走迷宫

题目描述

网站全局地址

课程内部地址

已知一$N \times N$的迷宫，允许往上、下、左、右四个方向行走，现请你找出一条从左上角到右下角的路径。

输入数据有若干行，第一行有一个自然数N（$N \leq 20$），表示迷宫的大小，其后有N行数据，每行有N个0或1（数字之间没有空格，0表示可以通过，1表示不能通过），用以描述迷宫地图。入口在左上角$(1,1)$处，出口在右下角(N,N)处。所有迷宫保证存在从入口到出口的可行路径。

输出数据仅一行，为从入口到出口的路径（为确保答案的唯一性，在x表示行，y表示列的前提下，请严格按照 左 右 上 下 的顺序试探路径）。

主要思路

本题为迷宫问题，首选使用搜索。而为了方便输出路径，我们选择DFS，因为DFS不需要BFS一样维护队列，而是可以一边搜索一边保存路径。

依照课程中的讲解，我们先处理输入，把地图保存在一个二维数组中并开始深搜。由于不能走重复路径，因而不妨设置一个二维数组保存访问情况，一边行走一边标记，在到达点(N,N)后输出路径并结束程序。

代码

为了方便表示地图的长、宽，我们设置两个变量n,m，使得$m=n$（n为输入）。

```
#include<bits/stdc++.h>
using namespace std;
int m,n,x,y,dx[]={0,0,-1,1},dy[]={-1,1,0,0},route[401],maps[21][21];//
bool vis[21][21];
void print(int k)
{
    cout<<"(1,1)";//先输出坐标(1,1)
    int a=1,b=1;//a,b表示横纵坐标，一开始为(1,1)
    for(int i=1;i<k;i++)
    {
        a+=dx[route[i]],b+=dy[route[i]];//调用route数组，route数组保存的
        cout<<"->("<<a<<','<<b<<')';//格式化输出路径
    }
    exit(0);//输出完路径，通过这一行直接结束整个程序
}
void dfs(int dep)//dep表示深度，即调用次数
{
    if(x==n&&y==m)//到达迷宫终点则视为结束，应该输出路径
    {
        print(dep);
        return;
```

```
        }
        for(int i=0;i<4;i++)
        {
            int nx=x+dx[i],ny=y+dy[i];
            if(nx>0&&nx<=n&&ny>0&&ny<=m&&!maps[nx][ny]&&!vis[nx][ny])
            /*
            要想前往到坐标为(nx,ny)的位置，则需要满足三个条件：
            1. 必须在地图内
                1-1. 横坐标大于0
                1-2. 横坐标小于等于n
                1-3. 纵坐标大于0
                1-4. 纵坐标小于等于m（也就是n）
            2. 该位置数值为0（只有数值为0的迷宫才是可以走得通的）
            3. 该位置没有被走过（即该位置的vis数组标记为false）
            */
            {
                vis[nx][ny]=1;//将新位置的vis数组标记为true，表示已经访问
                x=nx,y=ny,route[dep]=i;//x和y进行值的替换，表示位置已经变动，然
                dfs(dep+1);//继续下一层搜索
                vis[x][y]=0;//回溯——把这个位置的vis数组标记为false
                x-=dx[i],y-=dy[i];//回溯——把x和y替换为原值
            }
        }
    }
    int main()
    {
        scanf("%d",&n);//输入地图的边长
        m=n;
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=m;j++)
            {
                scanf("%1d",&maps[i][j]);
                //对于一个int类型变量t，scanf("%1d",&t);语句可以每次接收一个数位，
            }
        }
        vis[1][1]=true;//把(1,1)的vis数组标记为true，表示已经走过，因为深搜开始时
        x=y=1;//x和y都赋值为1，因为初始坐标为(1,1)
        dfs(1);//从第1层开始搜索
        return 0;
    }
```

易错点/注意事项

搜索类题目的细节是很多的。一旦犯了错误，那么可能造成满盘皆输。如果这一步错了，即使其他部分都是正确的，那么也只能拿到像70, 40分甚至0分（即“期望得分”）。下面的错误都是我犯过的。

- 易错点1：拓展顺序错误

拓展顺序取决于方向增量数组，所以方向增量数组必须写对。错误的方向增量数组有很多种，下面列举一种：

```
int dx[]={-1,0,0,1},dy[]={0,-1,1,0};
//这是按照上左右下的顺序的
```

使用上述的错误增量数组，得分为70分，拿到这么高的分的原因是因为很多时候合法拓展的方式只有1种，所以顺序的重要性也就不大，但是要想拿到100分，那么方向增量数组必须按照正确的顺序进行赋值。

解决方法：

```
int dx[]={0,0,-1,1},dy[]={-1,1,0,0}
//正确顺序——左右上下
```

- 易错点2：直接输入地图

期望得分：0分

本题可以用字符或整型二维数组保存地图，如果是字符地图的话，可以直接用cin输入，但是对于用整型保存的地图，我们不能直接输入，而是要一个数位一个数位地输入。

解决方法1：

```
for(int i=1;i<=n;i++)
{
    for(int j=1;j<=m;j++)
    {
        scanf("%1d",&maps[i][j]);
    }
}
```

解决方法2：

```
for(int i=1;i<=n;i++)
{
    for(int j=1;j<=m;j++)
    {
        char ch;
        cin>>ch;
        if(ch=='1')maps[i][j]=1;
        else maps[i][j]=0;
        //上述两句可以简化成一句代码——maps[i][j]=ch=='1';
    }
}
```

- 易错点3：忘记标记vis(1,1)的值为真

期望得分：40分

对于某一些数据，如果我们忘记标记$vis(1,1)$为真，那么搜索的过程中很有可能回到$(1,1)$，然后继续搜索下去。这样路径就会重复。

解决方法：

```
vis[1][1]=true;//该句要在搜索前执行
```

- 易错点4：回溯顺序错误

期望得分：70分

错误回溯代码：

```
x-=dx[i],y-=dy[i];//先将x、y重新赋值
vis[x][y]=0;//再将vis数组重新标记
```

对于上述代码，程序会把x,y的状态回到上一步，然后再对原先位置的vis标记为假（即未访问）。但是实际上我们要做的是把当前（即行不通）的位置的vis标记为假，起到回溯的作用。

解决方法：

```
vis[x][y]=0;//先将vis数组重新标记
x-=dx[i],y-=dy[i];//再将x、y重新赋值
```

我们只需要把两个语句交换顺序即可，可见注意细节的重要性！

- 易错点5：输出范围错误

期望得分：0分

在最后输出的时候，由于要用$print$函数进行输出，所以要涉及到传参。而传送的参数为到达迷宫终点的层数，即深搜函数中的dep传递给$print$函数中的k。但是$route(k)$并没有被赋值过，因为深搜函数还未运行到向4个方向拓展，就已经进行打印了。所以循环应该是在$[1,k)$区间，而不是$[1,k]$区间。

错误代码：

```
for(int i=1;i<=k;i++)
{
    a+=dx[route[i]],b+=dy[route[i]];
    cout<<"->("<<a<<','<<b<<')';
}
```

解决方法：

```
for(int i=1;i<k;i++)//此时route[k]的值其实是0（因为全局变量所有元素默认为0），
{
    a+=dx[route[i]],b+=dy[route[i]];
    cout<<"->("<<a<<','<<b<<')';
}
```