

《算法分析与设计》

课后作业

作业编号_____作业 7_____

学 号_____

姓 名_____

专 业_____

学 院_____

二 0 二四年五月

第 7 章 回溯法和分支限界法

题目 1 (回溯法)

数独游戏是在 9×9 的方格中填放 1~9 的数字，要求每一行、每一列以及 3×3 的方格中的数字均不能相同，如下图所示。

1	4	3	6	2	8	5	7	9
5	7	2	1	3	9	4	6	8
9	8	6	7	5	4	2	3	1
3	9	1	5	4	2	7	8	6
4	6	8	9	1	7	3	5	2
7	2	5	8	6	3	9	1	4
2	3	7	4	8	1	6	9	5
6	1	9	2	7	5	8	4	3
8	5	4	3	9	6	1	2	7

现方格中某些数字为空缺（用 0 表示），希望你编写程序能够将空缺的数字补齐。

输入要求：

输入包含 9 行，每一行包含 9 个数字，对应每个方格中的数字，0 表示该方格的数字为空。

输出要求：

输出为 9 行，每行 9 个数字，也就是补齐后的 9×9 的方格中的数字。

样例输入：

103000509

002109400

000704000

300502006

060000050

700803004

000401000

009205800

804000107

样例输出:

143628579

572139468

986754231

391542786

468917352

725863914

237481695

619275843

854396127

(可以参考力扣网或《代码随想录》里的“数独”游戏)

要求:

- 1、写出问题的目标函数和约束函数。
- 2、画出样例输入时的解空间树 (标出解空间树的宽度和深度等于什么。如果树的规模太大, 可以用省略号)。
- 3、采用回溯法编写程序实现上述题目要求。
- 4、以上 (包括源代码) 均采用手写方式写在作业本上, 然后拍照附在电子版里, 再附一个测试的屏幕截图。只提交作业的电子版。

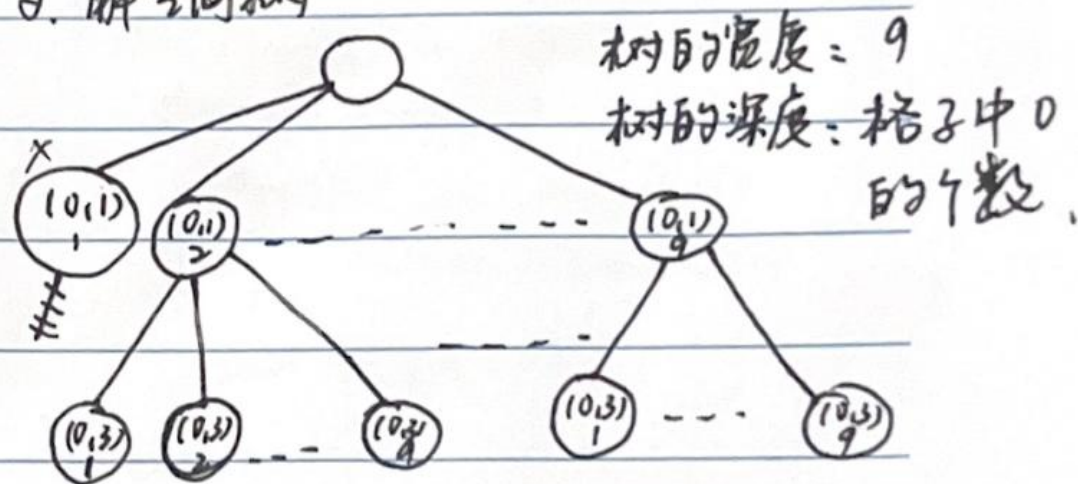
7.1

目标函数：1. 将所有空格正确填满

约束函数：1. 满足数独的规则（行、列、九宫格内不能有相同数字）

2. 不要越界

2. 解空间树：



```

#include <iostream>
#include <vector>
using namespace std;

bool isValid ( const vector<vector<int>>
&board, int row, int col, int num)
{
    for ( int i=0; i<9; i++) {
        if ( board board[row][i] == num ||
            board[i][col] == num ||
            board[3*(row/3)+i/3]
            [3*(col/3)+i%3] == num)
        {
            return false;
        }
        return true;
    }
}

bool solveSudoku (vector<vector<int>>
&board, int row, int col) {
    if ( row==9 ) { return true; }
    if ( col==9 ) {
        return solveSudoku (board, row+1, 0);
    }
    if ( board[row][col] != 0 ) {
        return solveSudoku (board, row,
        col+1);
    }
    for ( int num=1; num<=9; num++) {
        if ( isValid (board, row, col, num) )
        {
            board[row][col] = num;
            if ( solveSudoku (board, row, col+1) )
            {
                return true;
            }
            board[row][col] = 0;
        }
    }
    return false;
}

int main() {
    vector<vector<int>> board (9, vector<int>
(9));
    char a;
    for ( int i=0; i<9; i++) {
        for ( int j=0; j<9; j++) {
            cin >> a;
            board[i][j] = a - '0';
        }
    }
    if ( solveSudoku (board, 0, 0) ) {
        for ( int i=0; i<9; i++) {
            for ( int j=0; j<9; j++) {
                cout << board[i][j];
            }
            cout << endl;
        }
    } else {
        cout << "no solution exists" << endl;
    }
    return 0;
}

```

犀浦校区地址：四川省成都市郫都区犀浦镇犀安路999号

邮编：611756

问题 1 输出 调试控制台 终端

103000509
002109400
000704000
300502006
060000050
700803004
000401000
009205800
804000107
143628579
572139468
986754231
391542786
468917352
725863914
237481695
619275843
854396127

o (base) a1111@Jorhans-MacBook-Air homework7 %

题目 2 (广度优先法)

一个人在一个由方格组成的矩形区域中按上、下、左、右四个方向移动 (每次移动记为一步), 区域内可能存在障碍物, 每个障碍物占用一个方格, 无法通行。计算人从起点方格移动终点方格所需的最少步数。

输入要求:

测试数据的第一行包含 7 个整数 n 、 m 、 k 、 x_0 、 y_0 、 x_1 、 y_1 , 分别表示矩形区域的行、列方格数 (n 和 m), 区域内障碍物的数量 (k), 人的起点方格坐标 (行列号, x_0 和 y_0), 终点方格坐标 (行列号, x_1 和 y_1)。之后的 k 行, 每行包含两个整数, 分别表示障碍物所占用的方格的行列号。

输出要求:

输出 1 行。如果能够到达为所需的最少步数, 否则输出 Impossible。

要求:

- 1、采用标准的广度优先算法编写程序实现上述题目要求。



西南交通大学

SOUTHWEST JIAOTONG UNIVERSITY

第 页

```

题目2:
1. #include <iostream>
   #include <vector>
   #include <queue>
   using namespace std;
   typedef struct
   {
       int x0, y0;
       int distance;
   } node
   int n, m; Vector<Vector<int>>> block;
   int MIN_COST = 10000;
   bool meetblock(int x, int y)
   {
       for (int i = 0; i < block.size(); i++)
       {
           if (x == block[i][0] && y == block[i][1])
           {
               return true;
           }
       }
       return false;
   }
   bool overboundary(int x, int y)
   {
       if (x > n || y > m || x < 1 || y < 1)
       {
           return true;
       }
       else return false;
   }
   Vector<Vector<bool>> visited;
   void BFS (node current, int x1, int y1)
   {
       queue<node> nodelist;
       nodelist.push(current);
       while (!nodelist.empty())
       {
           current = nodelist.front();
           nodelist.pop();
           if (current.x0 == x1 && current.y0 == y1)
           {
               MIN_COST = current.distance;
               return;
           }
           for (int i = 0; i < 4; i++)
           {
               if (!meetblock(current.x0 + direction[i][0],
                               current.y0 + direction[i][1]) ||
                   overboundary(current.x0 + direction[i][0],
                                current.y0 + direction[i][1]) ||
                   visited[current.x0 + direction[i][0]][current.y0 +
                                                                direction[i][1]] == true)
               {
                   continue;
               }
               visited[current.x0 + direction[i][0]][current.y0 +
                                                         direction[i][1]] = true;
               nodelist.push({current.x0 + direction[i][0],
                               current.y0 + direction[i][1],
                               current.distance + 1});
           }
       }
       return;
   }
   int main()
   {
       int k, x0, y0, x1, y1;
       cin >> n >> m >> k >> x0 >> y0 >> x1 >> y1;
       block.resize(k, Vector<int>(2));
       for (int i = 0; i < k; i++)
       {
           cin >> block[i][0] >> block[i][1];
       }
       node start({x0, y0, 0});
       visited.resize(n+1, vector<bool>(m+1, false));
       BFS(start, x1, y1);
       cout << MIN_COST << endl;
   }

```

犀浦校区地址：四川省成都市郫都区犀浦镇犀安路999号

邮编：611756

问题 输出 调试控制台 终端

```

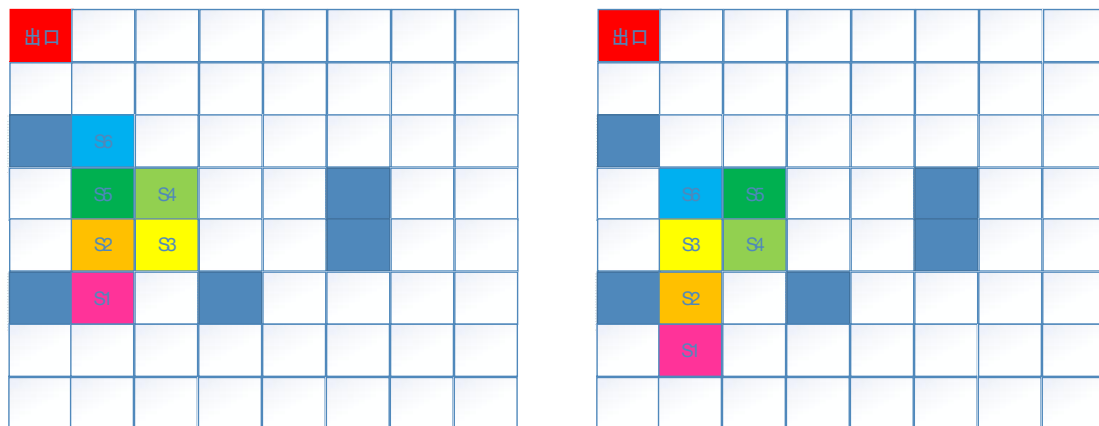
4 3 3 1 1 1
1 3
2 1
4 1
4

```

o (base) a1111@Jorhans-MacBook-Air homework7 %

题目 3 (分支限界法)

可能有不少的同学玩过“贪吃蛇”的游戏，游戏中蛇头带动整个蛇的移动，蛇身将沿蛇头移动过的位置进行移动。如果将整条蛇分成若干个方格，则可以表示为 $S_1, S_2, S_3, \dots, S_L$ ，其中 S_1 为蛇头， S_L 为蛇尾，中间则是蛇头到蛇尾之间的部分。蛇在某个区域内移动时，如果蛇头所在位置的上下左右四个方向没有其它的物体，则蛇头 S_1 可以朝其中任何一个方格移动，蛇身则填补前面移动后的区域，也就是 S_2 移动到 S_1 所在区域， S_3 移动到 S_2 所在区域……，依次类推，如下图所示。



备注：左图为蛇移动前的情况。右图为蛇向下移动一个方格后的情况。蓝色填充的方格表示有障碍物。

请判断蛇是否可以从它当前所在的位置移动到出口所在的位置，如果能，则给出最少需要移动的步数。如果不能，则输出“Impossible”。

输入要求：

输入的第 1 行包含三个整数 n, m ($1 \leq n, m \leq 20$) 和 L ($2 \leq L \leq 8$)，分别表示蛇所在区域的大小（行，列方格数）以及蛇的长度。其后的 L 行用于表示从蛇头到蛇尾依次占用的方格的行列值。紧接着的 1 行包含一个整数 K ，表示该区域内障碍物占用的方格数。之后的 K 行，每行包含两个整数，分别表示障碍物所占用的方格的行列值。出口所在的位置为 $(1, 1)$ ，且出口处没有障碍物。

输出要求：

输出 1 行，如果蛇能够到达出口，则输出蛇从当前位置移动到出口位置最少需要移动的方格数量，否则输出“Impossible”。

样例输入：

```
5 5 4
5 3
4 3
```


4 2

3 2

2

3 1

3 4

要求:

- 1、写出问题的目标函数、约束函数和限界函数。手工计算样例的输出值。
- 2、画出样例输入时的解空间树（只画两层，在树的节点中标出蛇头坐标）。
- 3、画出用分支限界法求解该问题时堆的变化过程，指明堆中每个结点的值及其含义。
- 4、画出采用分支限界法求解该问题时的搜索空间树。
- 5、采用分支限界法编写程序实现上述题目要求。



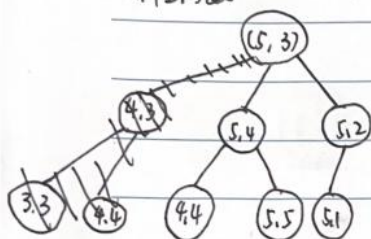
SOUTHWEST JIAOTONG UNIVERSITY

题目3


约束函数: ① 不越界 ② 不遇到障碍物

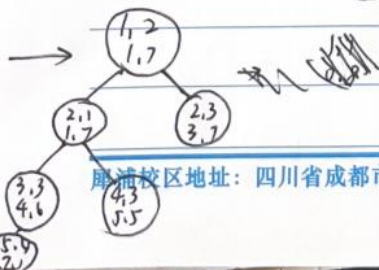
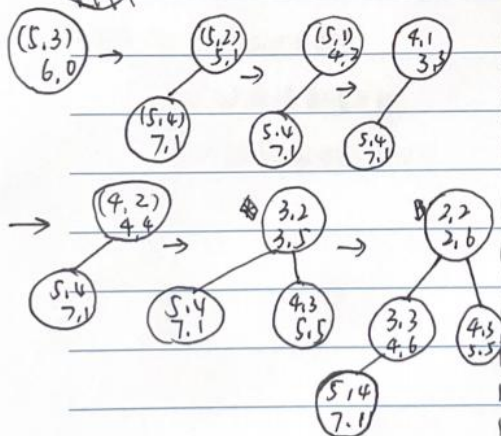
③ 经过的点没有 visited

2. 解空间数

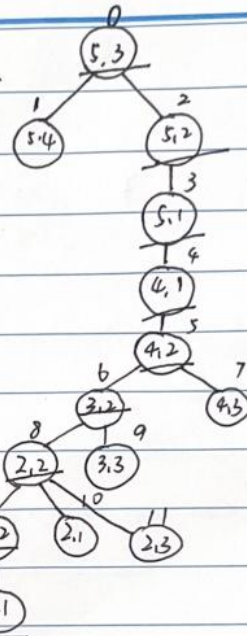


3.

3. 



4.



```
5. #include <iostream>
#include <vector>
#include <queue>
using namespace std;
typedef struct
{
    int x, y;
    int distance;
}
int n, m;
int count_distance (snake head)
{
    return (head.x0-1+head.y0-1);
}
bool meetblock (int x, int y, vector
<snake> current)
{
    for (int i=0; i<block.size(); i++)
    {
        if (x==block[i][0] && y==block[i][1])
    }
```

犀浦校区地址：四川省成都市郫都区犀浦镇犀安路999号

邮编: 611756

```
#include <iostream>
```

```
#include <vector>
```

```
#include <queue>
```

```

#include <chrono>
using namespace std;
using namespace std::chrono;
typedef struct snake
{
    int x0,y0;
    int step;
    int distance;
}snake;
int exit_x,exit_y;
int direction[4][4]={{1,0},{0,1},{-1,0},{0,-1}};
int MIN_DIS=100000;
vector<vector<int>> > block;
vector<vector<bool>> > visited;
int n,m;
int countdistance(snake head)
{

    return (head.x0-1+head.y0-1);
}
bool meetblock(int x,int y,vector<snake> current)
{
    for (int i = 0; i < block.size(); i++)
    {
        if (x==block[i][0]&&y==block[i][1])
        {
            return true;
        }
    }
    for (int i = 1; i < current.size(); i++) //吃到自己
    {
        if (x==current[i].x0&&y==current[i].y0)
        {
            return true;
        }
    }
}

```

```

        }
    }
    return false;
}
bool overboundary(int x,int y)
{
    if (x>n||y>m||x<1||y<1)
    {
        return true;
    }else return false;
}
void moveforward(vector<snake> &current,vector<snake> tempsnake)
{
    for (int i = 1; i < current.size(); i++)
    {
        current[i].x0=tempsnake[i-1].x0;
        current[i].y0=tempsnake[i-1].y0;
    }
}
struct cmp {
    bool operator()( vector<snake> a,vector<snake>  b) {
        return  a[0].distance+a[0].step > b[0].distance+b[0].step;  //小顶堆
    }
};
void BFS(vector<snake> current,int x1,int y1)
{
    priority_queue<vector<snake>, vector<vector<snake>>, cmp> snakelist;
    snakelist.push(current);  //列表中存储的应该是状态
    while (!snakelist.empty())
    {
        current=snakelist.top();  //取蛇的状态
        snakelist.pop();
        if (current[0].x0==x1&&current[0].y0==y1)
        {

```

```

        // MIN_DIS=min(current.distance,MIN_DIS);
        // continue;
        MIN_DIS=current[0].step;
        return;
    }
    for (int i = 0; i < 4; i++)
    {
        if
(meetblock(current[0].x0+direction[i][0],current[0].y0+direction[i][1],current)

||overboundary(current[0].x0+direction[i][0],current[0].y0+direction[i][1])

||visited[current[0].x0+direction[i][0]][current[0].y0+direction[i][1]])
        {
            continue;
        }

visited[current[0].x0+direction[i][0]][current[0].y0+direction[i][1]]=true;
        vector<snake> tempsnake1=current;
        vector<snake> tempsnake2=current;
        tempsnake1[0].x0=current[0].x0+direction[i][0];
        tempsnake1[0].y0=current[0].y0+direction[i][1];
        tempsnake1[0].step++;
        tempsnake1[0].distance=countdistance(tempsnake1[0]);
        moveforward(tempsnake1,tempsnake2);
        snakelist.push(tempsnake1);
    }
}

return;
}

int main()
{
    int k,x0,y0,l;

```

```

    cin>>n>>m>>l;
    vector<snake> snakes;
    snakes.resize(l,{0,0,0,10000});
    for (int i = 0; i < l; i++)
    {
        cin>>snakes[i].x0>>snakes[i].y0;
    }
    cin>>k;
    block.resize(k,vector<int>(2));
    for (int i = 0; i < k; i++)
    {
        cin>>block[i][0]>>block[i][1];
    }
    visited.resize(n+1,vector<bool>(m+1,false));
    snakes[0].distance=snakes[0].x0-1+snakes[0].y0-1;
    //auto start = high_resolution_clock::now();
    BFS(snakes,1,1);
    //auto stop = high_resolution_clock::now();
    //auto duration = duration_cast<microseconds>(stop - start);
    cout<<MIN_DIS<<endl;
    //cout << "Execution time: " << duration.count() << " microseconds" <<
endl;
}

```

```

问题 输出 调试控制台 终端
5 5 4
5 3
4 3
4 2
3 2
2
3 1
3 4
8
(base) a1111@Jorhans-MacBook-Air homework7 %

```

题目 4 (分支限界法)

小李希望开车到全国各地旅游,他发现旅途中各个城市的汽油价格也不相同,显然如果能够采取合理的加油方式,将会节省整个旅途的费用。假设汽车开始时油箱为空,汽车每走一个单位的距离将耗费一个单位的汽油。请帮助小李找到一种最省钱的方式完成他的整个自驾游。

输入要求:

输入的第一行包含两个整数 n ($1 \leq n \leq 1000$) 和 m ($0 \leq m \leq 10000$), 分别表示城市的数量以及道路的数量。其后的一行包含 n 个整数, 分别表示 n 个城市的汽油价格。其后的 m 行, 每一行包含三个整数 u, v, d , 表示城市 u, v 之间的距离为 d ($1 \leq d \leq 100$)。最后一行包含三个整数 c ($1 \leq c \leq 100$), s, e , 分别表示油箱的容量, 出发的城市和最后到达的城市。

输出要求:

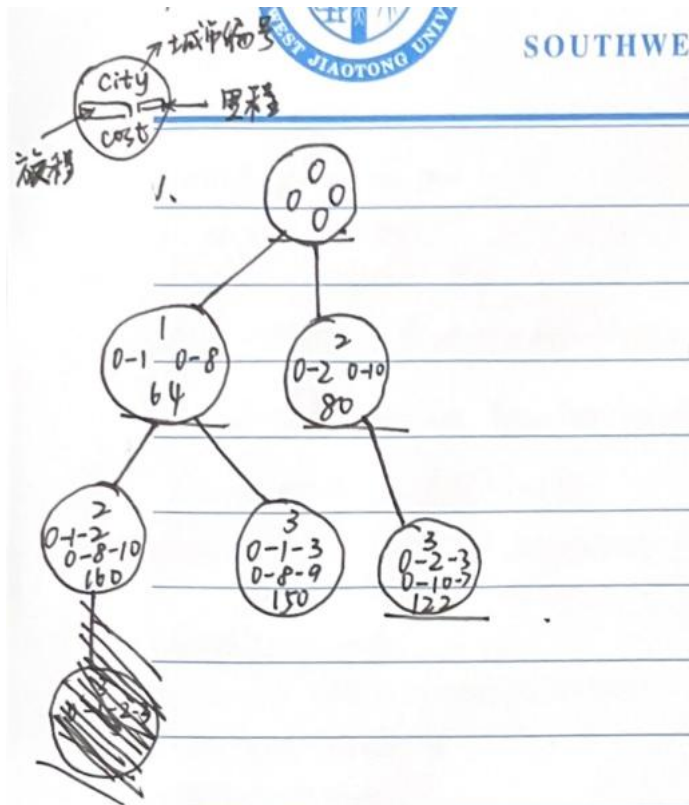
如果能够帮助小李找到一个最省钱从 s 到 e 的路线, 则输出对应的费用, 否则输出 “impossible”。

样例输入:

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

要求:

- 1、画出采用分支限界法求解样例输入时的解空间树 (必要时可以用省略号)。
- 2、给出采用分支限界法求解该问题时的目标函数, 约束条件以及限界函数。
- 3、采用分支限界法编写程序实现上述题目要求。



2. 目标函数：最小化费用 $\min(cost)$

约束条件：1. 城市之间有路

2. 城市没有被走过

3. 城市间距离不大于10

```

#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
int n,m;
vector<int> gas_price;
vector<vector<int>> map;
  
```

```

vector<bool> visited;
typedef struct
{
    int citylabel;
    // int left_gas;
    // int gas_price;
    vector<int> travel_distance;
    vector<int> citytraveled;
    int cost;
} travel_status;
int MIN_COST=10000;
int count_cost(vector<int> citytraveled,vector<int> travel_distance)
{
    int min_cost=100000;
    for (int i = 0; i < citytraveled.size()-1; i++)
    {
        min_cost=min(min_cost,gas_price[citytraveled[i]]);
    }
    int min_city_label;
    for (int i = 0; i < citytraveled.size()-1; i++)
    {
        if (gas_price[citytraveled[i]]==min_cost)
        {
            min_city_label=citytraveled[i];
        }
    }

    int sum_cost=0;
    for (int i = 0; i < citytraveled.size(); i++)
    {
        if (citytraveled[i]==min_city_label)
        {
            for (int k = i+1; k < travel_distance.size(); k++)
            {

```

```

        sum_cost=sum_cost+min_cost*travel_distance[k];
    }
    if (i==0)
    {
        return sum_cost;
    }
    vector<int> sub_citytraveled,sub_travel_distance;
    sub_citytraveled.assign(citytraveled.begin(),citytraveled.begin()+i+1);

sub_travel_distance.assign(travel_distance.begin(),travel_distance.begin()+i+1);

        return sum_cost+count_cost(sub_citytraveled,sub_travel_distance);
    }

}
}
struct cmp{
    bool operator()(travel_status a,travel_status b)
    {
        return a.cost>b.cost;
    }
};
void BFS(int start,int end,int total_gas)
{
    visited[start]=true;
    priority_queue<travel_status,vector<travel_status>,cmp> status_list;
    travel_status initial_status,next_status;
    initial_status.citylabel=start;
    initial_status.travel_distance.push_back(0);
    initial_status.citytraveled.push_back(start);

    initial_status.cost=count_cost(initial_status.citytraveled,initial_status.travel_distance);
    status_list.push(initial_status);

```

```

while (!status_list.empty())
{
    travel_status current=status_list.top();
    status_list.pop();
    visited[current.citylabel]=true;
    if (current.citylabel==end)
    {
        MIN_COST =
min(MIN_COST,count_cost(current.citytraveled,current.travel_distance));
        return;
    }
    for (int i = 0; i < map[current.citylabel].size(); i++)
    {
        if
(map[current.citylabel][i]!=0&&visited[i]==false&&map[current.citylabel][i]<=total_
gas)
        {
            next_status.citylabel=i;
            next_status.travel_distance=current.travel_distance;
            next_status.travel_distance.push_back(map[current.citylabel][i]);
            next_status.citytraveled=current.citytraveled;
            next_status.citytraveled.push_back(i);

            next_status.cost=count_cost(next_status.citytraveled,next_status.travel_distance);
            status_list.push(next_status);
        }
    }
}
}

```

```

int main()
{
    cin>>n>>m;
    gas_price.resize(n,0);
    map.resize(n,vector<int>(n,0));
    visited.resize(n,false);
    for (int i = 0; i < n; i++)
    {
        cin>>gas_price[i];
    }
    int a,b,d;
    for (int i = 0; i < m; i++)
    {
        cin>>a>>b;
        cin>>d;
        map[a][b]=map[b][a]=d;
    }
    int total_gas,start,end;
    cin>>total_gas>>start>>end;
    BFS(start,end,total_gas);
    cout<<MIN_COST<<endl;
}

```

测试截图:

```

问题 1  输出  调试控制台  终端
4 6
8 10 6 7
0 1 8
0 3 12
122
(base) a1111@Jorhans-MacBook-Air homework7 %

```

基本思想:

分层图+最短路径

参考文档:

https://blog.csdn.net/qg_45735851/article/details/108219481?spm=1001.2101.3001.6650.5&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5-108219481-blog-124555794.pc_relevant_downloadblackli

[stv1&depth 1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5-108219481-blog-124555794.pc_relevant_downloadblacklistv1&utm_relevant_index=10](#)