

认识深度优先搜索



码上得知-1-深度优先搜索的概念



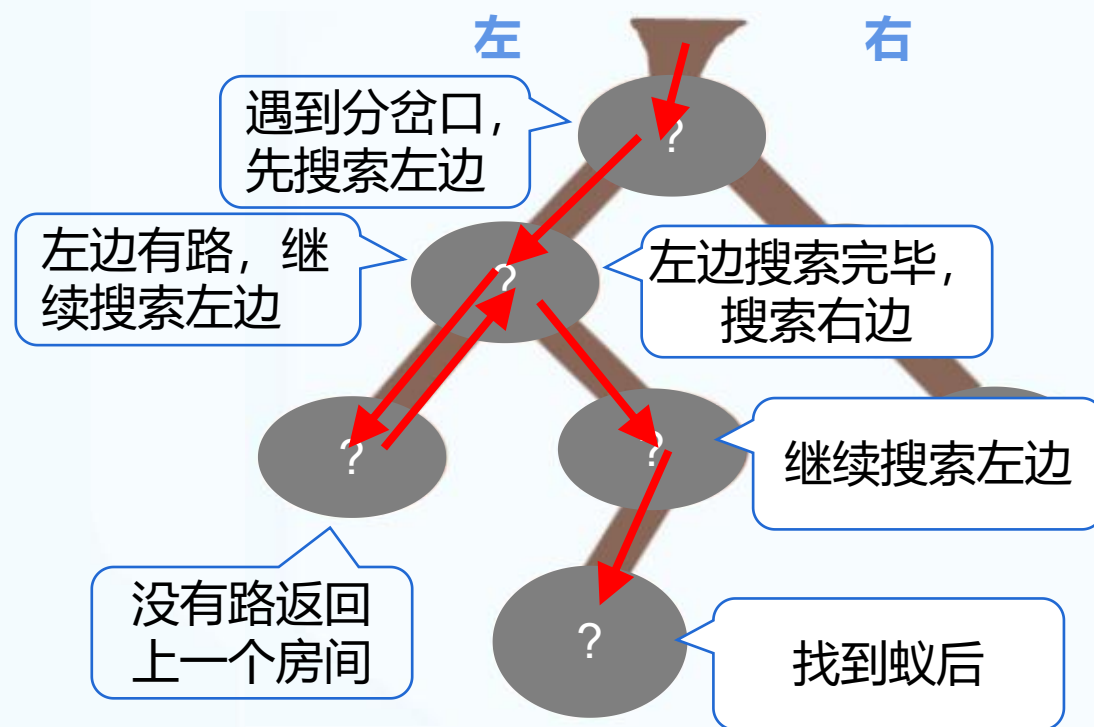
探索蚁巢

蚂蚁是一种团结、勇敢的昆虫。并且每个蚁巢都有一只蚁后。大熊有一只机械蚁，可以进入蚁巢观察蚁后。



探索蚁巢

蚁穴有许多的房间，胡乱走很容易迷路，这里提供了一种搜索方法。

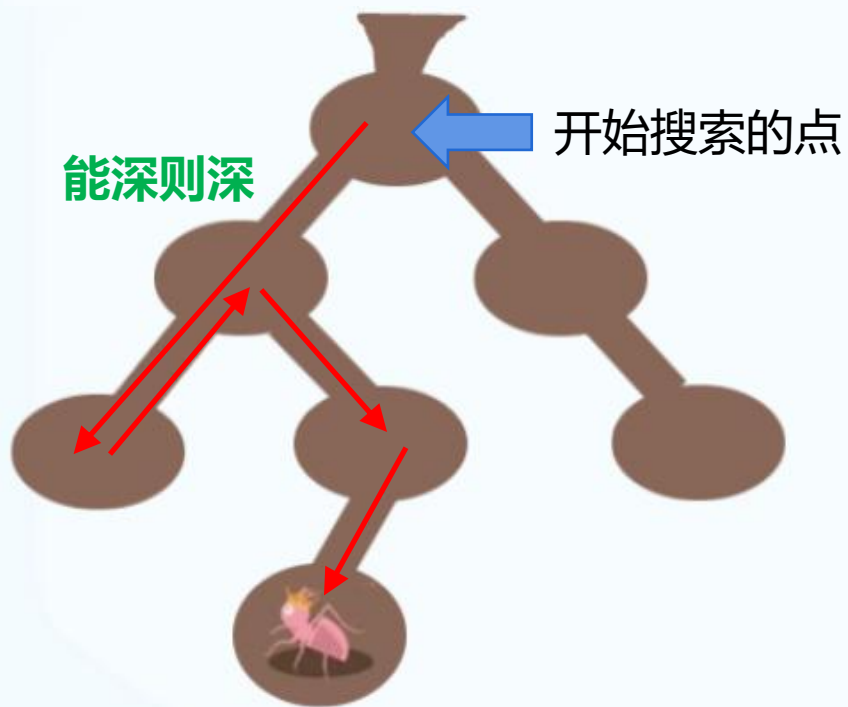


1. 遇到分岔口**先搜索左边，再搜索右边**
2. 没有路时返回上一个房间，换其他路继续搜索
3. 搜索过的房间不再搜索



深度优先搜索

这种搜索方法在计算机中广泛应用。



从一个点开始，按照一定的顺序（先左后右）

沿着某条路往下走，**能深则深**

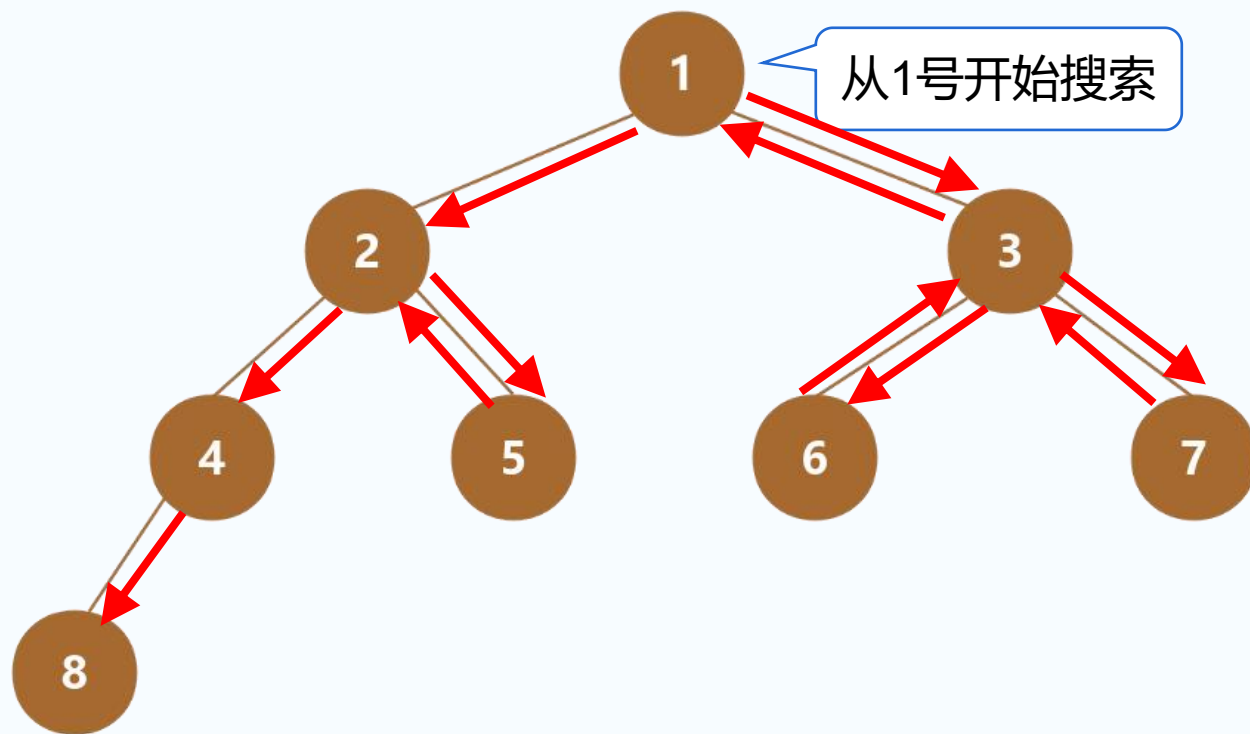
如果发现没有到达目标且没有路可再走，退回上一个点（**不能深则退**），换条路，然后继续走

如此往复，直至可能的结果都被搜索完或到达目标。

这种**能深则深，不能深则退**的方法，称之为**深度优先搜索**。

练一练1

根据深度优先搜索思想，下图按照**先右后左**的顺序从1号开始搜索，最后搜索
8号，搜索的路径是1 3 7 6 2 5 4 8（搜索过的点不再统计）



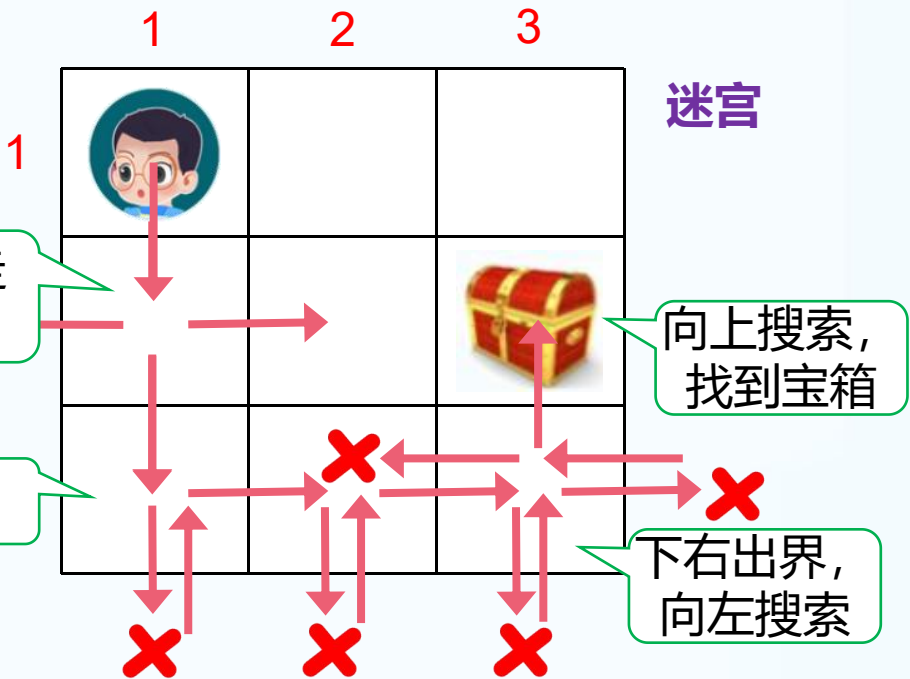
迷宫问题

深搜（深度优先搜索）也可以搜索迷宫。

假设有一个3行3列共9个房间的迷宫，迷宫内有一个宝箱。

每个房间都可以通往周围（上下左右）四个房间，但是不能走出迷宫。

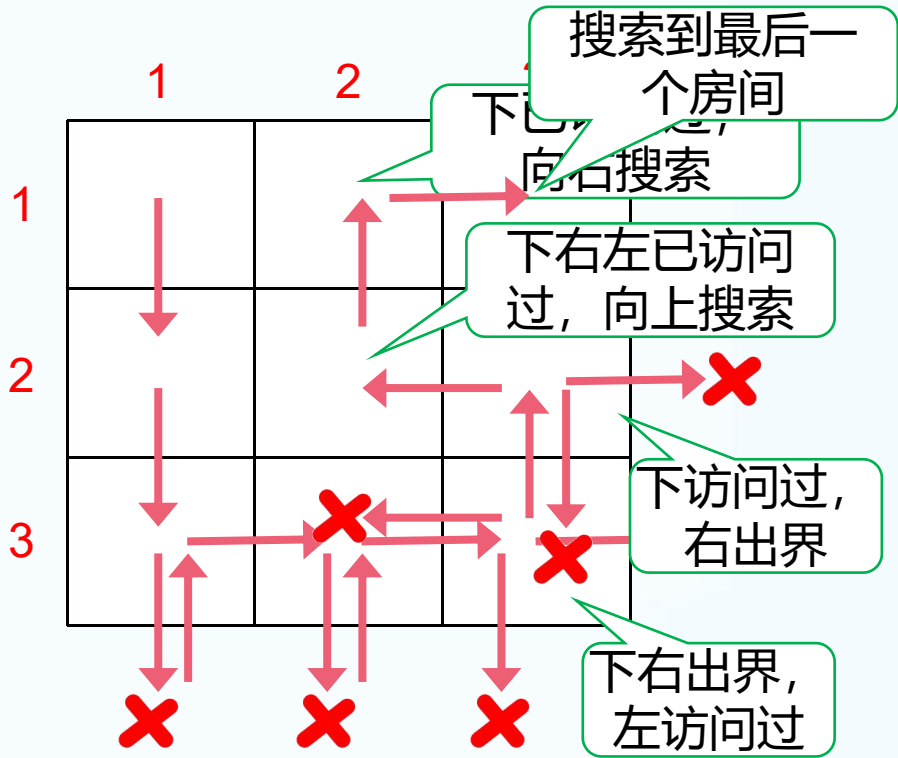
1. 规定一个搜索顺序，例如：**下右左上**
假设人在(1,1)位置，通过
2. 无法搜索时，返回上一个房间换其它方向继续搜索
3. 访问过的房间不再访问



搜索迷宫

如果迷宫没有宝箱，按照刚才的搜索方法，从 (1, 1) 开始搜索，最后搜索的房间是哪一个？

1. 规定一个搜索顺序，例如：下右左上
2. 无法搜索时，返回上一个房间换其它方向继续搜索
3. 访问过的房间不再访问





练一练2

假设有一个3行3列共9个房间的迷宫，从（1，2）房间开始，按照以下条件最后搜索 **A** 房间？

1. 规定一个搜索顺序，例如：下右上左
2. 无法搜索时，返回上一个房间换其它方向继续搜索
3. 访问过的房间不再访问

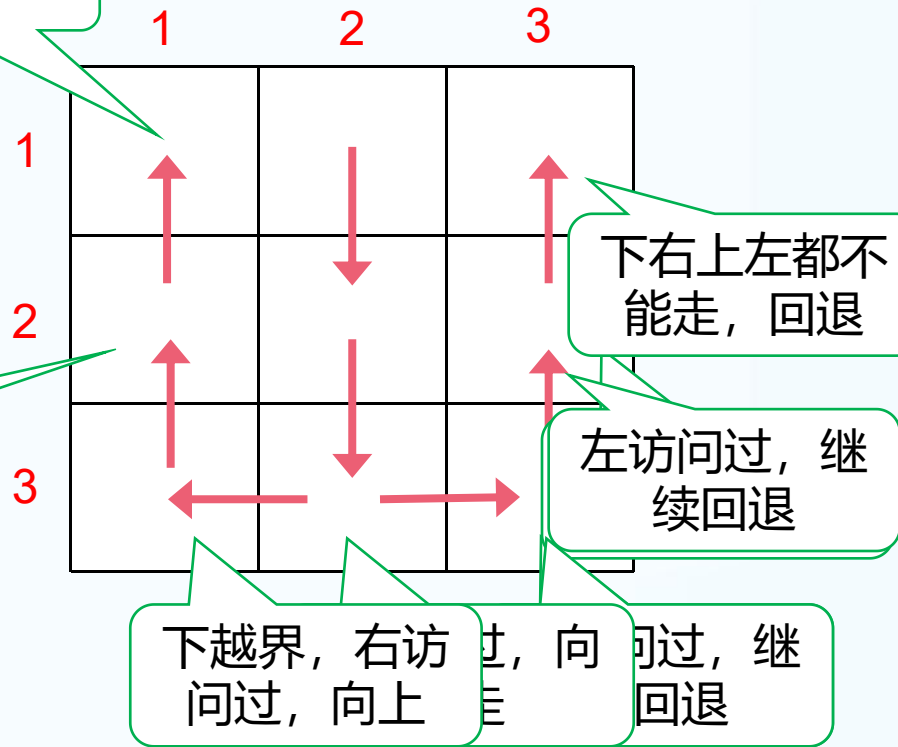
A. (1, 1)

B. (3, 1)

C. (1, 3)

C. (3, 3)

搜索到最后一个房间



下右访问过，
向上

下越界, 右访问过, 向右访问过, 继续
访问过, 向上访问过, 回退

深度优先搜索的概念

从一个点开始，沿着某条路往下走，一直走到底（**能深则深**），如果走完后发现没有走到目标点且没有位置可走，返回到上一个点（**不能深则退**），换其他方向，然后继续走到底，如此往复，直至所有可能的结果都被搜索完。

这种**能深则深，不能深则退**的方法，称之为**深度优先搜索**，简称**DFS**。





码上得知-2-深度优先搜索的实现

问题描述

3行3列共9个房间的迷宫，从 (1, 2) 房间开始，按照以下条件最后搜索 (1, 1) 房间。代码如何实现呢？

- 1. 规定一个搜索顺序，例如：下右上左
- 2. 无法搜索时，返回上一个房间换其它方向继续搜索
- 3. 访问过的房间不再访问

从 (1, 2) 出发，先搜索一个相连的房间，再通过相连的房间搜索下一个房间，一步一步的搜索，每一步的搜索方式都是相同的。

这样的思想特别符合递归思想。

	1	2	3
1			
2			
3			

递归是一个大的问题拆解成若干个小的问题，每个小问题的结构都相同。

递归实现搜索过程

可以自定义dfs()函数，实现深度优先搜索过程。

当前行号

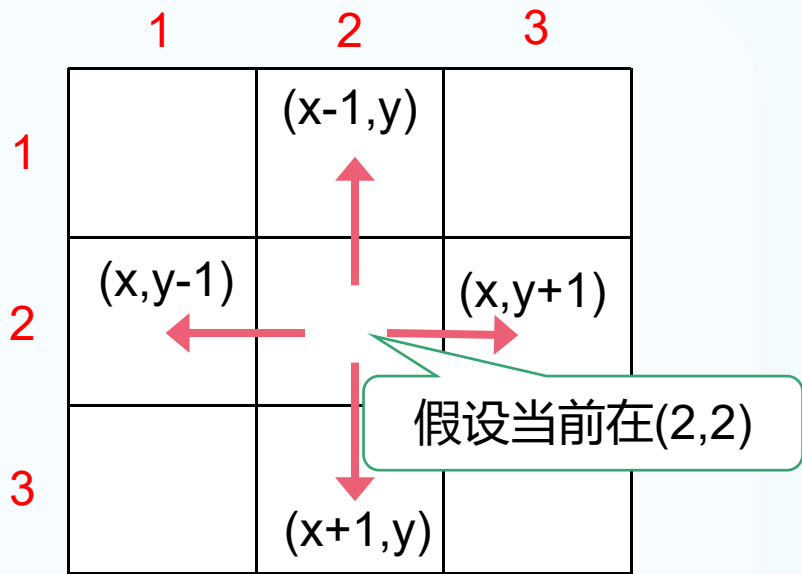
当前列号

```
void dfs(int x, int y){
```

```
    dfs(x + 1, y); // 搜索下面
    dfs(x, y + 1); // 搜索右面
    dfs(x - 1, y); // 搜索上面
    dfs(x, y - 1); // 搜索左边
```

按照下右上的顺序
搜索周围4个方向。

要保证新去的房间在地图内，同时该房间
不能是已经搜索过的房间。



递归实现搜索过程

```
void dfs(int x, int y){
    if( x+1>0&&x+1<=3&&y>0&&y<=3 && vis[x+1][y] == 0 )
        vis[x+1][y] = 1; //能搜索去之前要先标记
        dfs(x + 1, y); //搜索下面
    }

    dfs(x, y + 1); //搜索上面
    dfs(x, y - 1); //搜索左边
}
```

以向下搜索为例。

其他方向同理。

如何表示房间是否搜索过？

	1	2	3
1			
2			
3			

创建一个二维数组，初始化为0。
0表示未搜索，1表示已搜索

vis[][]

递归实现搜索过程

`int vis[4][4];` ← 定义在全局区，所有元素默认为0。

```
void dfs(int x, int y){  
    if(x+1>0&&x+1<=3&&y>0&&y<=3&&vis[x+1][y]==0){  
        vis[x+1][y] = 1; //能搜索 去之前要先标记  
        dfs(x + 1, y); //搜索下面  
    }  
    if(.....){  
        vis[x][y+1] = 1; //能搜索 去之前要先标记  
        dfs(x, y + 1); //搜索右面  
    }  
    if(.....){  
        vis[x-1][y] = 1; //能搜索 去之前要先标记  
        dfs(x - 1, y); //搜索上面  
    }  
    if(.....){  
        vis[x][y-1] = 1; //能搜索 去之前要先标记  
        dfs(x, y - 1); //搜索左边  
    }  
}
```

4个方向都要判断**是否越界**和**是否被搜索过**，有没有什么办法可以简化一下代码呢？



递归实现搜索过程

假设当前点为 $(2, 2)$

向下搜索: $x + 1, y + 0$

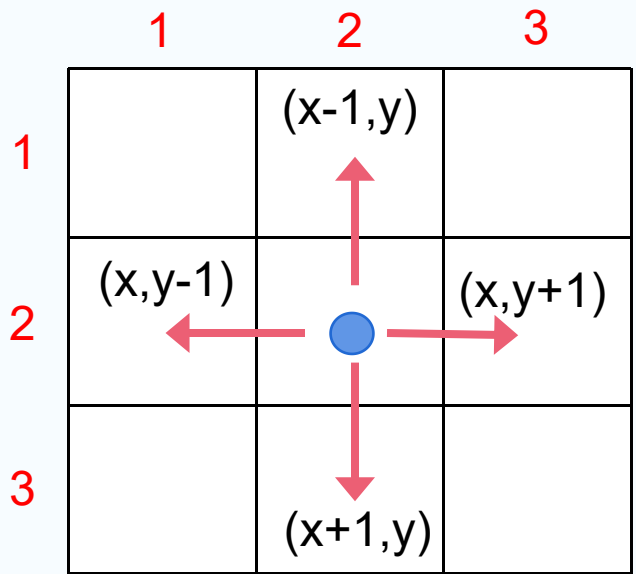
向右搜索: $x + 0, y + 1$

向上搜索: $x + (-1), y + 0 + 0$

向左搜索: $x + 0, y + (-1)$

4个方向要去的点行列下标, 均为当前点的行列下标加上一个数字得到, 所以可以定义数组来存储对应方向行列下标的变化。

加法



递归实现搜索过程

```
int dx[4] = {1, 0, -1, 0}; // 下右上左行下标变化
int dy[4] = {0, 1, 0, -1}; // 下右上左列下标变化
int vis[4][4];
void dfs(int x, int y){
```

可以每搜索1个房间就累加搜索房间个数，到第9个时结束。

```
    for(int i = 0; i < 4; i++){ // 循环枚举4个方向
        int tx = x + dx[i]; // 要去方向点的行下标
        int ty = y + dy[i]; // 要去方向点的列下标
        if(tx < 0 && tx >= 3 && ty < 0 && ty <= 3 && vis[tx][ty] == 0){ // 判断
            vis[tx][ty] = 1; // 已被搜索，去标记
            dfs(tx, ty); // 继续从(tx, ty)搜索
        }
    }
}
```

如何输出最后一个搜索的房间呢？



递归实现搜索过程

```
int ans = 0; // 搜索房间的个数
int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1};
int vis[4][4];
void dfs(int x, int y){
    for(int i = 0; i < 4; i++){ // 循环枚举4个方向
        int tx = x + dx[i], ty = y + dy[i]; // 要去方向点的行列下标
        if(tx > 0 && tx <= 3 && ty > 0 && ty <= 3 && vis[tx][ty] == 0){ // 判断
            vis[tx][ty] = 1; // 能去标记
            dfs(tx, ty); // 继续从(tx, ty)搜索
        }
    }
}
```

应该在主函数
中如何调用
dfs函数呢?



递归实现搜索过程

```
#include<iostream>
using namespace std;
int ans = 0;
int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1};
int vis[4][4];
void dfs(int x, int y){
    .....
}
int main(){
    vis[1][2] = 1;
    dfs(1, 2);
    return 0;
}
```

从(1,2)点出发,
所以要标记点
(1,2), 并调用
dfs函数。



写一写

```
#include<iostream>
using namespace std;
int ans = 0; // 搜索房间的个数
int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1};
int vis[4][4]; // 标记搜索过的房间
void dfs(int x, int y){
    ans++; // 累加搜索房间的个数
    if(ans == 9){ // 到了最后一个房间
        cout << x << ' ' << y;
        return;
    }
    for(int i = 0; i < 4; i++){ // 循环枚举4个方向
        int tx = x + dx[i], ty = y + dy[i]; // 要去方向点的行列下标
        if(tx>0&&tx<=3&&ty>0&&ty<=3&&vis[tx][ty]==0){ // 判断
            vis[tx][ty] = 1; // 能去标记
            dfs(tx, ty); // 继续从(tx, ty)搜索
        }
    }
}
int main(){
    vis[1][2] = 1;
    dfs(1, 2);
    return 0;
}
```

深度优先搜索的模板

```
//方向数组，记录对应方向的行列下标变化
int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1};
int vis[4][4]; //标记数组
void dfs(int x, int y) { //搜索点(x, y)
    if(.....) { //目标判断条件
        //到达目标的执行代码
        return;
    }
    for(int i = 0; i < 4; i++) { //循环枚举4个方向
        int tx = x + dx[i], ty = y + dy[i]; //要去方向点的行列下标
        if(.....) { //判断(tx, ty)不能越界 且 没被搜索过
            vis[tx][ty] = 1; //能去标记
            dfs(tx, ty); //继续从(tx, ty)搜索
        }
    }
}
```

深度优先搜索的应用



迷宫

【问题描述】

有 1 个 $n \times n$ 的迷宫方格，在方格内 “0” 表示可以通行，“1” 表示是障碍物不能通行，在 (n, n) 位置有一个宝箱。现在有个人在左上角 $(1, 1)$ 的位置，他在迷宫内可以向当前位置的上、下、左、右四个方向行走，能不能在迷宫里走到宝箱位置 (n, n) 。

注意：测试数据保证起点和终点均为 “0”，走的过程不能走出迷宫。

【输入描述】

输入第一行为 n ($2 \leq n \leq 10$)，表示 $n \times n$ 的方格，接下来有 n 行，每行 n 个整数，0 表示可以行走，1 表示不能行走，每个整数之间有个空格。

【输出描述】

如果可以走到终点，输出 “YES”，否则输出 “NO”

【样例输入】

```
3
0 0 1
1 0 0
0 1 0
```

【样例输出】

```
YES
```

问题分析

此题从**左上角(1, 1)**搜索到**右下角(n, n)**，每个格子可以**上、下、左、右**四个方向行走。方格为**0**表示可以通行，**1**表示是障碍物不能通行。

起点为(1, 1),
默认起点可以走

	1	2	3
1	0	0	1
2	1	0	0
3	0	1	0

每个位置可以向上下左右四个方向走。**越界、元素值为1、搜索过的点均不可搜索。**

终点为(n, n)

可以使用深度优先搜索模板，目标为搜索到(n, n)点，搜索过程中判断能继续搜索的条件为要去的格子在地图范围内、没被搜索过以及元素值不为1。



解题步骤

1. 定义需要的数组及变量，并在主函数中进行输入。

```
int n, g[11][11]; // 地图大小 以及 地图
int main(){
    cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    return 0;
}
```

解题步骤

2. 定义标记数组和方向数组，并根据题目要求和深搜模板定义并实现深搜函数。

```
bool flag; // 标记是否找到目标点 0: 否, 1: 是
int vis[11][11]; // 标记数组
int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1}; // 方向数组 下右上左
void dfs(int x, int y){
    if(x == n && y == n){ // 到达目标
        flag = 1; // 到了就把flag设置为1
        return;
    }
    for(int i = 0; i < 4; i++){ // 搜索四个方向
        int tx = x + dx[i], ty = y + dy[i]; // 求要去的格子的行列下标
        if(tx > 0 && tx < n && ty > 0 && ty < n && !vis[tx][ty] && g[tx][ty] == 0){
            vis[tx][ty] = 1; // 标记
            dfs(tx, ty); // 继续搜索
        }
    }
}
```

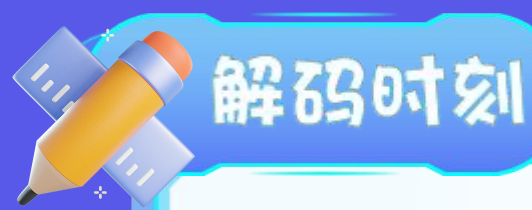
可以定义一个bool类型变量来标记是否找到目标点。



解题步骤

3. 在主函数中调用dfs函数，从(1, 1)开始搜索，并根据flag的值输出。

```
int main(){  
    // 输入  
  
    vis[1][1] = 1;  
    dfs(1, 1);  
    if(flag == 1) cout << "YES";  
    else cout << "NO";  
  
    return 0;  
}
```



【完整代码1】

```
#include<iostream>
using namespace std;
int n, g[11][11]; //地图大小 以及 地图
bool flag; //标记是否找到目标点 0: 否, 1: 是
int vis[11][11]; //标记数组
int dx[4] = {1, 0, -1, 0}, dy[4] = {0, 1, 0, -1}; //方向数组 下右上左
void dfs(int x, int y){
    if(x == n && y == n){
        flag = 1; //到了就把flag设置为1
        return;
    }
    for(int i = 0; i < 4; i++){ //搜索四个方向
        int tx = x + dx[i], ty = y + dy[i]; //求要去的格子的行列下标
        if(tx > 0 && tx <= n && ty > 0 && ty <= n && vis[tx][ty] == 0 && g[tx][ty] == 0){
            vis[tx][ty] = 1; //标记
            dfs(tx, ty); //继续搜索
        }
    }
}
```



【完整代码2】

```
int main(){
    cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    vis[1][1] = 1;
    dfs(1, 1);
    if(flag == 1) cout << "YES";
    else cout << "NO";
    return 0;
}
```

探索迷宫

【题目描述】

有一个 $m*n$ 格的迷宫(表示 m 行、 n 列), 用0表示可以通行, 1表示障碍物不能通行, 从迷宫的 (1,1) 位置开始出发, 到指定的位置停止 (两个数据描述, 分别表示行和列)。走时只能是“上下左右”四个方向。如果无法到达输出“NO” (表示无路), 否则输出“YES”。注意: 第一行第一列元素坐标为 (1,1)。

【输入描述】

第一行是两个数 m, n ($1 < n, m < 20$), 接下来是 m 行 n 列由1和0组成的数据。最后一行表示终点的坐标。

【输出描述】

如果能到达输出YES, 否则输出NO。

【样例输入】

```
5 6
0 0 0 1 0 1
1 1 1 1 0 0
0 0 0 1 1 0
0 0 0 0 0 1
0 0 1 0 1 0
3 3
```

【样例输出】

```
NO
```

问题分析

这道题与上一个迷宫题很类似，**不同点**在于上一题说明了**起点没有障碍物**，本题**并没有说明起点的情况**，所以需要**判断起点是否存在障碍物**，同时本题的**终点是需要输入的**。

起点存在障碍物时，
直接输出NO。

	1	2	3
1	0	0	1
2	1	0	1
3	0	1	0

解题步骤

1. 创建需要的数组及变量。

```
int m, n; // 地图大小
int ex, ey; // 终点坐标
int g[21][21]; // 地图
bool flag; // 标记是否找到目标点
int vis[21][21]; // 标记数组
// 行下标变化下右上左
int dx[4] = {1, 0, -1, 0};
// 列下标变化下右上左
int dy[4] = {0, 1, 0, -1};
```

2. 主函数中读入数据。

```
int main(){
    cin >> m >> n;
    for(int i = 1; i <= m; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    cin >> ex >> ey;
    if(g[1][1] == 1){
        cout << "NO";
        return 0;
    }
}
```

输入后判断起点格子是否为障碍物，是直接输出NO，结束程序。

解题步骤

3. 定义并实现dfs深度优先搜索函数。

```
void dfs(int x, int y){  
    if( x == ex && y == ey ){  
        flag = 1; //到了就把flag设置为1  
        return;  
    }  
  
    for(int i = 0; i < 4; i++){ //搜索四个方向  
        int tx = x + dx[i], ty = y + dy[i]; //求要去的格子的行列下标  
        if( tx > 0 && tx <= m && ty > 0 && ty <= n && vis[tx][ty] == 0 ){  
            vis[tx][ty] = 1; //标记  
            dfs(tx, ty); //继续搜索  
        }  
    }  
}
```

解题步骤

4. 主函数中调用dfs函数，从(1, 1)开始搜索，并根据flag的值输出。

```
int main(){
    cin >> m >> n;
    for(int i = 1; i <= m; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    cin >> ex >> ey;
    if(g[1][1] == 1){
        cout << "NO";
        return 0;
    }
    vis[1][1] = 1;
    dfs(1, 1);
    if(flag == 1) cout << "YES";
    else cout << "NO";
    return 0;
}
```

【完整代码1】

```
#include<iostream>
using namespace std;
int m, n, ex, ey, g[21][21];; //地图大小 终点坐标 地图
bool flag; //标记是否找到目标点
int vis[21][21]; //标记数组
int dx[4] = {1,0,-1,0}; //行下标变化下右上左
int dy[4] = {0,1,0,-1}; //列下标变化下右上左
void dfs(int x, int y){
    if( x == ex && y == ey ){
        flag = 1; //到了就把flag设置为1
        return;
    }
    for(int i = 0; i < 4; i++){ //搜索四个方向
        int tx = x + dx[i], ty = y + dy[i]; //求要去的格子的行列下标
        if( tx>0 && tx<=m && ty>0 && ty<=n && vis[tx][ty]==0 && g[tx][ty]==0 ){
            vis[tx][ty] = 1; //标记
            dfs(tx, ty); //继续搜索
        }
    }
}
```

【完整代码2】

```
int main(){
    cin >> m >> n;
    for(int i = 1; i <= m; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    cin >> ex >> ey;
    if(g[1][1] == 1){
        cout << "NO";
        return 0;
    }
    vis[1][1] = 1;
    dfs(1, 1);
    if(flag == 1) cout << "YES";
    else cout << "NO";
    return 0;
}
```

八个方向

【题目描述】

已知山洞里面是由许多房间组成的迷宫，**每个房间可以通往周围八个房间**，迷宫大小是一个 **$N*N$** 的正方形，其中有一些**蝙蝠堵路**。现在从**起始(1,1)**的位置进入洞穴**寻找宝藏（已有一个宝箱）**，如果可以找到宝藏输出YES，否则输出NO。

【输入描述】

第一行是一个正整数 N ($2 < N \leq 10$)，后面包含 $N*N$ 行由0, 1, 2组成的矩阵，其中**0表示可以走，1表示蝙蝠，2表示宝藏的位置**。

(注意：**第一个房间没有蝙蝠**)

【输出描述】

一行，找到宝藏输出YES，否则输出NO。

【样例输入】

```
5
0 0 0 0 0
0 0 1 1 1
0 0 0 1 0
0 1 0 1 2
0 0 0 0 1
```

【样例输出】

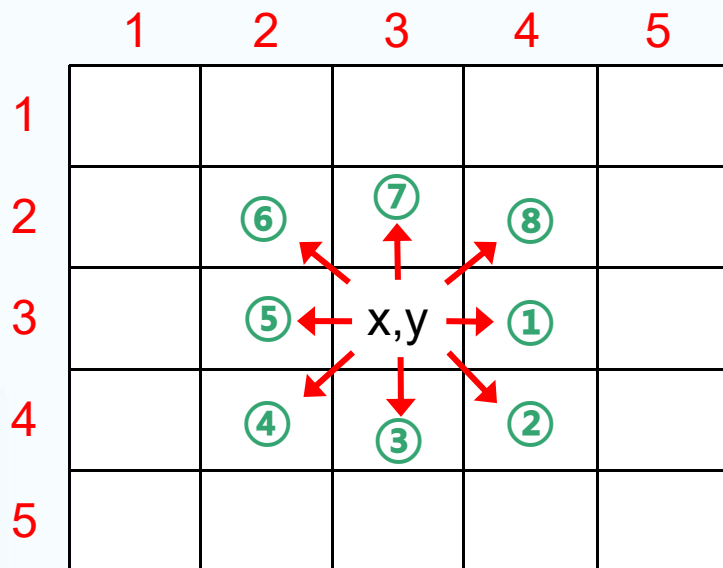
```
YES
```

问题分析

此题需要注意三点：

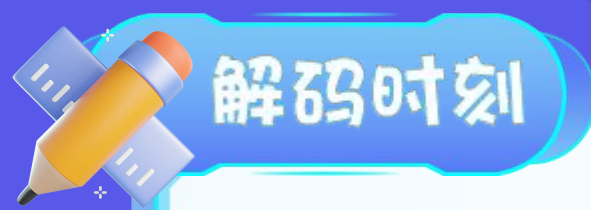
1. 每个房间有8个方向可以搜索。
2. 第一个房间没有蝙蝠，所以不需要判断起点。
3. 终点为地图中值为2的点。

```
//方向 右、右下、下、左下、左、左上、上、右上
//dx存储对应方向行下标变化
int dx[8] = {0,1,1,1,0,-1,-1,-1};
//dy存储对应方向列下标变化
int dy[8] = {1,1,0,-1,-1,-1,0,1};
```



- | | |
|------|----------------|
| ① 右 | $x + 0, y + 1$ |
| ② 右下 | $x + 1, y + 1$ |
| ③ 下 | $x + 1, y + 0$ |
| ④ 左下 | $x + 1, y - 1$ |
| ⑤ 左 | $x + 0, y - 1$ |
| ⑥ 左上 | $x - 1, y - 1$ |
| ⑦ 上 | $x - 1, y + 0$ |
| ⑧ 右上 | $x - 1, y + 1$ |

将减法替换成加法后，
同样可定义方向数组
存行列下标的变化。



解题步骤

1. 定义需要的数组及变量。

```
int n, g[11][11]; //地图大小 地图
bool flag; //标记是否找到目标点
int vis[11][11]; //标记数组
//右 右下 下 左下 左 左上 上 右上
//dx存储对应方向行下标变化
int dx[8] = {0,1,1,1,0,-1,-1,-1};
//dy存储对应方向列下标变化
int dy[8] = {1,1,0,-1,-1,-1,0,1};
```

2. 主函数中输入数据。

```
int main(){
    cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    return 0;
}
```

解题步骤

3. 定义并实现dfs深度优先搜索函数。

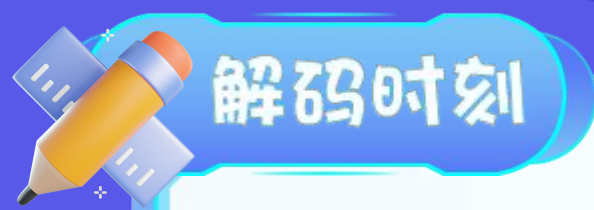
```
void dfs(int x, int y){
    if(g[x][y] == 2){ // 地图元素值为2表示到达终点。
        flag = 1; // 到了就记, flag设置为1
        return;
    }
    for(int i = 0; i < 8; i++){ // 搜索八个方向
        int tx = x + dx[i], ty = y + dy[i]; // 求要去的格子的行列下标
        if( (tx > 0 && tx < n && ty > 0 && ty < m) && !vis[tx][ty] && g[tx][ty] != 1 ){
            vis[tx][ty] = 1; // 标记
            dfs(tx, ty); // 继续搜索
        }
    }
}
```



解题步骤

4. 主函数中调用dfs函数，从(1, 1)开始搜索，并根据flag的值输出。

```
int main(){
    cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    vis[1][1] = 1;
    dfs(1, 1);
    if(flag == 1) cout << "YES";
    else cout << "NO";
    return 0;
}
```



【完整代码1】

```
#include<iostream>
using namespace std;
int n, g[11][11];; //地图大小 地图
bool flag; //标记是否找到目标点
int vis[11][11]; //标记数组
//右 右下 下 左下 左 左上 上 右上
//dx存储对应方向行下标变化
int dx[8] = {0,1,1,1,0,-1,-1,-1};
//dy存储对应方向列下标变化
int dy[8] = {1,1,0,-1,-1,-1,0,1};
void dfs(int x, int y){
    if( g[x][y] == 2 ){
        flag = 1; //到了就把flag设置为1
        return;
    }
    for(int i = 0; i < 8; i++){ //搜索八个方向
        int tx = x + dx[i], ty = y + dy[i]; //求要去的格子的行列下标
        if( tx>0 && tx<=n && ty>0 && ty<=n && vis[tx][ty]==0 && g[tx][ty]!=1 ){
            vis[tx][ty] = 1; //标记
            dfs(tx, ty); //继续搜索
        }
    }
}
```



【完整代码2】

```
int main(){
    cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    vis[1][1] = 1;
    dfs(1, 1);
    if(flag == 1) cout << "YES";
    else cout << "NO";
    return 0;
}
```

中国象棋的马

【题目描述】

马在中国象棋以日字形规则移动，在一个 $n \times m$ 的棋盘上，每个格子内 “0” 表示可以走，“1” 表示是障碍物不能走。马从左上角 (1,1) 位置出发，不考虑掉绊马腿的情况，问能不能走到 (ex,ey) 格子。注意：(1,1) 位置不为1。

【输入描述】

第一行为 n 和 m ，表示 $n \times m$ 的棋盘 ($1 < n, m \leq 10$)。
接下来 n 行，每行 m 个整数，0 表示可以行走，1 表示不能行走，每个整数之间有个空格。
最后一行，表示要到达的格子位置。

【输出描述】

如果可以走到指定位置，输出 “YES”，否则输出 “NO”

【样例输入】

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

【样例输出】

```
YES
```

问题分析

此题目与八个方向题目类似，需要统计马能走的八个位置(马走日)。

假设马的位置为 (x, y) $x = 3, y = 3$

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

- ① $(x - 2, y + 1)$ ⑤ $(x + 2, y - 1)$
- ② $(x - 1, y + 2)$ ⑥ $(x + 1, y - 2)$
- ③ $(x + 1, y + 2)$ ⑦ $(x - 1, y - 2)$
- ④ $(x + 2, y + 1)$ ⑧ $(x - 2, y - 1)$

//dx存储对应方向行下标变化

```
int dx[8] = {-2,-1,1,2,2,1,-1,-2};
```

//dy存储对应方向列下标变化

```
int dy[8] = {1,2,2,1,-1,-2,-2,-1};
```

解题步骤

1. 创建需要的数组及变量。

```
int n, m, g[11][11]; // 地图大小 地图
bool flag; // 标记是否找到目标点
int vis[11][11]; // 标记数组
int ex, ey; // 目标位置
// dx 存储对应方向行下标变化
int dx[8] = {-2, -1, 1, 2, 2, 1, -1, -2};
// dy 存储对应方向列下标变化
int dy[8] = {1, 2, 2, 1, -1, -2, -2, -1};
```

2. 在主函数中输入数据。

```
int main(){
    cin >> n >> m;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            cin >> g[i][j];
        }
    }
    cin >> ex >> ey;
    return 0;
}
```

解题步骤

3. 定义并实现dfs深度优先搜索函数。

```
void dfs(int x, int y){
    if(x == ex && y == ey){ // 到达目标位置
        flag = 1; // 到了就把flag设置为1
        return;
    }
    for(int i = 0; i < 8; i++){ // 搜索八个方向
        int tx = x + dx[i], ty = y + dy[i]; // 求要去的格子的行列下标
        if( tx > 0 && tx <= m && ty > 0 && ty <= n && !vis[tx][ty] && g[tx][ty] == 0 ){
            vis[tx][ty] = 1; // 标记
            dfs(tx, ty); // 继续搜索
        }
    }
}
```

目标位置为(ex, ey)。

解题步骤

4. 主函数中调用dfs函数，从(1, 1)开始搜索，并根据flag的值输出。

```
int main(){
    cin >> n >> m;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            cin >> g[i][j];
        }
    }
    cin >> ex >> ey;

    vis[1][1] = 1;
    dfs(1, 1);
    if(flag == 1) cout << "YES";
    else cout << "NO";
    return 0;
}
```

【完整代码1】

```
#include<iostream>
using namespace std;
int n, m, g[11][11], ex, ey; //地图大小 地图 目标位置
bool flag; //标记是否找到目标点
int vis[11][11]; //标记数组
//dx存储对应方向行下标变化 dy存储对应方向列下标变化
int dx[8] = {-2,-1,1,2,2,1,-1,-2}, dy[8] = {1,2,2,1,-1,-2,-2,-1};
void dfs(int x, int y){
    if(x == ex && y == ey){
        flag = 1; //到了就把flag设置为1
        return;
    }
    for(int i = 0; i < 8; i++){ //搜索八个方向
        int tx = x + dx[i], ty = y + dy[i]; //求要去的格子的行列下标
        if(tx > 0 && tx <= n && ty > 0 && ty <= m && vis[tx][ty] == 0 && g[tx][ty] == 0){
            vis[tx][ty] = 1; //标记
            dfs(tx, ty); //继续搜索
        }
    }
}
```

【完整代码2】

```
int main(){
    cin >> n >> m;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            cin >> g[i][j];
        }
    }
    cin >> ex >> ey;

    vis[1][1] = 1;
    dfs(1, 1);
    if(flag == 1) cout << "YES";
    else cout << "NO";
    return 0;
}
```



统计线路

【题目描述】

一个 $N*N$ 的迷宫方格，在方格内“0”表示可以走，“1”表示不能行走，“2”表示宝藏。现在从**左上角(1,1)**的位置进入迷宫寻找宝藏。统计有多少条线路可以找到宝藏（每条线路经过的格子只能访问1次）。注意：第一个格子不为1。

【输入描述】

第一行，一个正整数 N ($2 < N \leq 10$)，后面包含 $N*N$ 行由0, 1, 2组成的矩阵，其中0表示可以走，1表示不能走，2表示宝藏的位置。

【输出描述】

一行，一个整数，表示可以找到宝藏的线路。

【样例输入】

```
5
0 0 1 1 0
1 0 0 0 0
0 0 0 0 2
0 1 1 0 0
0 0 0 1 0
```

【样例输出】

```
12
```

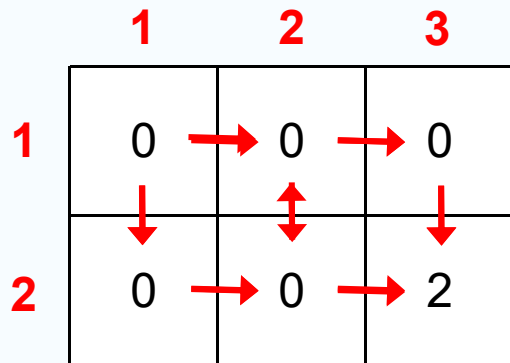


问题分析

本题与前面的题目的不同点在于要统计所有能找到宝藏的路线个数。

假设要搜索的迷宫如下图所示，搜索顺序：下右上左

每条路线上经过的格子只能访问1次，从(1, 1)出发到(2, 3)位置，一共有多少条不同的路线？



`g[][]`

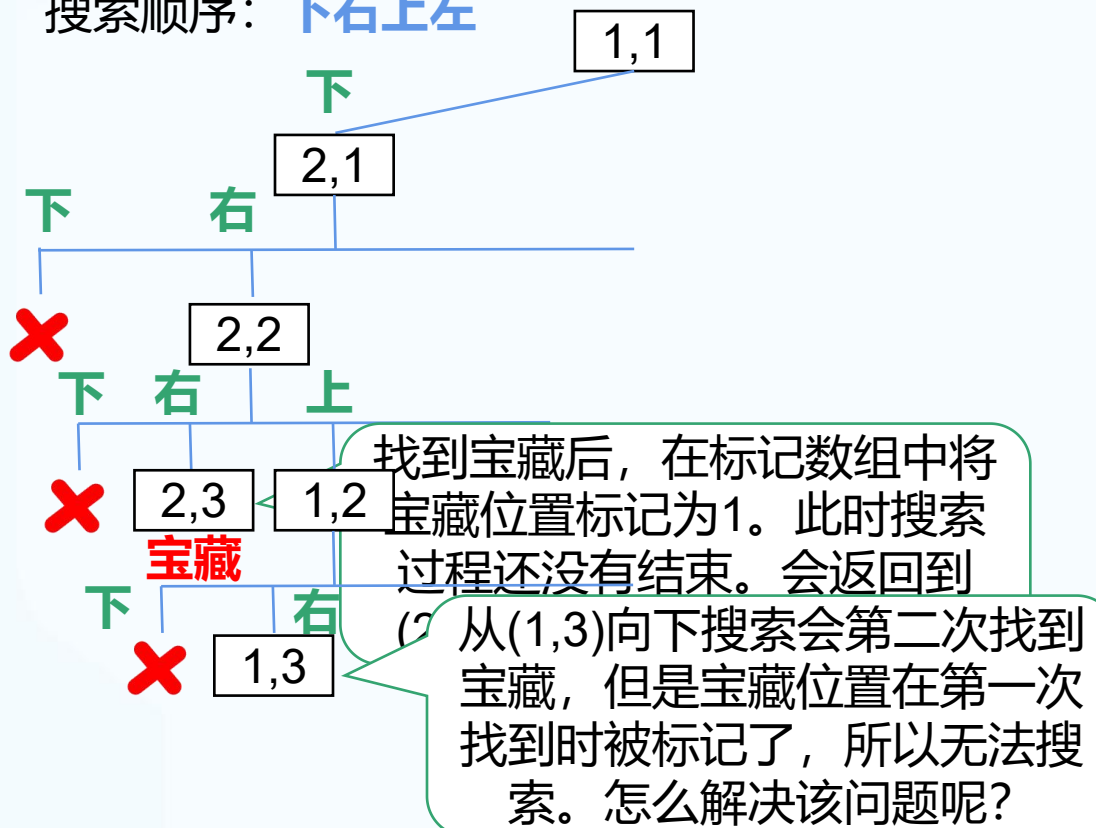
总共有4条路径，
可以到达宝藏位置。



问题分析

我们可以按照之前的深搜模板，模拟下整个递进和回归的搜索过程。

搜索顺序：**下右上左**



起始位置
标记为1。

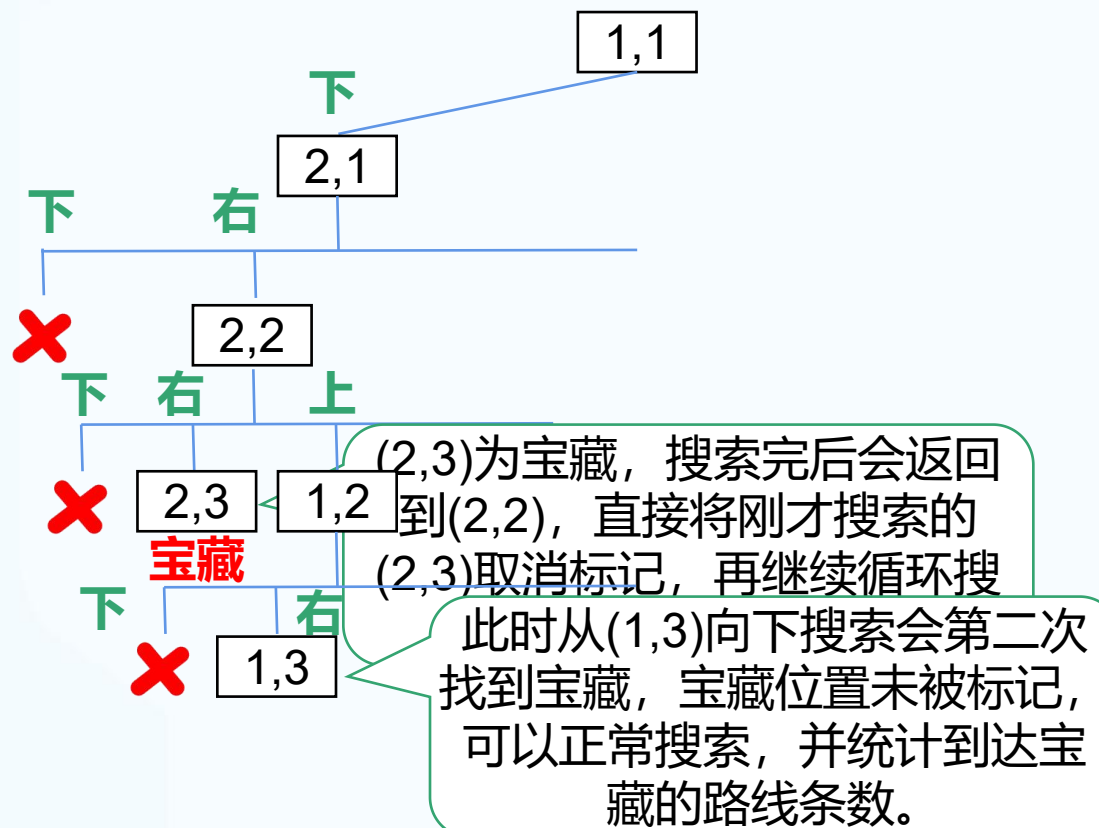
Diagram illustrating the second step of the bubble sort algorithm. The array is $[0, 0, 0]$. The first two elements are compared and found to be in order, so no swap occurs. The second and third elements are compared and found to be in order, so no swap occurs. The array remains $[0, 0, 0]$.

标记数组vis[]



问题分析

解决方法：要去搜索当前点的其他方向时，将刚才搜索过的点取消标记。过程如下：



	1	2	3
1	1	0 → 0	0
2	1 ↓ 1	↑ 1 → 0	↓ 0

标记数组vis[][]



解题步骤

1. 创建需要的数组及变量。

```
int n, g[11][11]; // 地图大小 地图
int vis[11][11]; // 标记数组
int ans; // 统计找到的路线条数
// dx 存储对应方向行下标变化
int dx[4] = {1, 0, -1, 0};
// dy 存储对应方向列下标变化
int dy[4] = {0, 1, 0, -1};
```

2. 在主函数中输入数据。

```
int main(){
    cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    return 0;
}
```



解题步骤

3. 定义并实现dfs深度优先搜索函数。

```
void dfs(int x, int y){  
    if(g[x][y]为宝藏){  
        ans++; //找到宝藏累加路线条数  
        return;  
    }  
    for(int i = 0; i < 4; i++){ //搜索四个方向  
        int tx = x + dx[i], ty = y + dy[i]; //求要去的格子的行列下标  
        if( (tx>0 && tx<=n && ty>0 && ty<=m && !vis[tx][ty] && g[tx][ty] != 1 ) ){  
            vis[tx][ty] = 1; //标记  
            dfs(tx, ty); //继续搜索  
            vis[tx][ty] = 0; //取消标记  
        }  
    }  
}
```

(tx, ty)搜索完后,
将(tx, ty)取消标记。



解题步骤

4. 主函数中调用dfs函数，从(1, 1)开始搜索，并输出路线条数。

```
int main(){  
    cin >> n;  
    for(int i = 1; i <= n; i++){  
        for(int j = 1; j <= n; j++){  
            cin >> g[i][j];  
        }  
    }  
  
    vis[1][1] = 1;  
    dfs(1, 1);  
    cout << ans;  
    return 0;  
}
```



【完整代码1】

```
#include<iostream>
using namespace std;
int n, g[11][11], ans; //地图大小 地图 统计找到的路线条数
int vis[11][11]; //标记数组
//dx存储对应方向行下标变化 dy存储对应方向列下标变化
int dx[4] = {1,0,-1,0}, dy[4] = {0,1,0,-1};
void dfs(int x, int y){
    if(g[x][y] == 2){
        ans++;
        return;
    }
    for(int i = 0; i < 4; i++){ //搜索八个方向
        int tx = x + dx[i], ty = y + dy[i]; //求要去的格子的行列下标
        if(tx>0 && tx<=n && ty>0 && ty<=n && vis[tx][ty]==0 && g[tx][ty]!=1){
            vis[tx][ty] = 1; //标记
            dfs(tx, ty); //继续搜索
            vis[tx][ty] = 0; //取消标记
        }
    }
}
```



【完整代码2】

```
int main(){
    cin >> n;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            cin >> g[i][j];
        }
    }
    vis[1][1] = 1;
    dfs(1, 1);
    cout << ans;
    return 0;
}
```