Fundamentals of Data Structures

Laboratory Projects 3

Ambulance Dispatch

Chapter 1: Introduction

1.1 The content of the question

It is an ambulance dispatch problem. We're given a map of the city, with all the information about the connected streets in the city and the time cost to go through. In this problem, we need to plan the dispatch way to the pick-up spot when the emergency calls. We're supposed to find the minimum time to reach the spot from the dispatch center. If there are many ways to choose, choose the dispatch center that has the most ambulances. If it's still settled down, choose the access that passes the least number of streets.

The first line of the input is the number of pick-up spots and ambulance dispatch centers.

The second line inputs the numbers of the ambulances in each center.

The third line tells how many accesses are in the graph. Then each line tells the vertexes and the weight of the edge.

Finally, input the number of pick-up spots, and their indexes. Then the program will output the result of each call. It tells the vertexes the ambulance passes through, and the time it costs. If there is no ambulance in each center, then print "All Busy".

1.2 The input and output sample(given by the question)

Sample Input

```
7 3    //pick-up spots and ambulance dispatch centers
3 2 2  //the number of ambulances in each center
16     //the edges of the graph
A-1 2 4// v1 v2 weight
A-1 3 2
3 A-2 1
4 A-3 1
A-1 4 3
6 7 1
1 7 3
1 3 3
3 4 1
6 A-3 5
6 5 2
5 7 1
A-2 7 5
A-2 1 1
3 5 1
5 A-3 2
8      //the number of pick-up spots
6 7 5 4 6 4 3 2//pick-up spots
```

Sample 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
```

Chapter 2: Algorithm Specification

2.1 The data structures

2.1.1 Graph

I use the adjacent matrix to store the information about the graph. It may cost a lot of the space, but it will be more efficient to traverse the graph.

```
typedef struct GNode *graph;
struct GNode
{
  int Nv;//the number of vertexs
  int Ne;//the number of edges
  WeightType dist[MaxVNum][MaxVNum];
  //the distance between v1 and v2;
  Vertex start,desti;
  //start vertex and destination
};
```

2.1.2 Array

These arrays are used to store the important information about the vertexes and edges. For example, the dist[] is used to store the time cost between from the pick-up spot to each vertex. And the contents of the array will be updated when the programming is doing the searching algorithm on the graph.

```
int num[11];//num[]is used to store the number of ambulances in each center
int dist[1010],number[1010];prev[1010];
int visited[1010];
//dist[]is used to store the currDist;
//number[]is used to store the number of strees
//prev[]is used to store the pre-vertex.
```

2.2 The Algorithm

2.2.1 Dijkstra Algorithm

I use **Dijkstra Algorithm** to seach for the shortest path(minimum time cost path). Then I will choose the most decent one(according to the problem conditions) among the ambulance dispatch centers.

Pseudo Code

```
int Search(graph G,int v,int maxVertexNum)
{
  int i,j,minv;
  dist[v]=0;   //the dist[]is to
  visited[v]=1;//the first vertex is visited
  number[v]=1; //the number of streets that the ambulance has to pass.
  while(1)
  {
    i=findMinDist(G,dist,visited);
    if(i==-1) break;//donnot find the minimum
    visited[i]=1;//mark the vertex
    for(j=1;j<=G->Nv;j++)
    {
      if(G->dist[i][j]<INFINITY&&!visited[j])
      {
        if(G->dist[i][j]!=0&&dist[i]+G->dist[i][j]<dist[j])//renew the vertex j
          dist[j]=dist[i]+G->dist[i][j];
      }
    }
    v=i;//renew the present vertex.
  }
```

2.3 The functions

```
graph ReadG(graph G,int n);
void buildVertex(int *v1,int *v2);
int Search(graph G,int v,int maxVertexNum);
int findMinDist(graph G,int dist[],int visited[]);
```

2.3.1 Read Graph

This function is used to read in the information of each edge(vertexes and weight), and then store them into the adjacent matrix.

Pseudo Code

```
graph ReadG(graph G,int n)
{
  char c;
  int i,w=0,v1=0,v2=0;
  for(i=0;i<n;i++)
  {
    buildVertex(&v1,&v2);
    G->dist[v1][v2]=w;
    G->dist[v2][v1]=w;
  }
  return G;
}
```

2.3.2 buildVertex

This function is used to return the indexes of vertexes we input. We need use this function to judge the vertex is a dispatch center or a pick-up spots. Because the dispatch centers' indexes are behind the pick-up spots'.

Pseudo Code

```
void buildVertex(int *v1,int *v2)
{
  int c=0;
  char a;
  scanf("%c",&a);
  if(a=='A')
  {
    scanf("%c%d",&a,&c);//c is the index of the dispatch center
  }
  else if(a>='0'&&a<='9')
  {
    c=a-'0';
    do
    {
      scanf("%c",&a);
      if(a!=' ')
        c=c*10+a-'0'; //if the number>10
    }while(a!=' ');
  }
  *v1=c;
  /* the same to *v2 */
  *v2=c;
```

```
}
```

2.3.3 findMinDist

This algorithm is used to find the next vertex(not visited), which has the minimum dist(time cost) path.

Pseudo Code

```
int findMinDist(graph G,int dist[],int visited[])
{
  int minV,minDist,i;
  minDist=INFINITY;
  for(i=1;i<=G->Nv;i++)
  {
    if(!visited[i]&&dist[i]<minDist)
    {
      minDist=dist[i];
      minV=i;
    }
  }
  if(minDist<INFINITY)
    return minV;
  else  return -1;//error
}
```

Chapter 3: Test the project

3.1 Case 1:

Input Sample

```
7 3
3 2 2
16
A-1 2 4
A-1 3 2
3 A-2 1
4 A-3 1
A-1 4 3
6 7 1
1 7 3
1 3 3
3 4 1
6 A-3 5
6 5 2
5 7 1
A-2 7 5
A-2 1 1
3 5 1
```

```
5 A-3 2
9
6 6 6 6 6 6 6 6 6
```

Output Sample

```
A-3 5 6    //6->A-2 and A-3 cost the same time
4          //A-3->6 only needs to go through 2 streets
A-2 3 5 6  //A-2 has more ambulances
4
A-3 5 6
4
A-2 3 5 6
4
A-1 3 5 6
5
A-1 3 5 6
5
A-1 3 5 6
5
All Busy
All Busy
```

In this sample, I test the priority of the judge conditions. When the time cost is the same, then we need choose the center that has more ambulances. When the numbers of ambulances are the same, then we need to choose the one that go through less streets.

3.2 Case 2: The path will pass through other dispatch center.

Input Sample

```
4 2
2 1
5
A-1 1 1
A-1 2 1
A-2 2 1
A-2 3 1
A-2 4 1
2
3 4
```

Output Sample

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

This sample is from a student in our course. The path is special because it will pass through other dispatch center, therefore the output need to be taken note of.

3.3 Case 3: All busy

Input Sample

```
7 3
3 2 2
16
A-1 2 4
A-1 3 2
3 A-2 1
4 A-3 1
A-1 4 3
6 7 1
1 7 3
1 3 3
3 4 1
6 A-3 5
6 5 2
5 7 1
A-2 7 5
A-2 1 1
3 5 1
5 A-3 2
8
6 7 5 4 6 4 3 2
```

Output Sample

```
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
```

This sample is given by the problem.

3.5 Case 4:

Input Sample

```
7 3
3 2 2
16
A-1 2 4
A-1 3 2
3 A-2 1
4 A-3 1
A-1 4 3
6 7 1
1 7 3
1 3 3
3 4 1
6 A-3 5
6 5 2
5 7 1
A-2 7 5
A-2 1 1
3 5 1
5 A-3 2
6
3 3 2 5 4 6
```

Output Sample

```
A-2 3
1
A-2 3
1
A-1 2
4
A-3 5
2
A-3 4
1
A-1 3 5 6
5
```

Chapter 4: Analysis and Comments

4.1 Time Complexity of the Program

$$TimeComplexity = O(|V|^2 + |E|)$$

In Dijkstra's algorithm, we need to find the smallest unknown distance vertex, and then update all the vertexes it's adjacent to. The operation of searching for the smallest distance vertex is time complexity of O(|V|^2). The operation of updating all the adjacent vertexes is O(|E|).

Actually, in this program we can use heap to optimize the time complexity of the program. Using `DeleteMin` to find the smallest vertex. --O(log|V|) However, we need to keep doing `DecreaseKey` until an unknown vertex emerges. --O(|E|log|V|)

$$Time'Complexity'After'Optimization = O(|E|log|V|)$$

4.2 Space Complexity of the Program

$$Space'Complexity = O(|V|^2)$$

The adjacent matrix array to store the graph needs N^2 spaces. Other arrays need N spaces, which are used in Dijkstra's algorithm. Therefore, the total space complexity of the program is O(|V|^2), V represents the total number of pick-up spots and dispatch centers.

Actually, in this program we can use adjacent multilists to store the information of the graph.

$$Time'Complexity'Using'Adajcent'Multilists = O(|V| + |E|)$$

Chapter 5: Declaration

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

Appendix：Source Code in C

```
#include<stdio.h>
#include<stdlib.h>
#define INFINITY 65535
#define MaxVNum 1011
/*----definition of graph---*/
typedef int Vertex;
typedef int WeightType;
typedef struct GNode *graph;
struct GNode
{
  int Nv;//the number of vertexs
  int Ne;//the number of edges
  WeightType dist[MaxVNum][MaxVNum];
  //the distance between v1 and v2;
  Vertex start,desti;
  //start vertex and destination
};
```

```
/*-----definition of the functions-----*/
graph ReadG(graph G,int n);
//in the next n lines, the user will input the information about the graph, then we
build it.
void buildVertex(int *v1,int *v2);
int Search(graph G,int v);
//search for the nearest ambulance.
int findMinDist(graph G,int dist[],int visited[]);
void printVertex(int v);

/*--------variables--------*/
int spot=0, center=0,maxVertexNum=0;
int num[11];//num[]is used to store the number of ambulances in each center
int dist[1010],number[1010];prev[1010];
int visited[1010];
//dist[]is used to store the currDist;
//number[]is used to store the number of strees
//prev[]is used to store the pre-vertex.

int main()
{
  int i,j, line,numv,v,cen;
  char c;
  graph G;
  G=(graph)malloc(sizeof(struct GNode));
  G->Ne=0;
  G->Nv=0;
  scanf("%d %d",&spot, &center);
  maxVertexNum=spot+center;
  G->Nv=maxVertexNum;
  for(i=1;i<=center;i++)
    scanf("%d",&num[i]);
  scanf("%d",&line);
  G->Ne=line;
  scanf("%c",&c);  //absorb the backspace;
  G=ReadG(G,line); //read in and build the graph
  scanf("%d",&numv);   //the number of pick-up spots

  for(i=0;i<numv;i++)
  {
    scanf("%d",&v);
    cen=Search(G,v);
    if(cen!=-1)
    {
      printf("A-%d",cen-spot);//print the center
      for(j=cen;prev[j]!=v;j=prev[j])
        printVertex(prev[j]);
      printVertex(prev[j]);    //print the path
      printf("\n");
```

```
      printf("%d\n",dist[cen]);   //print the length of the path
    }
  }
  return 0;
}

/*---int Search---*/
/*
  function: Searching for the ambulance center which has the shortest path
  algorithm: Using Dijkstra's algorithm
  variables:
    -graph G: the graph we had built
    -int v: the index of the pick-up spot
  return value: the index of the decent ambulance center
*/
int Search(graph G,int v)
{
  int i,j,cnt=0,minv;
  for(i=1;i<=G->Nv;i++)//initialization
  {
    if(G->dist[v][i]!=0)
      dist[i]=G->dist[v][i];
    else dist[i]=INFINITY;
    number[i]=0;
    visited[i]=0;
    prev[i]=0;
  }
  dist[v]=0;
  visited[v]=1;//the first vertex is visited
  number[v]=1;//the number of streets that the ambulance has to pass.
  for(i=1;i<=G->Nv;i++)
    if(G->dist[v][i])
    {
      number[i]=number[v]+1;
      prev[i]=v;
    }
  while(1)
  {
    i=findMinDist(G,dist,visited);
  //  prev[i]=v;
  //  printf("minV=%d\n",i);
    if(i==-1) break;//donnot find the minimum
    visited[i]=1;
    for(j=1;j<=G->Nv;j++)
    {
      if(G->dist[i][j]<INFINITY&&!visited[j])
      {
        if(G->dist[i][j]!=0&&dist[i]+G->dist[i][j]<dist[j])//renew the vertex j
        {
```

```
          dist[j]=dist[i]+G->dist[i][j];
          number[j]=number[i]+1;
          prev[j]=i;
        }
        else if(G->dist[i][j]!=0&&dist[i]+G->dist[i]
[j]==dist[j]&&number[j]>number[i]+1)
        //the length of the streets are equal,but the numbers of streets are different
        {
          number[j]=number[i]+1;
          prev[j]=i;
        }
      }
    }
    v=i;
  }
  int unbusy=0;
  //output the result
  for(i=1;i<=center;i++)
  {
    if(!num[i]) unbusy++;
    else minv=i+spot;
  }
  if(unbusy==center)
  {
    printf("All Busy\n");
    return -1;
  }
  for(i=spot+1;i<=G->Nv;i++)
  {
    if(dist[i]<dist[minv]&&num[i-spot])
      minv=i;
    else if(dist[i]==dist[minv]&&num[i-spot]&&num[i-spot]>num[minv-spot])
      minv=i;
    else if(dist[i]==dist[minv]&&num[i-spot]&&num[i-spot]==num[minv-spot]&&number[i]
<number[minv])
      minv=i;
  }
  if(!num[minv])
  {
    num[minv-spot]--;
    return minv;
  }
  else  return -1;
}

/*---int findMinDist---*/
/*
  function: to find the vertex which has the shortest path and not visited
  variables:
```

```
    -graph G: the graph we had built
    -int dist[]: store the weight of each verge
  return value: the index of the minDist
*/
int findMinDist(graph G,int dist[],int visited[])
{
  int minV,minDist,i;
  minDist=INFINITY;
  for(i=1;i<=G->Nv;i++)
  {
    if(!visited[i]&&dist[i]<minDist)
    {
      minDist=dist[i];
      minV=i;
    }
  }
  if(minDist<INFINITY)
    return minV;
  else  return -1;//error
}

/*---graph ReadG---*/
/*
  function: read in and build the graph
  variables:
    -graph G: it's a null graph that we need to put in.
    -int n: the number of edges we need to read in.
  return value: the graph we've built.
*/
graph ReadG(graph G,int n)
{
  char c;
  int i,w=0,v1=0,v2=0;
  for(i=0;i<n;i++)
  {
    buildVertex(&v1,&v2);
    scanf("%d%c",&w,&c);
    //char c is used to absorb the space
    G->dist[v1][v2]=w;
    G->dist[v2][v1]=w;
  }
  return G;
}

/*---void buildVertex---*/
/*
  function: to deal with the input edges and their weights
  variables:
    -int *v1: the first edge
```

```
    -int *v2: the second edge
  we use the pointers to return the indexes of the vertexes
*/
void buildVertex(int *v1,int *v2)
{
  int c=0;
  char a;
  scanf("%c",&a);
  if(a=='A')
  {
    scanf("%c%d",&a,&c);
    c=c+spot;
    scanf("%c",&a);//absorb the spacekey
  }
  else if(a>='0'&&a<='9')
  {
    c=a-'0';
    do
    {
      scanf("%c",&a);
      if(a!=' ')
        c=c*10+a-'0';
    }while(a!=' ');
  }
  *v1=c;

  c=0;
  scanf("%c",&a);
  if(a=='A')
  {
    scanf("%c%d",&a,&c);
    c=c+spot;
  }
  else if(a>='0'&&a<='9')
  {
    c=a-'0';
    do
    {
      scanf("%c",&a);
      if(a!=' ')
        c=c*10+a-'0';
    }while(a!=' ');
  }
  *v2=c;
}
/*---void printVertex---*/
/*
  function: print the index of vertex in the right form
  variables:
```

```
    -int v: the index of vertex
  we need to judge this vertex is a dispatch center or a pick-up spot.
*/
void printVertex(int v)
{
  if(v>spot)
    printf(" A-%d",v-spot);
  else
    printf(" %d",v);
}
```