                        Adj_f[num+Ns][i] = Adj_f[num+Ns][ve]+Adj_f[ve][i];
                        H[num]->dist[i] =  Adj_f[num+Ns][ve]+Adj_f[ve][i];//memorize dist message
                        H[num]->path[i] = ve;//memorize the path message
                        H[num]->street[i] = H[num]->street[ve]+1;//memorize street message
                    }
                }
            }
        }
    }
}
```

This function is a version of Dijkstra algorithm.

The brief steps of this algorithm is to define two arrays, which considered as collected and distance.

For every loop, we find min distance of uncollected elements and memorize index. Then we update the distance for every uncollected vertex by index. Finally, we reset collected array and put index into it. After [V] times loop, we can find min distance.

In my function, we update streets' number and path at same time.

function 7:Print(int call)

```
void Print(int call1){//print result for every call
    int min_dist=INF;
    int min_street=INF;
    int max_Am=-1;
    int index=-1;
    int i,j,k;
    for(i=1;i<=Na;i++){//find minmum distance at first
        if(min_dist>H[i]->dist[call1]&&H[i]->Am>0){//ensure this hospital has ambulance
            index = i;//record index
            min_dist = H[i]->dist[call1];//update distance message
            min_street = H[i]->street[call1];//update street message =
            max_Am = H[i]->Am;//update ambulance message
        }
    }
    if(index == -1){//if not be found, index is -1 and it means All busy
        printf("\nAll Busy");
        return;
    }
    for(i=1;i<=Na;i++){//keep  ambulance number in mind,and update result
        if(min_dist==H[i]->dist[call1]&&max_Am<H[i]->Am&&H[i]->Am>0){
            index = i;
            min_street =  H[i]->street[call1];
            max_Am = H[i]->Am;
        }
    }
```

```
    for(i=1;i<=Na;i++){//keep street in mind,and update result
        if(min_dist==H[i]->dist[call1]&&max_Am==H[i]->Am&&H[i]->Am>0){
            if(min_street>H[i]->street[call1]){
                index = i;
                min_street = H[i]->street[call1];
            }
        }
    }
    (H[index]->Am)--;//reduce ambulance after every call
    Outputpath(index,call1);//output result
    printf("\n%d",H[index]->dist[call1]);//output distance
}
```

’ This part is to print result of every call. And distance is considered first and ambulance number street crossed are considered in turn.

function 10: OutputPath(int index,int call)

```
void Outputpath(int index,int call1){
    int stack[MAXSIZE];//define a stack to inverse the result
    int top=0;//stack's top
    if((flag++)==0)printf("A-%d",index);//adjust format of outputting
    else printf("\nA-%d",index);
    int i;
    for(i=call1;i!=index+Ns;i=H[index]->path[i])stack[top++]=i;//push into stack one by one
    for(i=top-1;i>=0;i--){//print result on the screen
        if(stack[i]>Ns)
        printf(" A-%d",stack[i]-Ns);
        else
        printf(" %d",stack[i]);
    }
}
```

We find a optimal index in advance steps. So we will print path in order at this part. To reverse the result, we use a stack to implement it.

summary: In my code, there are two key thoughts:

1) Dijkstra for every hospital rather than Dijkstra for spots ,and we can rearrange route according the data we get

from Dijkstra. They are path data, ambulance number, street number and distance.

2) We should keep in mind that if a hospital' s ambulance are used out. It will convert to a normal vertex. So the output may have "A-xxx" .For example, maybe one output is "A-3 3 1 2 A-2 6", which is a route from A-3 to 6.

Data Structure:

(1) stack:

it is implemented by array. And it is designed to reverse the result:

code is as follow:

```
int stack[MAXSIZE];//define a stack to inverse the result
int top=0;//stack's top
if((flag++)==0)printf("A-%d",index);//adjust format of outputting
else printf("\nA-%d",index);
int i;
for(i=call1;i!=index+Ns;i=H[index]->path[i])stack[top++]=i;//push into stack one by one
for(i=top-1;i>=0;i--){//print result on the screen
    if(stack[i]>Ns)
    printf(" A-%d",stack[i]-Ns);
    else
    printf(" %d",stack[i]);
}
```

(2) structure data:

To simplify the question, i defined a structure hospital to store data of a hospital.

```
struct hospital{
    int path[MAXSIZE];//
    int Am;           //
    int dist[MAXSIZE];//
    int know[MAXSIZE];//
    int street[MAXSIZE];
}*H[100];
```

(3) Two-dimension array:

adjacency matrix is stores as a Two-dimension array

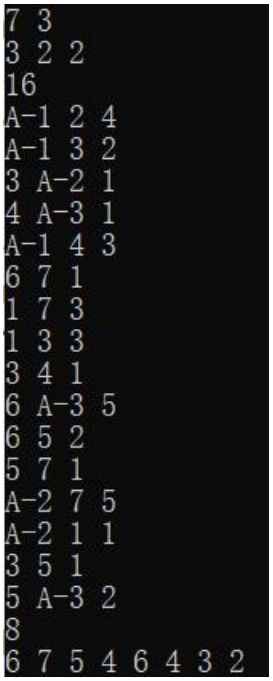
Chapter 3: Testing Results

There are four main possibilities in the project:

(1) there do not exist any other strength cases

For example:

case1:Input:



output:

```
A-3 5 6
4
A-2 3 5 7
3
A-3 5
2
A-2 3 4
2
A-1 3 5 6
5
A-1 4
3
A-1 3
2
All Busy
```

case 2:input:

```
2 2
1 3
4
A-1 1 2
1 A-2 3
A-2 2 4
2 A-1 1
3
1 1 2
```

output:

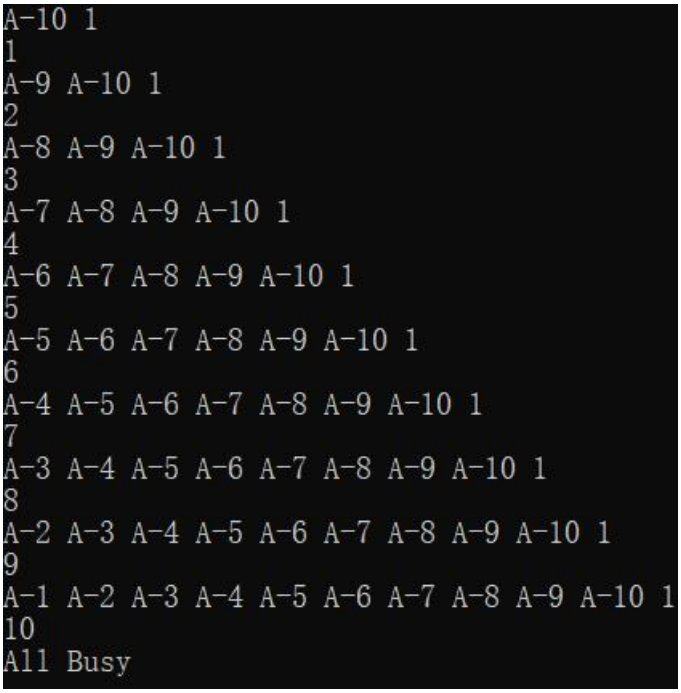
```
A-1 1
2
A-2 1
3
A-2 2
4
```

(2) When the output have other hospitals

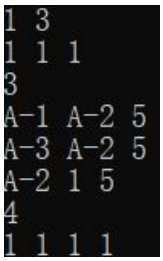
case 1:input:

```
1 10
1 1 1 1 1 1 1 1 1 1
10
A-1 A-2 1
A-2 A-3 1
A-3 A-4 1
A-4 A-5 1
A-5 A-6 1
A-6 A-7 1
A-7 A-8 1
A-8 A-9 1
A-9 A-10 1
A-10 1 1
11
1 1 1 1 1 1 1 1 1 1 1
```

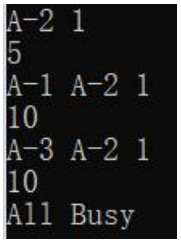
output:



case 2:input:

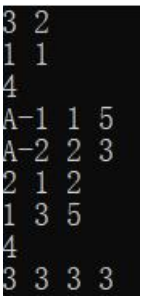


output:

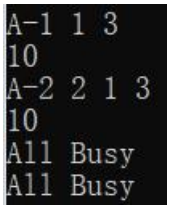


(3) when we should make a decision according to the streets' number:We choose a hospital which will cross less streets.

case:input :

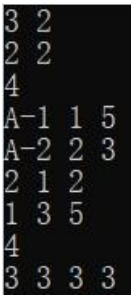


output:

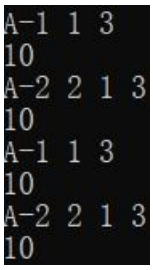


(4) When we should make a decision according to the ambulances' number:We choose a hospital which has more ambulances.

case:input:



output:



(5) When the inputting scale is very large:

case:input:

1000 9

100 100 100 100 100 100 100 100 100

1008

1 2 1

2 3 1

3 4 1

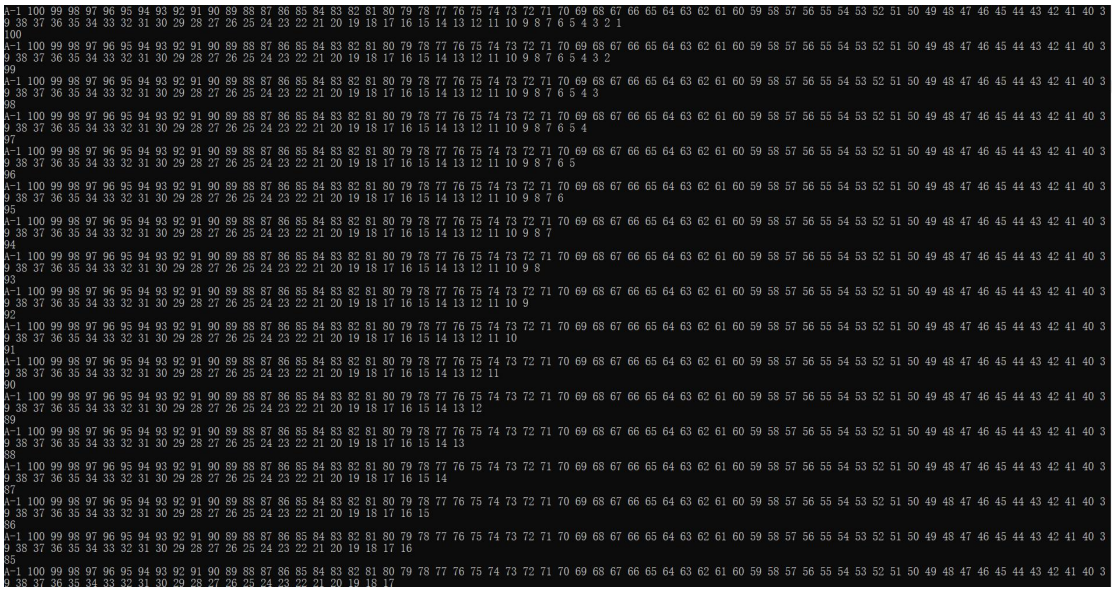
(……We omit part input, and the format is i i+1 1,and if i is the times of 100, we input A-i/100 i+1 1,until i=999)

999 1000 1

1000

1 2 …… (omit most) 999 1000

output:



(…… omit most)

```
5
A-9 900 899 898 897
4
A-9 900 899 898
3
A-9 900 899
2
A-9 900
1
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
All Busy
```

(………… omit other lines “All busy”)

And I will leave this test data in my folder “ReadMe.txt”,you can copy it and run it by yourself directly.

Chapter 4: Analysis and Comments

(1) Analyse and comments for the algorithm:

a) Time Complexity:

The Time Complexity depends on Dijkstra algorithm and Print function.

We use “n” to represent the Ns we input. Apparently,T(n) = max{T(Dij),T(Print)}. First, we consider T(Dij). In Dijkstra algorithm, for every call of this function, we scan all the vertexes , and should calculate O(n^2) times to loop and update every vertex’s distance in the graph. And for every hospital, we call this function, so time complexity becomes

O(k*n^2),become hospitals number is smaller than ten, so T(Dij) = O(n^2).

Next, we consider T(Print). Three times of scanning all the vertexes cost O(3*n).And to output a path, we scan all vertex again, which cost O(n).Therefore, T(Print) = O(3*n)+O(n) = O(n).

Above all, time complexity of the code is max{T(Dij),T(Print)},which is O(n^2).T(n) = O(n^2),in which n is the Ns we input.

b) Space Complexity:

The space complexity of the whole process is determined by adjacency matrix and stack as follow:

```
int Adj[MAXSIZE][MAXSIZE];
int Adj_f[MAXSIZE][MAXSIZE]

int stack[MAXSIZE];

struct hospital{
    int path[MAXSIZE];
    int Am;
    int dist[MAXSIZE];
    int know[MAXSIZE];
    int street[MAXSIZE]
}*H[100];
```

It means we use 5*MAXSIZE+2*MAXSIZE*MAXSIZE memory space to store int value.It has no relationship with n, so S(n)=O(1).

we only use constant memory space of the computer when we execute the programmer.

c)Comments:

For this algorithm, I think both time complexity and space complexity is accepted. If we must optimize the algorithm, we can use min heap to store distance data ,and time complexity of search min distance can be reduced to O(n*log n).

Appendix: Source Code(in C)

```
#include<stdio.h>
#include<stdlib.h>
#define INF 111111
#define MAXSIZE 10000
#define ERROR -1

struct hospital{
    int path[MAXSIZE];//memorize the path for each verterx's father node
    int Am;             //memorize the number of ambulances in this hospital
    int dist[MAXSIZE];//memorize the distance from hospital to verterx
```

```
        int know[MAXSIZE];//memorize the node which is alrady be collected
        int street[MAXSIZE];//memorize the the number of streets that we must pass to reach the verterx
    }*H[100];

    int Adj[MAXSIZE][MAXSIZE];//adjacency matrix
    int Adj_f[MAXSIZE][MAXSIZE];//auxiliary adjacency matrix
    int Na,Ns,edge,k,call,flag;//Na is the number of hospital,Ns is spots,edge is edge,flag is to adjust the format of outputing

    void Init_Adj();//initial the adjacency matrix
    void InputRoute();//input the edge and construct the graph
    void Print(int k);//print the final consequence
    void CopyAdj();//copy to assign to matrix "Adj_f"
    int TurnToInt(char *ch);//Turn a char to integer
    void Dij(int num);//Dijkstra algorithm
    int FindMin(int dist[],int know[]);//Find a min term of distance for a verterx
```

```
    void Outputpath(int index,int call1);//output
consequence


    int main(){
        int i,j;
        scanf("%d %d",&Ns,&Na);//input the number of
spots and hospitals
        for(i=1;i<=Na;i++){
            H[i] = (struct hospital*)malloc(sizeof(struct
hospital));//apply for space
            scanf("%d",&H[i]->Am);//number of ambulances
        }
        scanf("%d",&edge);//number of edges
        Init_Adj();//initial the adj matrixm, inwhich adj[i][i] is
0 and others are INF
        for(i=1;i<=edge;i++){
            InputRoute();//input route
        }
        for(i=1;i<=Na;i++)Dij(i);//Dijkstra for every
hosptial,record path message and distance message
        scanf("%d",&call);//number of calls
        for(i=1;i<=call;i++){
```

```
            scanf("%d",&k);
            Print(k);//arange every route for every call
        }
    }
    void Init_Adj(){//initial the adj matrixm, in which adj[i][i]
is 0 and others are INF
        int i,j;
        for(i=1;i<=Na+Ns;i++){
            for(j=1;j<=Na+Ns;j++){
                if(i==j){
                    Adj[i][j]=0;//the distance from a verterx to
itself is zero
                }
                else {
                    Adj[i][j]=INF;//the distance from a verterx
to others are infinite
                }
            }
        }
    }
    void InputRoute(){
        int ve1,ve2,rank;
```

```
        char ch[10],ch1[10];
        scanf("%s",ch);//input as a string
        scanf("%s",ch1);
        scanf("%d",&rank);//the rank of every edge
        if(ch[0]!='A'){//when inputting is not a hospital
            ve1 = TurnToInt(ch);//convert string to integer
        }else{
            ve1 = TurnToInt(ch+2)+Ns;
        }
        if(ch1[0]!='A'){//when inputting is not a hospital
            ve2 = TurnToInt(ch1);//convert string to integer
        }else{
            ve2 = TurnToInt(ch1+2)+Ns;
        }
        Adj[ve1][ve2] = rank;//assign to the adjacency
matrix
        Adj[ve2][ve1] = rank;//assign to the adjacency
matrix
    }
    int TurnToInt(char *ch){//convert string to integer
        int sum=0,i=0;
        while(ch[i]!='\0'){//stop when read '\0'
```

```
        sum = sum*10 + ch[i]+0-'0';
        i++;
    }
    return sum;//return sum
}
void CopyAdj(){//copy adjacency matrix
    int i,j;
    for(i=1;i<=Na+Ns;i++){
        for(j=1;j<=Na+Ns;j++){
            Adj_f[i][j]=Adj[i][j];//assign to every element
        }
    }
}
void Dij(int num){//Dijkstra algorithm
    int j,i,ve;
    CopyAdj();//copy matrix
    for(i=1;i<=Na+Ns;i++){//initial hospital's message
        H[num]->dist[i]=Adj_f[num+Ns][i];
        if(H[num]->dist[i]<INF)//discuss different cases
            H[num]->path[i] = num+Ns;
        else
            H[num]->path[i] = -1;
```

```
            H[num]->know[i]=0;//when not be collected,
assign 0
            H[num]->street[i]=0;//when cross streets are
unnessary
        }
        while(1){
            ve =
FindMin(H[num]->dist,H[num]->know);//find minterm in
dist[]
            if(ve==ERROR) break;//if the graph is all scaned
            H[num]->dist[ve]=Adj_f[num+Ns][ve];//assign
the value to dist[]
            H[num]->know[ve] = 1;//ve is what we find as a
nearest verterx
            for(i=1;i<=Na+Ns;i++){
                if(i != num+Ns){//except for hospital itself

    if(H[num]->know[i]==0&&Adj_f[ve][i]<INF){//if it's not
be dealt

    if(Adj_f[num+Ns][i]>Adj_f[num+Ns][ve]+Adj_f[ve][i])//
update distance's message
```

```
                    {
                        Adj_f[num+Ns][i] =
Adj_f[num+Ns][ve]+Adj_f[ve][i];
                        H[num]->dist[i] =
Adj_f[num+Ns][ve]+Adj_f[ve][i];//memorize dist message
                        H[num]->path[i] = ve;//memorize
the path message
                        H[num]->street[i] =
H[num]->street[ve]+1;//memorize street message
                    }
                }
            }
        }
    }
    int FindMin(int dist[],int know[]){//find minterm
        int i,min=INF-1,index=ERROR;
        for(i=1;i<=Na+Ns;i++){
            if(min > dist[i] && know[i] == 0 &&
dist[i]>0){//record the index of min term in uncollected
array
                min = dist[i];
```

```
                index = i;
            }
        }
        return index;//return index
    }
    void Print(int call1){//print result for every call
        int min_dist=INF;
        int min_street=INF;
        int max_Am=-1;
        int index=-1;
        int i,j,k;
        for(i=1;i<=Na;i++){//find minmum distance at first

    if(min_dist>H[i]->dist[call1]&&H[i]->Am>0){//ensure
this hospital has ambulance
                index = i;//record index
                min_dist = H[i]->dist[call1];//update distance
message
                min_street = H[i]->street[call1];//update
street message =
                max_Am = H[i]->Am;//update ambulance
message
```

```
            }
        }
        if(index == -1){//if not be found, index is -1 and it
means All busy
            printf("\nAll Busy");
            return;
        }
        for(i=1;i<=Na;i++){//keep    ambulance number in
mind,and update result

    if(min_dist==H[i]->dist[call1]&&max_Am<H[i]->Am&
&H[i]->Am>0){
                index = i;
                min_street =    H[i]->street[call1];
                max_Am = H[i]->Am;
            }
        }
        for(i=1;i<=Na;i++){//keep street in mind,and
update result

    if(min_dist==H[i]->dist[call1]&&max_Am==H[i]->Am
&&H[i]->Am>0){
```

```
                if(min_street>H[i]->street[call1]){
                    index = i;
                    min_street = H[i]->street[call1];
                }
            }
        }
        (H[index]->Am)--;//reduce ambulance after every
call
        Outputpath(index,call1);//output result
        printf("\n%d",H[index]->dist[call1]);//output
distance
    }
    void Outputpath(int index,int call1){
        int stack[MAXSIZE];//define a stack to inverse the
result
        int top=0;//stack's top
        if((flag++)==0)printf("A-%d",index);//adjust format
of outputting
        else printf("\nA-%d",index);
        int i;
        for(i=call1;i!=index+Ns;i=H[index]->path[i])stack[to
p++]=i;//push into stack one by one
```

```
        for(i=top-1;i>=0;i--){//print result on the screen
            if(stack[i]>Ns)
            printf(" A-%d",stack[i]-Ns);
            else
            printf(" %d",stack[i]);
        }
    }
```

Declaration

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