Normal Ambulance Dispatch

Date: 2021-11-22

Chapter 1: Introduction

This project is going to solve the problem about how to deal with the emergency calls from different pick-up spots. We will first have a map of a city, which includes all the ambulance dispatch centers and all the pick-up spots and the thing and some emergency calls from all the possible pick-up spots. The thing we need to do is find the appropriate ambulance dispatch center to dispatch an ambulance to that spot. And there are some rules we have to obey. First is that we need to find the center has minimum time cost to arrive the spot. Second is when there are more than one centers have the same minimum time, we have to choose the center which contains more cars in it. Third is when there are more than one centers have not only the same minimum time cost but also have the same maximum number of cars, the number of streets in the path from the center to the spot will be taken into consideration and less streets is better.

In order to solve this problem, we have to use Dijkstra's Algorithms. From Ophaxor, we can know it is an algorithm which allows us to find the shortest path between any two vertices of a graph. It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph. Dijkstra's Algorithm works on the basis that any subpath B→D of the shortest path A→D between vertices A and D is also the shortest path between vertices B and D. Dijkstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors. The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

Chapter 2: Algorithms Specification

For the solution, we can divide it into three big parts. First is using the input information to make a map which contains all the ambulance dispatch centers, the pick-up spots and the streets connect two places in the city. Second, when we take a center as the source, we need to find the costs and the paths from this source to any other destinations in the city with the minimum cost. Third, with the rules have to obey, we can finally give the optimum choice.

1. Making a map of the city

In order to make a map of the city and make it convenient to use in the following steps, we use a matrix to complete. In the first to tenth rows and columns of the matrix, the information of ambulance dispatch centers is recorded while the following are used to record the pick-up spots. And the value of each means the cost when move from one place to another. So in the map, we can know which two places are connected and how much time we will cost when travel from one to another.

m <- the number of streets

for i<-1 to m do

```
    dot1[] <- the name of a place (ambulance dispatch center or pick-up spot)
    dot2[] <- the name of a place
    cost <- the time cost from one side to another along the street
    if dot1[0]= 'A' then d1 <- dot1[2]-'0'
    else d1 <- dot1[0]- '0'+10
    if dot2[0]= 'A' then d2 <- dot2[2]-'0'
    else d2 <- dot2[0]- '0'+10
    map[d1][d2] <- map[d2][d1] <- cost
  end for
```

2. Dijkstra's Algorithm

In order to find and record the shortest way from one ambulance dispatch center to any other places in the city, we use Dijkstra's Algorithm to complete it. And a two-dimension array is used to record all the centers.

```
  for i <- 0 to 3*na with step 3 do
    for t <- 0 to na+ns do
      min <- 32767
      for x <- 1 to 10+ns do
        if hos[i+2][x]=0 & min > hos[i][x] then
          min <- hos[i][x]
          v = x
        end if
      end for
      hos[i+2][v] <- 1
      for j <- 1 to 10+ns do
        if map[v][j] != 0 then
          if hos[i+2][j] = 0 then
            if hos[i][v] + map[v][j] < hos[i][j] then
              hos[i][j] <- hos[i][v] + map[v][j]
              hos[i+1][j] <- v
```

3. Choosing the optimum ambulance dispatch center

There are some rules to help us choose the optimum center to send a car. First, we should use the values recorded in array *hos* to find the minimum time cost center. If only one center is left then simply output the path and its cost and while no center is available, we should output 'All Busy'. Second, when more than two centers have the minimum time cost, we should compare the number of cars available in these centers and choose the center which contains more cars to respond to the emergency call. Third, also when there are more than two centers have the same number of maximum cars, we should take the number of streets it will pass when go for the destination and choose the center which has minimum streets in the path.

3.1 Cost

```
  count <- 0
  for j <- 0 to 3*na with step 3 do
    if min > hos[j][10+spot] & car[j/3+1] != 0 then min <- hos[j][10+spot]
```

```
end for
for j <- 0 to 3*na with step 3 do
   if min = hos[j][10+spot] & car[j/3+1] != 0 then
      count++
      flag[j/3+1] <- 1
      index <- j/3+1
   end if
end for
if count=0 then output ‘All Busy’
else if count=1 then out (3*index-3, 10+spot)
else then examine the number of cars in these centers
end if
```

3.2 Cars

```
samecar <- 0
for j <- 1 to na do
   if flag[j]=1 & car[j]>car[index] then index <- j
end for
for j <- 1 to na do
   if flag[j]=1 & car[j]=car[index] then
      samecar++
      flagcar[j] <- 1
      indexcar <- j
   end if
end for
if samecar = 0 then output ‘All Busy’
else if samecar = 1 then out (3*indexcar-3, 10+spot)
else then examine the number of streets go from these centers to the spot
end if
```

3.3 Streets

```
for j <- 1 to na do
   if flagcar[j]=1 then
      temp <- 10+spot
      while hos[3*j-3+1][temp]!=0 then
         street[j]++
         temp <- hos[3*j-2][temp]
      end while
   end if
end for
minstreet <- na+ns
for j <- 1 to na do
   if flagcar[j]=1 & minstreet>street[j] then
      minstreet <- street[j]
      indexstreet <- j
```

```
    end if
  end for
  out (3*indexstreet-3, 10+spot)
```

3.4 Output the results

```
  center <- the number of the center
  spot <- the number of the pick-up spot
  re_path[0] <- spot
  j <- 0
  while hos[center][spot]!=0 then
    re_path[++j] <- hos[center+1][spot]
    spot <- re_path[j]
  end while
  car[center/3+1]--
  printf (“A-%d”, center/3+1)
  for i <- j-1 to 1 do printf(“ %d”, re_path[i])
  printf (“\n”)
  printf(“%d”, hos[center][re_path[0]])
```

Chapter 3: Testing Results

Input	Output
1 1 2 1 A-1 1 2 2 1 1	A-1 1 2 A-1 1 2
2 1 1 3 A-1 1 2 A-2 A-1 3 A-2 1 4 2 1 2	A-1 1 2 All Busy
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	A-3 5 6 4 A-2 3 5 7 3 A-3 5 2 A-2 3 4 2

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

Chapter 4: Analysis and Comments

1. Analysis

In this program, there are many for loops, so in order to get the time complexities, we have to multiply them, and find the worst is O($Na \times Ns^2$). Na means the number of ambulance dispatch centers while Ns means the number of pick-up spots.

As for the space complexities, because we use arrays to record the result, so the space complexities are determined, so it is O(1).

Time complexities	Space complexities
O($na \times ns^2$)	O(1)

2. Comments

In this project, I have used Dijkstra's Algorithm to solve the problem and divided it into different cases and do different operations for each case. Maybe when doing the Dijkstra, I should also try to calculate and record the streets it will pass so that it will be easier in the following steps.

Appendix: Source Code (in C)

It was put on the last page.

Declaration

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

```
/*
 * File: ambulance_dispatch.cpp
 * Version: 1.0
 * ---------------------------------------------------------------
 * This file implements the problem about how to dispatch ambulance
 * in a city with some requires. First, it should be the fastest
 * way. Second, when more than one center are the fastest, the
 * ambulances should also be more. Third, when still cannot find
 * the unique way, the streets that should be go through also need
 * be taken into consideration.
 */

#include <stdio.h>
#define MAX 32767

/*
 * Global array: map, hos, car
 * ---------------------------
 * These arrays are used to record the information about the city.
 * map is used to record two places that can be connected by a
 * street. hos is used to record the paths and costs when we set
 * from one ambulance dispatch center to all the other places in
 * the city. And car is used to record the number of cars in each
 * ambulance dispatch center.
 */

static int map[2000][2000],hos[50][2000],car[20];

/*
 * Global variable: ns, na, m, k
 * ------------------------------
 * ns is used to record the number of the pick-up spots. na is used
 * to record the number of the ambulance dispatch centers. m is used
 * to record the number of streets connecting the spots and the centers.
 * k is used to record the number of emergency calls.
 */

static int ns,na,m,k;

/*
 * Function: out
 * Usage: out(center, spot);
 * -------------------------
 * This function give the shortest path from one pick-up spot to one
 * ambulance dispatch center and print all the nodes in the path from
 * the ambulance dispatch center to the spot with its cost.
 */

void out(int center, int spot)
{
    static int re_path[2000];
    int i,j = 0;
    re_path[0] = spot;

    // stop until find the ambulance dispatch center
    while(hos[center][spot]!=0){
        re_path[++j] = hos[center+1][spot];
        spot = re_path[j];
    }
    car[center/3+1]--; // minus one car for it has been dispatched

    // output all the nodes in the way and its total cost
    printf("A-%d",center/3+1);
    for (i=j-1;i>=0;i--){
        printf(" %d",re_path[i]-10);
    }
    printf("\n");
    printf("%d\n",hos[center][re_path[0]]);
}

int main()
{
    int i,j,t;
    scanf("%d %d",&ns,&na);
    for (i=1;i<=na;i++) scanf("%d",&car[i]);
    scanf("%d",&m);

    /*
     * Use a matrix to record all the connections in the city between
     * ambulance dispatch centers and pick-up spots. For those have
     * connections, the value would be turned from zero to the cost
     * from the place to another.
     * The beginning 10 rows are used to record the ambulance dispatch
     * center so they are record in map[1-10][] and map[][1-10]. And the
     * following are used to record the pick-up spots.
     */
    for (i=0;i<m;i++){
        char dot1[10],dot2[10];
        int d1,d2,cost;
        scanf("%s %s %d",dot1,dot2,&cost);
        if (dot1[0]=='A') d1 = dot1[2]-'0';
        else d1 = dot1[0]-'0'+10;
        if (dot2[0]=='A') d2 = dot2[2]-'0';
        else d2 = dot2[0]-'0'+10;
        map[d1][d2] = map[d2][d1] = cost;
    }

    /*
     * Use a two-dimension array to record cost, path and known information
     * to do the Dijkstra's Algorithms. So first initialize them. The first
     * line is infinity except the source and used to record the costs from
     * the source to this place, while the second and the third lines are
     * all zero and used to record the precursor and whether it has been
     * visited seperately.
     */
    for (i=0;i<3*na;i+=3){
        for (j=1;j<=10+ns;j++) {
            hos[i][j] = MAX;
            if (j==i/3+1) hos[i][j] = 0;
        }
    }

    for (i=0;i<3*na;i+=3){ // for each ambulance dispatch center, do one step
        int v;
        for (t=0;t<ns+na;t++){ // go through all the places in the city
            int min,x;
            min = MAX;
            // find smallest unknown distance vertex
            for (x=1;x<=10+ns;x++){
                if (hos[i+2][x]==0&&min>hos[i][x]){
                    min = hos[i][x];
                    v = x;
                }
            }
            hos[i+2][v] = 1; // tell that this vertex has been visited

            // find the smallest costs of each node
            for (j=1;j<=10+ns;j++){ // each node adjacent to v
                if (map[v][j]!=0){
                    if (hos[i+2][j]==0){
                        if (hos[i][v]+map[v][j] < hos[i][j]){
                            hos[i][j] = hos[i][v]+map[v][j];
                            hos[i+1][j] = v;
                        }
                    }
                }
            }
        }
    }

    scanf("%d",&k);
    for (i=1;i<=k;i++){ // for each emergency call, do one step
        int spot,min,count=0,index,samecar=0,indexcar,temp,minstreet,indexstreet;
        static int flag[11],flagcar[11],street[11];
        scanf("%d",&spot);
        min = MAX;
        // find the smallest costs from each ambulance dispatch center to the spot
        for (j=0;j<3*na;j+=3){
            if (min>hos[j][10+spot]&&car[j/3+1]!=0) min = hos[j][10+spot];
        }
        // examine whether more than one centers can arrive the spot with this cost
        for (j=0;j<3*na;j+=3){
            if (min==hos[j][10+spot]&&car[j/3+1]!=0){
                count++;
                flag[j/3+1] = 1;
                index = j/3+1;
            }
        }

        // case1: no center has ambulance to give a hand
        if (count==0) printf("All Busy\n");
        // case2: only one center can give the smallest cost
        else if (count==1) out(3*index-3,10+spot);
        // case3: more than one centers can give the smallest cost
        else {
            // find the maximum number of available cars in each smallest-costed center
            for (j=1;j<=na;j++){
                if (flag[j]==1&&car[j]>car[index]) index = j;
            }
            // examine whether more than one centers have the maximum number of cars
            for (j=1;j<=na;j++){
                if (flag[j]==1&&car[j]==car[index]){
                    samecar++;
                    flagcar[j] = 1;
                    indexcar = j;
                }
            }
            // case3-1: no center has ambulance to give a hand
            if (samecar==0) printf("All Busy\n");
            // case3-2: only one center can give the maximum number of cars
            else if (samecar==1) out(3*indexcar-3,10+spot);
            // case3-3: more than one centers can give the maximum number of cars
            else{
                // find the strees in the path from one center to one spot
                for (j=1;j<=na;j++){
                    if (flagcar[j]==1){
                        temp = 10+spot;
                        while(hos[3*j-3+1][temp]!=0){
                            street[j]++;
                            temp = hos[3*j-2][temp];
                        }
                    }
                }
                minstreet = na+ns;
                // find the minimum number of streets in the path and output it
                for (j=1;j<=na;j++){
                    if (flagcar[j]==1&&minstreet>street[j]){
                        minstreet = street[j];
                        indexstreet = j;
                    }
                }
                out(3*indexstreet-3,10+spot);
            }
        }
    }
    return 0;
}
```