**HDU1728逃离迷宫**

**Time Limit: 1000/1000 MS (**[**Java**](http://lib.csdn.net/base/javaee)**/Others)    Memory Limit: 32768/32768 K (Java/Others)  
Total Submission(s): 2426    Accepted Submission(s): 566**

**Problem Description**

　　给定一个 m × n (m 行 , n 列 ) 的迷宫，迷宫中有两个位置， gloria 想从迷宫的一个位置走到另外一个位置，当然迷宫中有些地方是空地， gloria 可以穿越，有些地方是障碍，她必须绕行，从迷宫的一个位置，只能走到与它相邻的 4 个位置中 , 当然在行走过程中，gloria 不能走到迷宫外面去。令人头痛的是， gloria 是个没什么方向感的人，因此，她在行走过程中，不能转太多弯了，否则她会晕倒的。我们假定给定的两个位置都是空地，初始时， gloria 所面向的方向未定，她可以选择 4 个方向的任何一个出发，而不算成一次转弯。 gloria 能从一个位置走到另外一个位置吗？

**Input**

　　第 1 行为一个整数 t (1 ≤ t ≤ 100), 表示[**测试**](http://lib.csdn.net/base/softwaretest)数据的个数，接下来为 t 组测试数据，每组测试数据中，   
　　第 1 行为两个整数 m, n (1 ≤ m, n ≤ 100), 分别表示迷宫的行数和列数，接下来 m 行，每行包括 n 个字符，其中字符 '.' 表示该位置为空地，字符 '\*' 表示该位置为障碍，输入数据中只有这两种字符，每组测试数据的最后一行为 5 个整数 k, x1 , y1 , x2 , y2 (1 ≤ k ≤ 10, 1 ≤ x1 , x2 ≤ n, 1 ≤ y1 , y2 ≤ m), 其中 k 表示 gloria 最多能转的弯数， (x1 , y1 ), (x2 , y2 ) 表示两个位置，其中 x1 ， x2 对应列， y1 , y2 对应行。

**Output**

　　每组测试数据对应为一行，若 gloria 能从一个位置走到另外一个位置，输出 “yes” ，否则输出 “no” 。

**Sample Input**

2

5 5

...\*\*

\*.\*\*.

.....

.....

\*....

1 1 1 1 3

5 5

...\*\*

\*.\*\*.

.....

.....

\*....

2 1 1 1 3

**Sample Output**

no

yes

一道可走重复点的经典bfs(dfs)

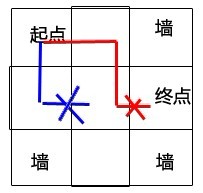
理论上一般的走迷宫肯定是bfs找一条最短路径(况且本题还只是确定是否有解)，于是我开始还是先用了dfs来做，想到可能会爆栈就放弃了，看了题解发现可用dfs，只要会剪枝就可以了(功力不够...)。所以我就想应该是一般的bfs，但是发现正确率只有24%,所以一定有坑！在发现输入坑后，又反复排查了代码中的初始化，清零队列，输入输出结果可能是障碍后，还是wa, 并且过了网上

给的许多特殊样例，依然WA,不由得我陷入了对人生的思考。。。对了，这次做题还发现一个从来没见过的错误就是#include<bits/stdc++.h>这个头文件会和定义在main()外的int型y1变量冲突......原因是头文件里定义了double型y1,所以重复定义了。。。。

看了题解(真是永远都这么菜)后，终于明白自己错误的原因了。一开始思路就错了。这个题目是求能否到达，所以肯定会有多种可以到达的解，这就会出现一个问题，多个可达路径中会有点被多次访问，并且这多个路径拐弯的次数也不同，恰好题目也限制了转弯次数，而我还在用一般的BFS模式做，这就产生了错误的原因：迷宫类型的BFS会设置一个vis[][]用0和1记录某一位置是否被访问过。问题就在这里(况且我写的还是最普通的BFS，即使不WA,也极有可能超时，因为搜索中有大量可以剪掉的部分)，这会造成把正确的解给覆盖掉！

如下图：

（假如转弯数k=1,起点终点下， 那么如果你的代码是优先向右搜索就会出错 ，因为红色的线是先搜的，由于最多转一次弯，所以不合题意；   
蓝色是后搜的，因为遇到转弯数相等(我的代码里是因为vis[][]=1访问过，实际上可以看到这样设置标记是错误的)所以不往下走了了，但是继续走是可以满足题意输出"yes"的   
）



所以，这个BFS不能用走过没走过来确定是否走或不走该点，因为同一个点在选择的不同路径上应该可以被访问多次，应该用别的条件来判断。既然给出了拐弯次数限制，这样就可以想到如果有解，则一定会在k内走完，所以可以设置每个点在当前路径上已经拐了几次弯，然后优先走当前k小的点。这样的好处就是一来避免了vis[][]错误剪枝漏解(因为同一点可以被多次访问)，二来是利用了贪心和宽搜的思想，即在满足k的条件下同时顺便找出了最优解，具体做法：对当前点的转向数做一个标记，对重复经过的点只第一次标记入队而后面搜索时不重复加入队列(当作看不到，因为我们要的是最小转弯数，交叉重复并不影响广搜贪心性的实现，这个题目考我们限制条件从一般简单题目的步数到现在的转弯数，目的就是在考我们这一点)，对四个方向 进行BFS, 但不是一步一步 , 也就是搜一个方向的时候一直搜索到沿这个方向能走到的尽头 , 这样搜的话 , 能够保证搜过的路径的转弯次数都是最小的 ( 因为这是基于转弯次数从 0 到 k 的 BFS，即优先走完转向k小的点).

下面是本人的错误代码

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<cstdio>

#include<cmath>

#include<algorithm>

#include<cstdlib>

#include<climits>

#include<deque>

#include<stack>

#include<vector>

#include<queue>

#include<bitset>

#include<map>

#include<set>

#include<functional>

#include<cstring>

#include<cctype>

#include<bits/stdc++.h>

typedef long long ll;

const int MOD=10e9+7;

const int INF=0x7fffffff;

const double ESP=10e-7;

const double Pi=acos(-1.0);

const int dr[]= {0,0,1,-1,1,1,-1,-1};//8向

const int dc[]= {-1,1,0,0,-1,1,-1,1};

const int fdr[]={0,0,1,-1};//上，下，右，左

const int fdc[]={-1,1,0,0};

#define maxn INF

#define min -INF

using namespace std;

char f[110][110];

int flag[110][110];

int x1,y11,x2,y2,m,n;

int res;

struct pos{

int x;

int y;

int d;

int kk;

};

int main(void)

{

int t,k,i,c,r,j,lastd,d,tmpk;

int first,last;

int xx,yy,x,y;

queue<pos>q;

pos tmp;

scanf("%d",&t);

for(i=0;i<t;i++)

{

memset(f,'\0',sizeof(f));

memset(flag,0,sizeof(flag));

while(!q.empty())

{

q.pop();

}

scanf("%d%d",&m,&n);

res=0;

first=last=0;

for(r=0;r<m;r++)

for(c=0;c<n;c++)

{

cin>>f[r][c];

}

scanf("%d%d%d%d%d",&k,&x1,&y11,&x2,&y2);

flag[y11-1][x1-1]=1;

x=y11-1;

y=x1-1;

tmp.x=x;

tmp.y=y;

tmp.d=-1;

tmp.kk=k;

q.push(tmp);

while(first<=last&&!q.empty())

{

if(res)

break;

else

{

tmp=q.front();

q.pop();

first++;

x=tmp.x;

y=tmp.y;

d=tmp.d;

lastd=d;

k=tmp.kk;

if(x==y2-1&&y==x2-1&&f[x-1][y-1]=='.')

{

res=1;

break;

}

for(j=0;j<4;j++)//上，下，右，左

{

xx=x+fdr[j];

yy=y+fdc[j];

if(xx==y2-1&&yy==x2-1&&f[xx][yy]=='.')

{

if(lastd==j&&k>=0)

{

res=1;

break;

}

if(lastd!=j)

{

k--;

if(k>=0)

{

res=1;

break;

}

else

{

flag[xx][yy]=0;//释放关节点

continue;

}

}

}

if(xx>=0&&xx<m&&yy>=0&&yy<n&&!flag[xx][yy]&&f[xx][yy]=='.'&&k>=0)

{

flag[xx][yy]=1;

tmp.x=xx;

tmp.y=yy;

tmp.d=j;

if(j!=lastd)

{

if(lastd!=-1)

{

tmpk=k-1;

if(tmpk<0)

continue;

else

{

tmp.kk=tmpk;

}

}

else

tmp.kk=k;

}

else

tmp.kk=k;

q.push(tmp);

last++;

}

}

}

}

if(res)

printf("yes\n");

else

printf("no\n");

}

return 0;

}

正解：

//bfs 93ms速度快于dfs

#include<iostream>

#include<cstring>

#include<cstdio>

#include<queue>

#include<bits/stdc++.h>

using namespace std;

#define INF 0x0fffffff

#define MANX INF

int m,n;

char a[110][110];

int vis[110][110];

int k,sx,sy,ex,ey;

int dir[4][2]={0,1,0,-1,1,0,-1,0};

struct node{

int x;

int y;

int cur;//当前点的转弯数

}now,nex;

void bfs(int x,int y)

{

memset(vis,0,sizeof(vis));

queue<node>s;

now.cur=-1;//起点不计转向

now.x=x;

now.y=y;

vis[x][y]=1;

s.push(now);

while(!s.empty())

{

now=s.front();

if(now.cur>=k)break;//0~k转弯数判断，还在范围内就继续执行，因为是一搜到底，所以k在下面的代码中不会超出k

s.pop();

for(int i=0;i<4;i++)

{

nex.x=now.x+dir[i][0];

nex.y=now.y+dir[i][1];

nex.cur=now.cur+1;////这里一定会改变方向，（因为要做到的是一搜到底）起点-1，所以恰好是0，即没算

while(1)//一搜到底

{

if(nex.x>=0&&nex.x<m&&nex.y>=0&&nex.y<n&&a[nex.x][nex.y]=='.')//忽略重复访问的点

{

if(nex.x==ex&&nex.y==ey)

{

cout<<"yes"<<endl;

return;

}

if(vis[nex.x][nex.y]==0)//重复点只入队一次，但可以多次经过

{

vis[nex.x][nex.y]=1;

s.push(nex);

}

nex.x+=dir[i][0];

nex.y+=dir[i][1];

}

else

break;

}

}

}

cout<<"no"<<endl;

}

int main()

{

int T;

cin>>T;

while(T--)

{

cin>>m>>n;

for(int i=0;i<m;i++)

for(int j=0;j<n;j++)

{

cin>>a[i][j];

/\*\*C语言的char[][]这样输入，不然会出错

for(int i=0;i<m;i++)

{

scanf("%s",a[i]);

}\*/

}

cin>>k>>sy>>sx>>ey>>ex;

sy--;

sx--;

ey--;

ex--;

bfs(sx,sy);

}

return 0;

}

DFS版：

//dfs 124ms

#include<iostream>

#include<cstring>

#include<cstdio>

#include<queue>

using namespace std;

#define INF 0x0fffffff

#define MAXN INF

int m,n;

char a[110][110];

int wan[110][110];//拐弯数

int k,sx,sy,ex,ey;

int fang[4][2]={0,1,0,-1,1,0,-1,0};

int flag;

void dfs(int x,int y,int dir)

{

int tx,ty;

if(x==ex&&y==ey)

{

if(wan[x][y]<=k)//到达

{

flag=1;

}

return;

}

if(wan[x][y]>k)

return;

for(int i=0;i<4;i++)

{

tx=x+fang[i][0];

ty=y+fang[i][1];

//剪枝.3,4步剪枝会直接影响结果正确性。去掉就会WA,是因为当前差解覆盖了比它优的解造成无解，也就是当前情况破坏了可能解的路径。充分说明了剪枝的重要性。

if(tx<0||tx>=m||ty<0||ty>=n)

continue;

if(a[tx][ty]=='\*')

continue;

/\*--------------核心-------------实际上本题的DFS不同于一般dfs，因为我们这里没设置vis[][]标记刚走过的路，所以意味着每个搜索分支都可以访问所有别的分支访问过的点(保证了搜索的任意性)，这样就可以不停更新出最优路径，同时这也使得题的dfs可以走回头路！这会造成死循环，所以这里用了另一种记录来防止回头，就是核心的第二条，若回头了，则i!=dir一定成立，所以当前wan[x][y]+1一定会大于wan[tx][ty],从而这一步就被剪掉了保证了DFS不会死住。 ------\*/

if(dir!=-1&&i==dir&&wan[tx][ty]<wan[x][y])//下一步转弯数小于前一步转弯数,说明此位置已遍历且更优，无须再遍历(不转弯情形)(红字去掉也对)

continue;

if(dir!=-1&&i!=dir&&wan[tx][ty]<wan[x][y]+1)//若下一步原有拐弯数比当前步进入该处拐弯数小,说明已遍历且更优，无须再遍历(转弯情形)(实际上也是为了防止回头死循环，一石二鸟，厉害)

continue;

/\*----------------------------------------\*/

if(dir!=-1&&i!=dir)

wan[tx][ty]=wan[x][y]+1;

else

wan[tx][ty]=wan[x][y];

dfs(tx,ty,i);

if(flag)

return;

}

}

int main()

{

int T;

cin>>T;

while(T--)

{

cin>>m>>n;

for(int i=0;i<m;i++)

for(int j=0;j<n;j++)

{

cin>>a[i][j];

}

cin>>k>>sy>>sx>>ey>>ex;

sy--;

sx--;

ey--;

ex--;

for(int i=0;i<m;i++)

for(int j=0;j<n;j++)

{

wan[i][j]=MAXN;//初始化所有点的转弯数为MAXN.后面剪枝会用

}

wan[sx][sy]=0;//初始

flag=0;

dfs(sx,sy,-1);

if(flag)

cout<<"yes"<<endl;

else

cout<<"no"<<endl;

}

return 0;

}