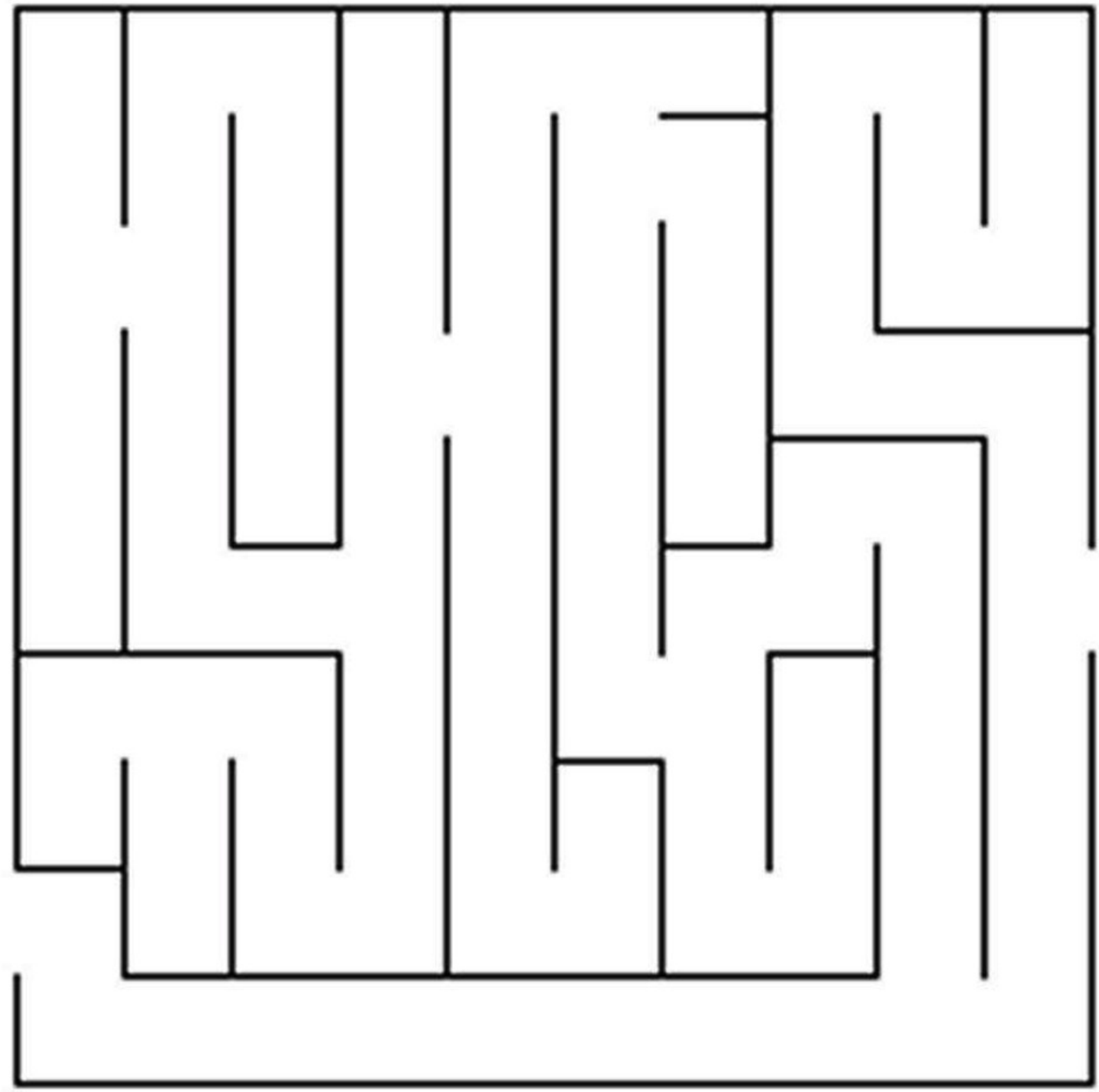
算法引入

小时候，我们会玩一种迷宫游戏。下面就是非常简单的一种：



（本文仅该图来源于网络）

完成该迷宫可以有很多种方法：

- 正向搜索
- 逆向倒推

不管使用哪一种方法，如果遇到走不通的路，我们都会到上一步的位置，然后重新寻找。只要反复地使用这种方法，我们就能够走出迷宫。

在OI中，我们会把迷宫进行抽象化，例如用 # 表示障碍， . 表示可走的位置，当然这样的操作只能还原普通的矩形迷宫（~~当然奇形怪状的迷宫就不行了~~）。

例如这个迷宫（数字表示长与宽）：

```
3 5
. . # . #
. . . . .
# . . # .
```

另外，给出迷宫随机生成代码。

假如在这个迷宫中，只能上下左右地移动，那么我们可以通过$(1,1) \to (2,1) \to (2,2) \to (2,3) \to (2,4) \to (2,5) \to (3,5)$的路径抵达终点。

我们可以把迷宫保存在一个二维字符数组$maps_{i,j}$中。对于这四个方向，横纵坐标会做相应的变化：

- 上$\to (-1,0)$
- 下$\to (1,0)$
- 左$\to (0,-1)$
- 右$\to (0,1)$

当然，有的题目还可以使用左上、左下、右上、右下，横纵坐标这时会这样变化：

- 左上$\to (-1,-1)$
- 左下$\to (1,-1)$
- 右上$\to (-1,1)$
- 右下$\to (1,1)$

为了方便表示，我们可以维护两个一维数组来保存这些横纵坐标的变化量，例如，如果只保存上下左右的话：

```
int dx[]={-1,1,0,0},dy[]={0,0,-1,1};
```

假如八个方向都保存，则有：

```
int dx[]={-1,1,0,0,-1,1,-1,1},dy[]={0,0,-1,1,-1,-1,1,1};
```

假设当前在地图中的位置为(x,y)，则我们可以进行一个循环，每一次在(x,y)的基础上增加(dx_i,dy_i)，得到新的位置$(nx,ny)=(x+dx_i,y+dy_i)$。当然，还得考虑边界问题和题目中的障碍等问题。

接下来就粗略地复习一下两种搜索：

深度优先搜索

第一种搜索叫做深度优先搜索，简称深搜，英文缩写为DFS。DFS适用于找到**第一个符合要求的答案**。DFS被形象地称为不撞南墙不回头的**一根筋**。之前的地图问题就适合用该方法。

该方法的模板：

```
void dfs(参数表)
{
    if(到达终点状态)
    {
        输出或保存结果;
        退出该层循环或整个程序;
    }
    for(拓展方式)
    {
        if(该步合法)
        {
            修改、计算、标记;
            dfs(新的参数表);
            回溯;(可选)
        }
    }
}
```

广度优先搜索

有深度，就有广度。广度优先搜索简称广搜，英文缩写为BFS。BFS适用于寻找**最优解**，个人认为要比DFS要好用一些，但是需要维护队列，而队列的维护可以有两种：

- 数组维护
- STL:queue 维护

队列用数组维护一般来说效率会高一些，但是要控制空间大小，不宜过小（否则会RE），也不宜过大（否则会MLE）。

而用STL中的queue的话，可以不考虑空间，但是时间会略慢一些（恐怕所有STL均是如此）。

数组维护的模板：

```
队列类型 q;//一般来说队列用q表示
int front=1,rear=1;//先定义头指针和尾指针，一开始一般都为1
while(front<=rear)
{
    队列类型 qf=q[front];//为了方便，可以保存头指针的队列元素
    for(拓展方式)
    {
        if(该步合法)
        {
            修改、计算、标记;
            q[++rear]=...;//入队
            if(到达终点状态)
            {
                进行重赋值或直接输出并退出;
            }
        }
    }
    front++;//头指针右移
}
```

queue维护的模板：

```
queue<队列类型>q;
while(q.size())//只要队列中有元素就继续操作
{
    队列类型 qf=q.front();//q.front()为头指针元素
    q.pop();//直接弹出队头
    for(拓展方式)
    {
        if(该步合法)
        {
            修改、计算、标记;
            q.push(...);//入队
            if(到达终点状态)
            {
                进行重赋值或直接输出并退出;
            }
        }
    }
}
```

应用与练习

配套题单

这一次我们将练习下列题目：

题目	难度	来源	算法

题目	难度	来源	算法
P1443 马的遍历	普及/提高-	无	BFS
P1135 奇怪的电梯	普及/提高-	无	BFS
P1019 单词接龙	普及/提高-	NOIP2000	DFS+字符串处理
U119869 迷宫	普及/提高-	无	DFS+路径保存
U119904 迷宫（加强版）	普及/提高-	无	记忆化DFS+路径保存
P2895 Meteor Shower	普及/提高-	USACO2008	BFS
P1825 Corn Maze	普及+/提高	USACO2011	BFS

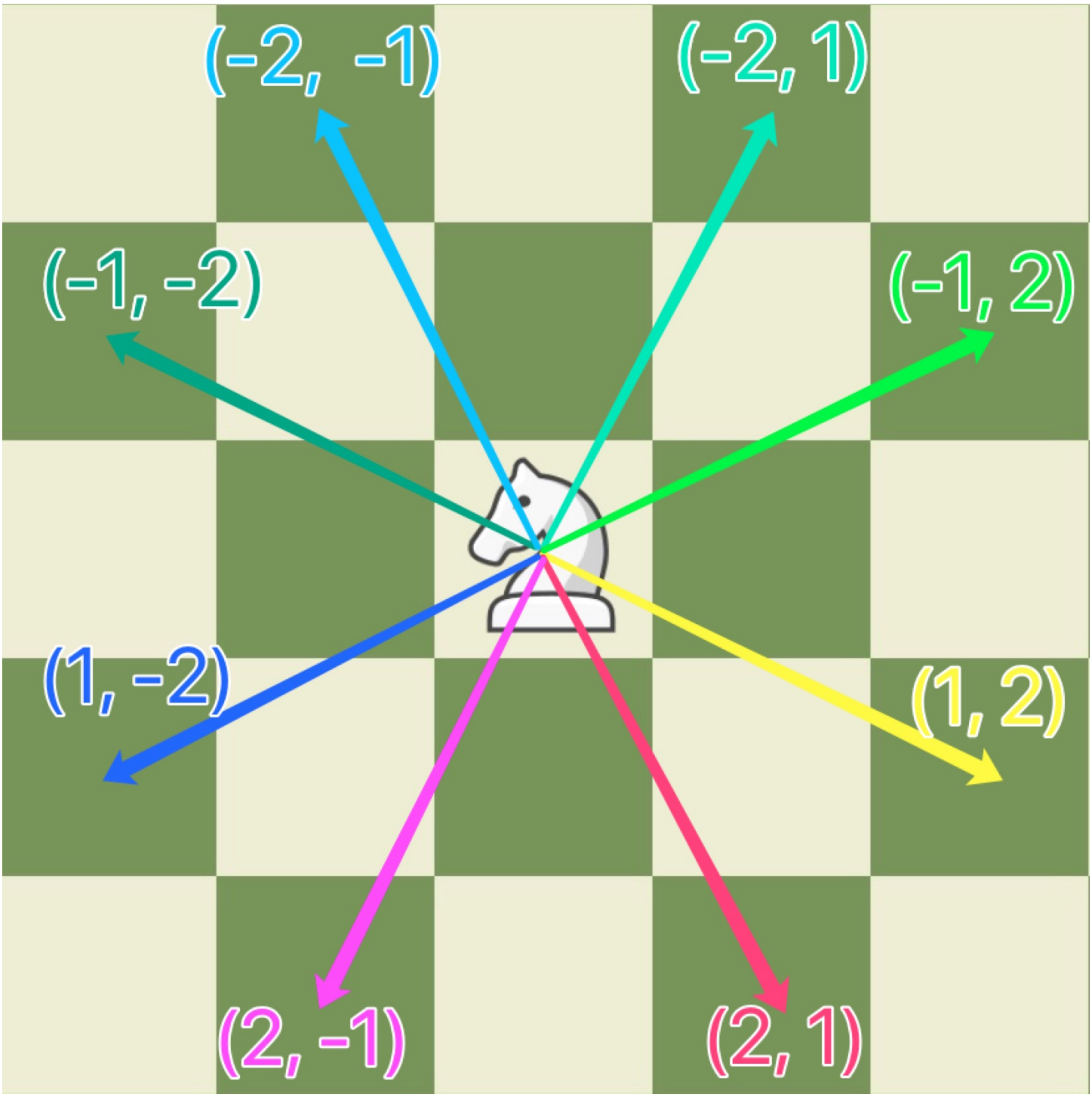
No.1 马的遍历

难度：★★☆

题目链接

这道题可以用宽搜来做。马的方向增量数组略有不同，我们在处理完之后，在宽搜的过程中，标记马需要到达对应位置的步数。

马的移动路径是本题的一个关键：



通过上图，我们可以写出增量数组：

```
int dx[]={-2,-2,-1,-1,1,1,2,2},dy[]={-1,1,-2,2,-2,2,-1,1};
```

Solution 1

期望得分：30分

用DFS的记忆化搜索进行保存和剪枝。

```
#include<bits/stdc++.h>
using namespace std;
int n,m,sx,sy,ans[401][401],dx[]={-2,-2,-1,-1,1,1,2,2},dy[]={-1,1,-2,2,-2,2,-1,1};
bool vis[401][401];
void dfs(int x,int y,int step)
{
    if(step>ans[x][y])return;//剪枝
    ans[x][y]=step;//标记步数
    for(int i=0;i<8;i++)
    {
        int nx=x+dx[i],ny=y+dy[i];
        if(nx<1||ny<1||nx>n||ny>m||vis[nx][ny])continue;
        vis[nx][ny]=true;//标记访问
        dfs(nx,ny,step+1);//继续搜索
        vis[nx][ny]=false;//回溯
    }
}
int main()
{
    memset(ans,0x3f,sizeof(ans));//初始步数为无限大
    scanf("%d%d%d%d",&n,&m,&sx,&sy);
    ans[sx][sy]=0;//初始格子标记为0
    vis[sx][sy]=true;//初始格子标记访问
    dfs(sx,sy,0);//开始搜索
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            if(ans[i][j]==0x3f3f3f3f)printf("%-5d",-1);//判断是否能访问到
            else printf("%-5d",ans[i][j]);
        }
        putchar('\n');
    }
    return 0;
}
```

Solution 2

期望得分：100分

用BFS的思想来做，因为BFS适合做最优解的题目，比DFS的效率要高很多。我们需要对BFS的队列进行维护，并且要注意各项细节。

代码：

```
#include<bits/stdc++.h>
using namespace std;
int n,m,dx[]={-2,-2,-1,-1,1,1,2,2},dy[]={-1,1,-2,2,-2,2,-1,1},front=1,rear=1,ans[401][401];
//dx,dy为马的方向增量数组，头、尾指针定义为1，ans数组保存输出的矩阵
bool vis[401][401];//保存是否被访问过
struct node
{
    int x,y,step;
}q[160001];//用结构体当做队列元素
int main()
{
    scanf("%d%d%d%d",&n,&m,&q[1].x,&q[1].y);
    for(int i=1;i<=n;i++)for(int j=1;j<=m;j++)ans[i][j]=-1;//全部置为-1（即假设无法遍历到）
    ans[q[1].x][q[1].y]=0;//把开始坐标的答案置为0
    vis[q[1].x][q[1].y]=true;//标记开始坐标为访问过
    while(front<=rear)
    {
        node qf=q[front];
        for(int i=0;i<8;i++)
        {
            int nx=qf.x+dx[i],ny=qf.y+dy[i];
            if(nx<1||ny<1||nx>n||ny>m||vis[nx][ny])continue;
            q[++rear]=(node){nx,ny,qf.step+1};//入队
            ans[nx][ny]=q[rear].step;//ans数组赋值
            vis[nx][ny]=true;//标记访问
        }
        front++;
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)printf("%-5d",ans[i][j]);//注意格式化输出方式
        putchar('\n');
    }
    return 0;
}
```

No.2 奇怪的电梯

难度：★★★

题目链接

这道题DFS和BFS都可以使用。其中DFS要用记忆化。

Solution 1

期望得分：100分

给DFS配上记忆化才能够让效率更高。这道题相比上一题，范围较小，转移不多，所以DFS可以过。

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,a,b,k[201],ans[201];
//ans[i]保存到达第i层需要按按钮的次数
void dfs(int cur,int step)
{
    if(step>ans[cur])return;//剪枝
    ans[cur]=step;//把到达该楼层所需按按钮次数赋值为当前的step
    int dx[]={k[cur],-k[cur]};//增量数组
    for(int i=0,nx;i<2;i++)
    {
        nx=cur+dx[i];//即将到达的新楼层
        if(nx>n||nx<1)continue;//进行范围的判断
        dfs(nx,step+1);//下一层搜索继续
    }
}
int main()
{
    memset(ans,0x3f,sizeof(ans));
    //初始把按按钮次数赋值为无限大，默认都不能到达
    scanf("%d%d%d",&n,&a,&b);
    for(int i=1;i<=n;i++)scanf("%d",&k[i]);
    dfs(a,0);//开始搜索
    if(ans[b]==0x3f3f3f3f)puts("-1");//如果无法到达（即次数为无限大），则输出-1
    else printf("%d",ans[b]);//否则输出最少的次数
}
```

Solution 2

期望得分：100分

至于BFS，我们可以这样来写：

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,goal,front=1,rear=1,a[201];
bool vis[201];//标记当前楼层是否已经到达
struct node
{
    int floor,step;
}q[2001];
int main()
{
    scanf("%d%d%d",&n,&q[1].floor,&goal);
    if(goal==q[1].floor)//特判一定要注意!
    {
        putchar('0');
        return 0;
    }
    for(int i=1;i<=n;i++)scanf("%d",&a[i]);//输入对应楼层按钮上的数字
    while(front<=rear)
    {
        node f=q[front];
        int d[]={a[f.floor],-a[f.floor]};
        /*
        方向增量数组:
        1. 向上a[f.floor]楼
        【前提条件】目标楼层小于等于n楼
        2. 向下a[f.floor]楼
        【前提条件】目标楼层大于0楼
        */
        for(int i=0;i<2;i++)
        {
            node r=(node){f.floor+d[i],f.step+1};
            //即将要入队的元素
            if(r.floor>n||r.floor<1||vis[r.floor])continue;
            //判断是否满足前提条件
            q[++rear]=r;//入队
            vis[r.floor]=true;//标记访问
            if(r.floor==goal)//判断是否到达目标楼层
            {
                printf("%d",r.step);
                return 0;
            }
        }
        front++;//头指针+1
    }
    puts("-1");//无法到达就输出-1
    return 0;
}
```

这道题的BFS写法有两个关键点:

- 特判起始楼层 = 目标楼层
- 每次判断是否超过楼层的限制

其中，特判起始楼层与目标楼层相等是在BFS时先拓展，后判断的情况下才需要的。如果不特判，那么就只能拿到90分。我们通过WA自动机得到的样例1来分析。

```
【样例输入】
10 1 1
5 5 5 5 5 5 5 5 5 5
【样例输出】
0
```

这时，初始楼层为1。如果按照我们上述代码中的方法，那么就将先进行拓展：$1 \to 6$，$1 \to -4$（舍去）。

接着，我们就到达了6楼。拓展：$6 \to 11$（舍去），$6 \to 1$。此时发现已经到达目标点，于是输出次数2，很明显与答案0不符。

当然，这在本题的DFS代码中是不需要考虑的。第一次进入搜索后，该位置的步数会被标记为0，因而后续的步数都必定大于这个数。最终该位置的步数就不会被更新，所以最后输出的结果是不受影响的。

No.3 单词接龙

题目链接

难度：★★★★

这道题考察了DFS和字符串的结合。

代码：

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,ans,t[21];//t数组保存使用次数
char ch;//保存开头字母
string s[21];//保存字符串
string transfer(string a,string b)//判断是否可以转换+如果可以就直接转换
{
    for(int i=0,j=a.size()-1;a[i]&&b[i];i++,j--)//新建两个指针，分别在b和a上
    {
        string na=a.substr(j),nb=b.substr(0,i+1);//取其子串
        if(na==nb&&na!=a&&nb!=b)return a+b.substr(i+1);//如果两子串相等且没有包含关系就可以拼接—直
    }
    return "ERROR";//因为正常情况没有大写字母，所以返回ERROR没有影响
}
void dfs(string cur)
{
    for(int i=1;i<=n;i++)
    {
        if(t[i]>1)continue;//已经使用过2次就不能再继续使用
        string ns=transfer(cur,s[i]);//把新拼接好的字符串赋值给ns
        if(ns=="ERROR")continue;//如果等于ERROR就说明无法拼接
        t[i]++;//使用次数加1
        dfs(ns);//继续深搜
        t[i]--;//回溯!
    }
    ans=max(ans,int(cur.size()));//取ans和当前字符串大小的最大值
}
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)cin>>s[i];
    cin>>ch;
    for(int i=1;i<=n;i++)
    {
        if(s[i][0]==ch)
        {
            memset(t,0,sizeof(t));//使用次数清零
            t[i]++;//该字符串开头已经使用了一次
            dfs(s[i]);//开始深搜
        }
    }
    cout<<ans;
    return 0;
}
```

No.4 迷宫

难度：★★★★

这两道题都是输出路径的题目，数据自造。DFS可以完美解决。

普通版

题目链接

Special Judge 代码

由于该题引入了Special Judge，因此对程序的要求是并不大的。路径我们可以用数组来保存，而枚举可以用DFS实现。

Solution 1

期望得分：10分

直接输出-1。

```
#include<bits/stdc++.h>
int main()
{
    puts("-1");
    return 0;
}
```

Solution 2

期望得分：100分

用DFS进行搜索，并把拓展方式（而并非路径）用数组保存，最后写一个打印路径的函数即可。

```cpp
#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];
//增量数组不必考虑顺序，因为有SPJ
bool vis[21][21];
void print(int k)
{
    cout<<"(1,1)";
    int a=1,b=1;//定义两个变量，用来输出路径
    for(int i=1;i<k;i++)
    {
        a+=dx[route[i]],b+=dy[route[i]];
        //每一次a和b加上增量数组对应的坐标
        cout<<"->("<<a<<','<<b<<')';
        //格式化输出
    }
    exit(0);//输出之后，退出整个程序
}
void dfs(int dep)
{
    if(x==n&&y==m)//如果到达了终点就是输出路径
    {
        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])
        {
            vis[nx][ny]=1;//标记访问
            x=nx,y=ny,route[dep]=i;
            //x和y赋值为新位置，然后路径数组设为i即可，表示增量的访问
            dfs(dep+1);//继续下层搜索
            vis[x][y]=0;//回溯
            x-=dx[i],y-=dy[i];//回溯
        }
    }
}
int main()
{
    scanf("%d",&n);
    m=n;//为了方便，定义一个变量m表示宽，使其等于n
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            scanf("%1d",&maps[i][j]);
            /*
            注意输入方式！scanf百分号后加1d表示读入单字符的整型变量
            例如：
            输入为1239848
            scanf("%1d",&x)之后，x被赋值为1
            */
```

```
        }
    }
    vis[1][1]=true;//点(1,1)必须经过，所以这里直接标记访问
    x=y=1;//这里的x和y用作全局变量，表示当前的坐标，初始都为1
    dfs(1);//从第一层开始搜索
    puts("-1");//不能到达终点就输出-1
    return 0;
}
```

加强版

题目链接

这道题要求较高，因为有多组测试数据，还要求最短路径。把时间放宽是为了不卡掉记忆化搜索。

Solution 1

期望得分：10分

该算法有细微的剪枝，但是其效果与没有几乎一样。能AC的只有测试点$1, 2, 3, 4, 6$。开O2对结果没有影响。

```cpp
#include<bits/stdc++.h>
using namespace std;
int t,m,n,x,y,ans,dx[]={0,0,-1,1},dy[]={-1,1,0,0},route[401],r[401],maps[21][21];
bool vis[21][21],flag;
void print(int k)
{
    printf("(1,1)");
    int a=1,b=1;
    for(int i=1;i<=k;i++)
    {
        a+=dx[route[i]],b+=dy[route[i]];
        printf("->(%d,%d)",a,b);
    }
    putchar('\n');
}
void dfs(int dep,int step)
{
    if(step<ans&&x==n&&y==m)//如果到达终点而且步数更少就更新步数和路径，并标记可以走出迷宫
    {
        ans=step;
        flag=true;
        for(int i=1;i<=step;i++)route[i]=r[i];
        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])
        {
            vis[nx][ny]=true;
            x=nx,y=ny,r[dep]=i;
            dfs(dep+1,step+1);
            vis[x][y]=false;
            x-=dx[i],y-=dy[i];
        }
    }
}
int main()
{
    scanf("%d",&t);
    while(t--)
    {
        flag=false;
        memset(vis,false,sizeof(vis));
        ans=0x3f3f3f3f;
        scanf("%d",&n);
        m=n;
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=m;j++)
            {
                scanf("%1d",&maps[i][j]);
            }
        }
```

```
        vis[1][1]=true;
        x=y=1;
        dfs(1,0);
        if(flag)
        {
            printf("%d\n",ans);
            print(ans);
        }
        else puts("-1");
    }
    return 0;
}
```

Solution 2

期望得分：100分

使用记忆化，保存每一个位置步数的最小值。

```cpp
#include<bits/stdc++.h>
using namespace std;
int t,m,n,x,y,ans[21][21],dx[]={0,0,-1,1},dy[]={-1,1,0,0},route[401],r[401],maps[21][21];
bool vis[21][21],flag;
void print(int k)
{
    printf("(1,1)");
    int a=1,b=1;
    for(int i=1;i<=k;i++)
    {
        a+=dx[route[i]],b+=dy[route[i]];
        printf("->(%d,%d)",a,b);
    }
    putchar('\n');
}
void dfs(int dep,int step)
{
    if(x==n&&y==m)
    {
        if(step<ans[x][y])//如果当前步数小于最少步数就执行
        {
            flag=true;//标记能走出迷宫
            ans[x][y]=step;//更新最短路径步数
            for(int i=1;i<=step;i++)route[i]=r[i];//更新最短的路径
        }
        return;
    }
    if(step>ans[x][y])return;//剪枝
    ans[x][y]=step;//更新最少步数
    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])
        {
            vis[nx][ny]=true;
            x=nx,y=ny,r[dep]=i;
            dfs(dep+1,step+1);
            vis[x][y]=false;
            x-=dx[i],y-=dy[i];
        }
    }
}
int main()
{
    scanf("%d",&t);//输入数据组数
    while(t--)
    {
        flag=false;//flag判断是否能够走出迷宫
        memset(vis,false,sizeof(vis));//初始设定都未访问过
        memset(ans,0x3f,sizeof(ans));//初始设定到达对应位置的步数接近为无穷大
        scanf("%d",&n);//输入迷宫大小
        m=n;
        for(int i=1;i<=n;i++)
        {
```

```
            for(int j=1;j<=m;j++)
            {
                scanf("%1d",&maps[i][j]);
            }
        }
        vis[1][1]=true;
        x=y=1;
        dfs(1,0);
        if(flag)//如果能够走出迷宫就执行
        {
            printf("%d\n",ans[n][m]);//输出最少步数+换行
            print(ans[n][m]);//输出最短路径
        }
        else puts("-1");//无法到达终点就输出-1
    }
    return 0;
}
```

No.5 Meteor Shower

题目链接

难度：★★★★☆

这道题考察的是BFS。我们在维护队列的同时，也要开辟一个二维数组来保存对应位置陨石降落的时间（初始为无限大）和对应格子是否已经被访问过。

Solution 1

期望得分：63分

本题是可以用DFS+记忆化的，但效率并不能满足时间限制。

```
#include<bits/stdc++.h>
using namespace std;
int n,dx[]={0,-1,0,0,1},dy[]={0,0,-1,1,0},land[305][305],step[305][305],ans=0x3f3f3f3f;
bool vis[305][305];
void dfs(int x,int y,int t)
{
    if(t>step[x][y])return;//剪枝
    step[x][y]=t;//重新赋值
    if(land[x][y]==0x3f3f3f3f)//保存最小步数并剪枝
    {
        ans=min(t,ans);
        return;
    }
    for(int i=1;i<5;i++)
    {
        int nx=x+dx[i],ny=y+dy[i];
        if(nx<0||ny<0||nx>304||ny>304||vis[nx][ny]||t+1>=land[nx][ny])continue;
        vis[nx][ny]=true;
        dfs(nx,ny,t+1);
        vis[nx][ny]=false;
    }
}
int main()
{
    memset(step,0x3f,sizeof(step));
    memset(land,0x3f,sizeof(land));
    cin>>n;
    for(int i=0,a,b,c;i<n;i++)
    {
        cin>>a>>b>>c;
        for(int i=0;i<5;i++)
        {
            int nx=a+dx[i],ny=b+dy[i];
            if(nx<0||ny<0||nx>304||ny>304)continue;
            land[nx][ny]=min(land[nx][ny],c);
        }
    }
    dfs(0,0,0);//开始搜索
    if(ans==0x3f3f3f3f)puts("-1");//无法到达就输出-1
    else printf("%d",ans);
    return 0;
}
```

Solution 2

期望得分：100分

使用BFS求最优解。

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,dx[]={0,-1,0,0,1},dy[]={0,0,-1,1,0},front=1,rear=1,land[305][305],vis[305][305];
struct node
{
    int x,y,t;
}q[90005];
int main()
{
    memset(land,0x3f,sizeof(land));//land保存的是对应位置陨石下落的时间，初始假定都不下落，所以可以认
    cin>>n;
    for(int i=0,a,b,c;i<n;i++)
    {
        cin>>a>>b>>c;//每次输入三个参数，表示(a,b)位置在c时间会掉落陨石
        for(int i=0;i<5;i++)//注意这里在[0,5)范围，因为0增量数组为(0,0)，包括当前位置
        {
            int nx=a+dx[i],ny=b+dy[i];
            if(nx<0||ny<0||nx>304||ny>304)continue;
            //注意！陨石只能在300以内掉落，但是可能会影响到更远的范围，用304是为了防止RE
            land[nx][ny]=min(land[nx][ny],c);//取掉落时间最小值
        }
    }
    while(front<=rear)
    {
        node qf=q[front];
        for(int i=1;i<5;i++)
        {
            int nx=qf.x+dx[i],ny=qf.y+dy[i];
            if(nx<0||ny<0||nx>304||ny>304||vis[nx][ny]||qf.t+1>=land[nx][ny])continue;
            /*
            满足下列情况的一种则不能继续拓展：
            1、出边界
            2、已经访问过
            3、该地已经被陨石影响
            */
            vis[nx][ny]=true;//标记访问
            q[++rear]=(node){nx,ny,qf.t+1};//入队
            if(land[nx][ny]==0x3f3f3f3f)//如果该地没有陨石降落就说明已经安全
            {
                printf("%d",q[rear].t);
                return 0;
            }
        }
        front++;
    }
    puts("-1");//不能到达安全的格子就输出-1
    return 0;
}
```

当然，由于本题的情况特殊，必须对答案进行寻找，因此还是BFS更合适一些，从时间上最能反映问题。

No.6 Corn Maze

题目链接

难度：★★★★★

这道BFS题的细节是比较多的，但只要我们能够一一处理好，就能够完美地解决该道题单中的“压轴”题。

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,m,sx,sy,dx[]={-1,0,0,1},dy[]={0,-1,1,0},maps[301][301],front=1,rear=1;
/*
对于maps[i][j]:
终点：-1
障碍：-2
传送门：'A'~'Z'所对应的整型值
*/
bool vis[301][301];
struct node
{
    int x,y,step;
}q[90001];
void travel(int &x,int &y)//如果是传送门就可以进行传送，该函数实现了传送功能，即改变横纵坐标
{
    int temp=maps[x][y];
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            if(i==x&&j==y)continue;//不能是相同的位置
            if(temp==maps[i][j])//判断两个传送门是否匹配
            {
                x=i;
                y=j;
                return;
            }
        }
    }
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            char ch;
            cin>>ch;
            switch(ch)
            {
                case '=':
                {
                    maps[i][j]=-1;
                    break;
                }
                case '@':
                {
                    sx=i;
                    sy=j;
                    //对开始坐标进行赋值
                    break;
                }
```

```
                case '.':break;
                case '#':
                {
                    maps[i][j]=-2;
                    break;
                }
                default:maps[i][j]=ch;
            }
        }
    }
    vis[sx][sy]=true;//开始位置要标记访问
    q[1]=(node){sx,sy,0};//初始化
    while(front<=rear)
    {
        node t=q[front];//取队头
        for(int i=0;i<4;i++)
        {
            int nx=t.x+dx[i],ny=t.y+dy[i];
            if(nx>0&&ny>0&&nx<=n&&ny<=m&&maps[nx][ny]!=-2&&!vis[nx][ny])//判断该位置是否满足要求
            {
                vis[nx][ny]=true;//标记访问过
                if(maps[nx][ny]>0)travel(nx,ny);//如果是传送门就传送
                q[++rear]=(node){nx,ny,t.step+1};//入队
                if(maps[q[rear].x][q[rear].y]==-1)//如果是终点就可以结束了
                {
                    printf("%d",q[rear].step);
                    return 0;
                }
            }
        }
        front++;
    }
    return 0;
}
```

总结

搜索的本质实际上是枚举，所以它的时间复杂度并不是很高。但是在竞赛中，它扮演了一个很出色的“骗分者”。当我们不能使用更好的方法时，搜索可以帮助我们拿到部分分，甚至全分。因此，掌握好搜索是至关重要的。

为了让大家能够更方便地找到本文的精华部分，特将其保存在洛谷云剪贴板中。