P2895 [USACO08FEB]Meteor Shower S

题目描述

贝茜听说了一个骇人听闻的消息：一场流星雨即将袭击整个农场，由于流星体积过大，它们无法在撞击到地面前燃烧殆尽，届时将会对它撞到的一切东西造成毁灭性的打击。很自然地，贝茜开始担心自己的安全问题。以 Farmer John 牧场中最聪明的奶牛的名誉起誓，她一定要在被流星砸到前，到达一个安全的地方（也就是说，一块不会被任何流星砸到的土地）。如果将牧场放入一个直角坐标系中，贝茜现在的位置是原点，并且，贝茜不能踏上一块被流星砸过的土地。 根据预报，一共有 M 颗流星 (1≤M≤50,000) 会坠落在农场上，其中第i颗流星会在时刻 Ti(0≤Ti≤1,000) 砸在坐标为 (Xi,Yi) (0≤Xi≤300，0≤Yi≤300) 的格子里。流星的力量会将它所在的格子，以及周围 4 个相邻的格子都化为焦土，当然贝茜也无法再在这些格子上行走。

贝茜在时刻 0 开始行动，它只能在第一象限中，平行于坐标轴行动，每 1 个时刻中，她能移动到相邻的（一般是 4 个）格子中的任意一个，当然目标格子要没有被烧焦才行。如果一个格子在时刻 t 被流星撞击或烧焦，那么贝茜只能在 t 之前的时刻在这个格子里出现。 贝西一开始在(0,0)。

请你计算一下，贝茜最少需要多少时间才能到达一个安全的格子。如果不可能到达输出-1。

输入格式

第一行：整数M，代表流星的数量

第二行到第M+1行:包含三个用空格分隔的整数Xi，Yi，Ti，表示流星将在Ti时间后降落在Xi,Yi格子

输出格式

一行：贝茜最少需要多少时间才能到达一个安全的格子。如果不可能到达输出-1

输入样例

```
4
0 0 2
2 1 2
1 1 2
0 3 5
```

输出样例

```
5
```

解析

在一个300x300的矩阵中，奶牛需要躲避天上的流星，并在流星到达之前跑到安全地带。我们先通过图像，来理解这道题。首先是流星的坠落时间与地表损坏的图像。

首先是，2秒后(0,0)点，连同其周围的2个点被灼烧。

	0	1	2	3	4	5
0	2	2				
1	2					
2						
3						
4						
5						

接下来是(2,1)点在2秒后，连同周围的4个点被灼烧。

	0	1	2	3	4	5
0	2	2	2			
1	2	2	2	2		
2			2			
3						
4						
5						

(1,1)点在2两秒后，连同周围的4个点被灼烧

	0	1	2	3	4	5
0	2	2	2			
1	2	2	2	2		
2		2	2			
3						
4						
5						

最后则是(0,3)点在5秒后被灼烧

	0	1	2	3	4	5
0	2	2	2			
1	2	2	2	2		
2	5	2	2			
3	5	5				
4	5					
5						

将所有点进行合并，则会得到下图，也就是说如果奶牛能够在最短的时间内跑出红色区域，就能得救。

	0	1	2	3	4	5
0	2	2	2			
1	2	2	2	2		
2	5	2	2			
3	5	5				
4	5					
5						

由此，我们推导出我们需要计算奶牛到达每个格子的最小时间，显然，这里应该使用广度优先搜索。在这里，我们同样用图像来进行标记奶牛到达每个格子所需要的时间。

	0	1	2	3	4	5
0	0	1	x			
1	1	x	x	x	9	
2	2	x	x	7	8	9
3	3	4	5	6	7	8
4	4	5	6	7	8	9
5	5	6	7	8	9	

上图中“x”代表无法到达这个点，因为当你到达这个点的时候，你所用的时间将大于等于流星落下的时间，导致放个被燃烧，奶牛无法站立。

通过观察，很明显，奶牛至少需要五步才能够逃离。因此，正确答案是5。

通过分析，我们需要建立以下几个数组：

1、死亡计时数组，用来记录每个格子被的时间，奶牛必须在这个时间前通过当前格子。注意这个格子一定要存储最早被销毁的那个时刻，因为流星掉落时会连同周围的四个格子一起灼烧。

2、通行计时数组，用来存储奶牛通过某个格子所用的最小时间，通过广搜计算。其中的最短的，并且大于最后一个流星落下的时间，就是我们所要求的结果值。

编码

```
#include<bits/stdc++.h>

using namespace std;
//最大的边界值
int const maxRow = 305;
int const maxCol = 305;
//最小的边界值
int const minRow = 0;
int const minCol = 0;
//最大的步骤数
int const maxStep = 0x3f3f3f3f;
//可移动的方向数量
int const forwardNum = 4;

struct Node {
    int Row; //行数
   int Col; //列数

   //有参数的构造函数
   Node(int row, int col) {
        this->Row = row;
        this->Col = col;
```

```
    }

    //无参数的构造函数
    Node() {};
};

//答案数组
int ans[maxRow][maxCol];
//死亡时间记录
int deadTime[maxRow][maxCol];

//搜索队列，队列版
queue<Node> SearchQueue;

int Forward[forwardNum][2] = {
        //第一维描述的纵向变化，第二维描述的是横向变化
       {-1, 0}, //向上移动
       {1,  0}, //向下移动
       {0,  -1},  //向左移动
       {0,  1},  //向右移动
};

//检测节点是否有效，可以被存储
bool CheckNode(int row, int col, int step) {
    //1、是否越界
    //2、是否被访问过
    if (row >= minRow && col >= minCol && row < maxRow && col < maxCc
        //没有走过这条路
       if (maxStep == ans[row][col]) {
           //能够赶在死亡时间之前
           if (step < deadTime[row][col]) {
               return true;
           }
       }
    }
    return false;
}

//广度优先搜索（通过队列）
void BfsByQueue(Node start) {
    //将移动的步数赋值为最大
    memset(ans, maxStep, sizeof(ans));
    //将起点的步数设置为0
    ans[start.Row][start.Col] = 0;
    //对列中有值,清空队列
    while (!SearchQueue.empty()) {
        SearchQueue.pop();
    }
    //将起点放到队列
```

```
    SearchQueue.push(start);
     //启动搜索循环
    while (!SearchQueue.empty()) {
         //取出第一个点
        Node curNode = SearchQueue.front();
         //从队列中删除
        SearchQueue.pop();
         //当前步数
        int step = ans[curNode.Row][curNode.Col];
         //查找四个可以走的方向
        for (int i = 0; i < forwardNum; ++i) {
             int newRow = curNode.Row + Forward[i][0];
             int newCol = curNode.Col + Forward[i][1];
             if (CheckNode(newRow, newCol, step + 1)) {
                 //将当前节点存入队列
                 SearchQueue.push(Node(newRow, newCol));
                 ans[newRow][newCol] = step + 1;
             }
         }
     }
}

//记录死亡时间
void recordDeadTime(int x, int y, int time) {
     if (x >= 0 && y >= 0) {
         deadTime[x][y] = min(deadTime[x][y], time);
     }
}

int main() {
     //设置死亡时间数组的值为无穷大
    memset(deadTime, maxStep, sizeof(deadTime));
     //流星的个数
    int n;
     cin >> n;
     //流星将要落下来的点
    int x, y;
     //流星落下来的时间
    int t, maxT;
     //初始化最早被摧毁的田地的时间
    for (int i = 0; i < n; ++i) {
         cin >> x >> y >> t;
         //记录流星下落的最大时间
        maxT = max(maxT, t);
         //记录某个田地最早被摧毁的时间
        recordDeadTime(x, y, t);
         for (int j = 0; j < forwardNum; ++j) {
             //上下左右移动
           //注意各个数组存储的顺序，第一维均是x轴，第二维均是y轴
```

```
                recordDeadTime(x + Forward[j][1], y + Forward[j][0], t);
            }
        }

        //最初时刻，奶牛在0,0点
        Node start = Node(0, 0);
        BfsByQueue(start);

        //到达安全距离的最短时间
        int minTime = maxStep;
        //遍历到达每个终点的最小步数
        for (int i = 0; i < maxRow; ++i) {
            for (int j = 0; j < maxCol; ++j) {
                //大于最后一颗流星落下的时间
                if (deadTime[i][j] > maxT) {
                    minTime = min(minTime, ans[i][j]);
                }
            }
        }
        //没有找到通路
        if (minTime == maxStep) {
            cout << -1;
        }
        //存在通路，打印最短时间
        else {
            printf("%d", minTime);
        }

        return 0;
}
```


›l) {