



回顾练习1

想要给动态数组v添加一个元素5，以下哪个选项可以添加？（ **A** ）

A. `v.push_back(5);`

B. `v.end()=5;`

C. `v.push(5);`

D. `v.pop_back(5);`

广度优先搜索

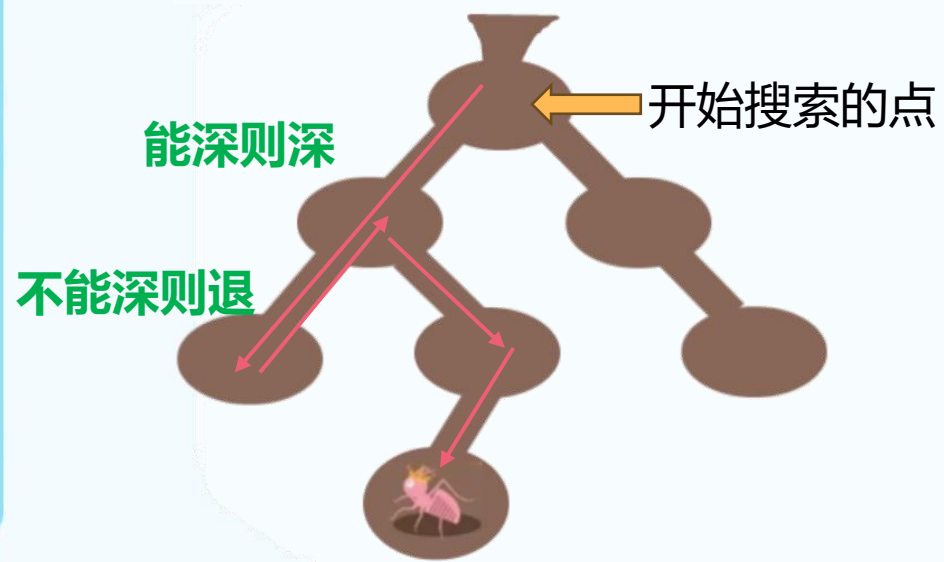


码上得知-1-广度优先搜索的概念



深度优先搜索

复习一下深度优先搜索概念



从某一个点开始，按照一定的**顺序**（先左后右）

沿着某条路往下走，**能深则深**

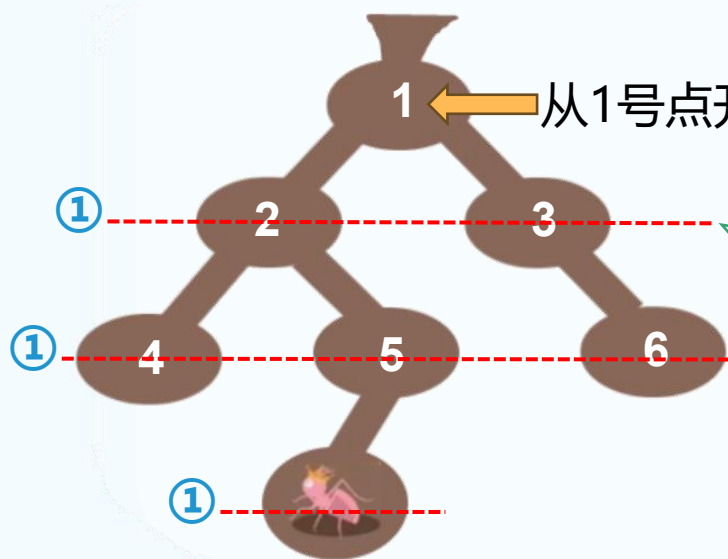
如果走完后发现不能到达目标，退回上一个点（**不能深则退**），换条路，然后继续走

如此往复，直至可能的结果都被搜索完。

这种**能深则深，不能深则退**的方法，称之为**深度优先搜索**。

广度优先搜索

还有一种非常著名的搜索方法——**广度优先搜索**



从1号点开始搜索

首先搜索**所有**与该点**相邻**的点（2号和3号点）

再进一步搜索2号和3号的**相邻点**（4，5，6号点）

2、3号点都是1号走**1步**到达的点，
称之为**第一层点**

4、5、6号点都是1号走**2步**到达的点，
称之为**第二层点**

这种**全面扩散，逐层递进**的搜索方法，称之为**广度优先搜索**简称**BFS**。

搜索迷宫

迷宫问题同样可以采用逐层搜索的方法寻找宝箱。

一个4*4的迷宫如图所示，在方格内0表示可以走，1表示障碍物不能走，2表示宝箱，现在从(1,1)位置出发，只能“上下左右”四个方向移动，使用**广度优先搜索寻找宝箱**。

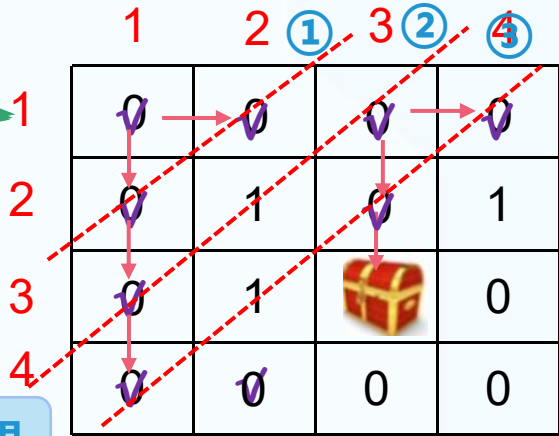


	1	2	3	4
1	0	0	0	0
2	0	1	0	1
3	0	1	2	0
4	0	0	0	0

广搜迷宫具体过程

- 1. 规定每个房间的搜索顺序，例如：下右上左
- 2. 搜索全部**相邻**房间
- 3. 搜索过的房间不再搜索

根据每个房间的搜索顺序，
搜索点周围相邻的房间，
如果没有宝箱，就标记已搜索



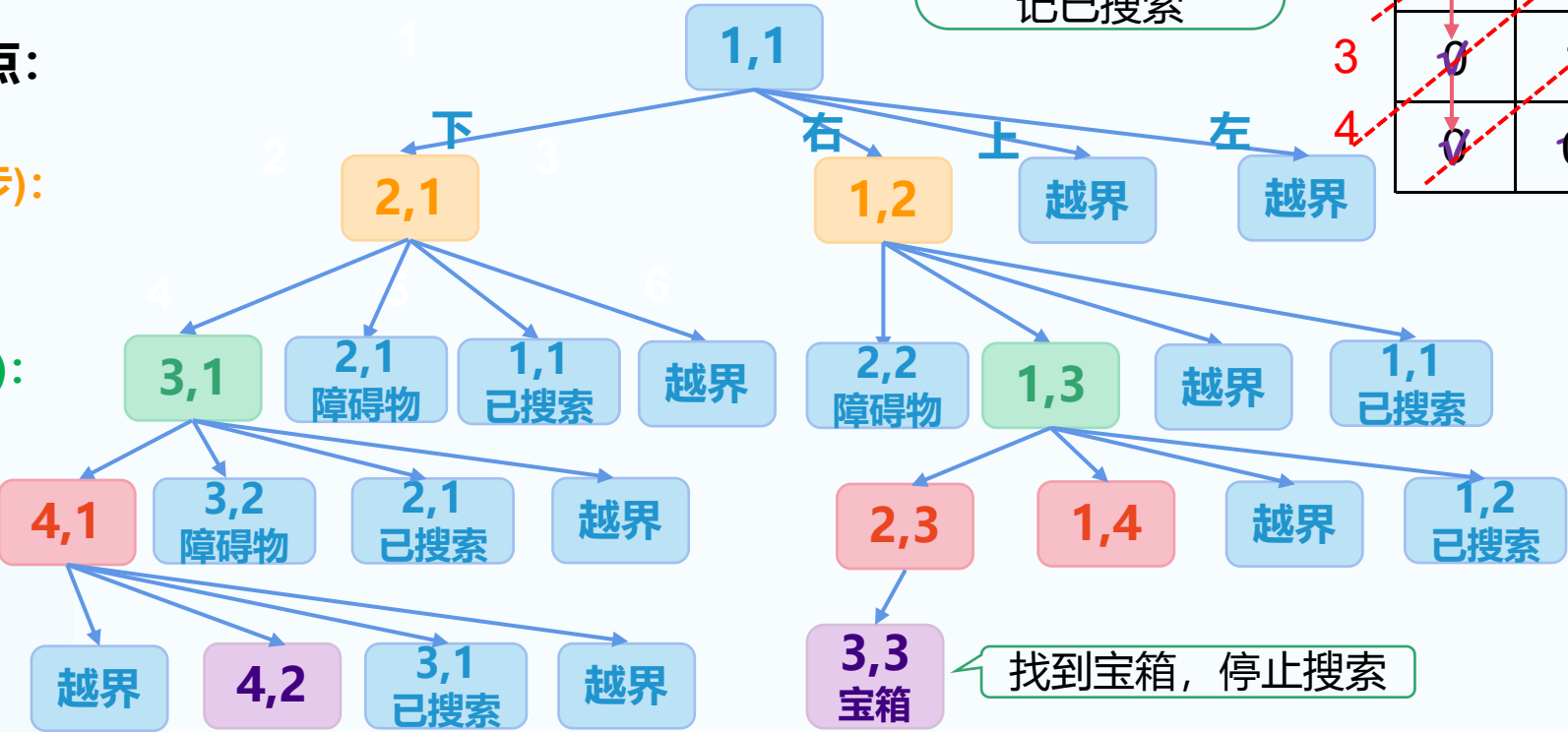
开始点:

第一层(1步):

第二层(2步):

第三层(3步):

第四层(4步):



练一练1

一个6*6的迷宫如图所示，在方格内0表示可以走，1表示障碍物不能走，2表示宝箱，现在从(1,1)位置出发，只能“上下左右”四个方向移动，第（**D**）层可以找到宝箱？（注意：出发位置不计入层数）

- A. 4
- B. 5
- C. 6
- D. 7

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	1	0	1	0	1
3	0	1	1	2	0	0
4	0	0	0	0	0	1
5	1	0	0	0	0	0
6	0	0	1	0	0	0

练一练1解析

假设每个房间搜索顺序：下右上左



	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	1	0	1	0	1
3	0	1	1	2	0	0
4	0	0	0	0	0	1
5	1	0	0	0	0	0
6	0	0	1	0	0	0



码上得知-2-广度优先搜索的实现

问题描述

【问题描述】

一个4*4的迷宫如图所示，在方格内0表示可以走，1表示障碍物不能走，2表示宝箱，现在从（1,1）位置出发，只能“上下左右”四个方向移动。用**代码实现**广搜寻找宝箱，找到输出“YES”，否则输出“NO”。

1. 规定每个房间的搜索顺序，例如：**下右上左**
2. 搜索当前全部**相邻**房间
3. 搜索过的房间不再搜索

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

代码实现广搜

开始点:

1,1

第一层(1步):

2,1

1,2

第二层(2步):

3,1

1,3

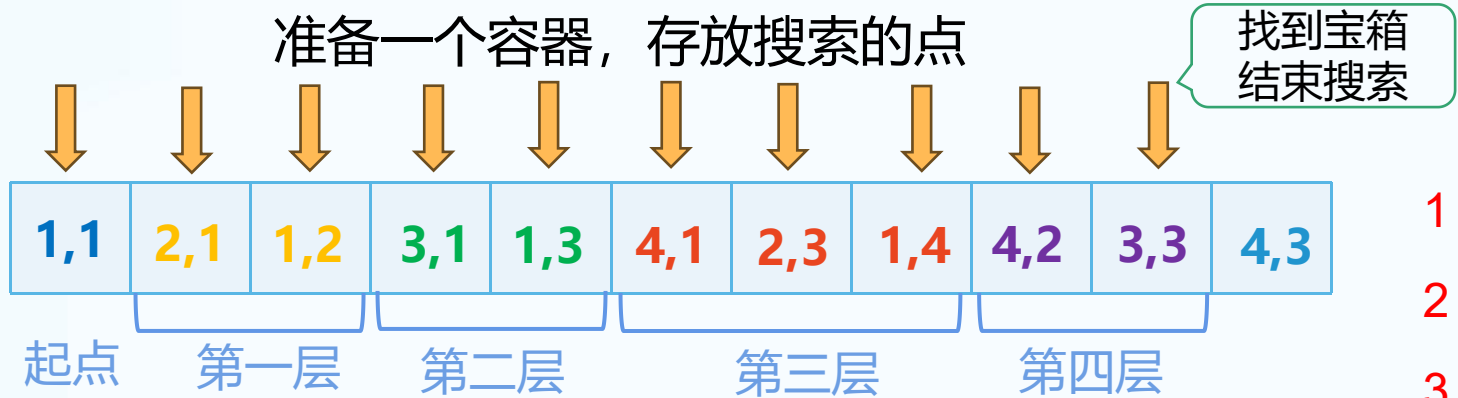
....

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

遍历相邻格子的顺序: 下右上左

广搜: 从开始点扩散到相邻点, 再由相邻点扩散到下一层相邻点, 以此类推...寻找宝箱。

代码实现广搜



先把 (1,1) 放入容器

判断元素 (1,1)，不是宝箱，相邻点放入容器

判断元素 (2,1)，不是宝箱，相邻点放入容器

判断元素 (1,2)，不是宝箱，相邻点放入容器

判断元素 (3,1)，不是宝箱，相邻点放入容器

判断元素 (1,3)，不是宝箱，相邻点放入容器

...

结论：搜索顺序是按层搜索，这种容器可以实现广度优先搜索。

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

遍历相邻格子的顺序：**下右上左**

相邻格子存入容器
下 右 上 左

代码实现广搜



搜索顺序是按层搜索

容器内的数据都具备，**先进先判断，后进去的后判断**特点。

队列

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

遍历相邻格子的顺序：**下右上左**

队列元素的存储

两个值用一个变量表示，可以用结构体

```
struct node{  
    int x; //行下标  
    int y; //列下标  
};  
node a={1,1}; //变量a表示(1,1)点
```

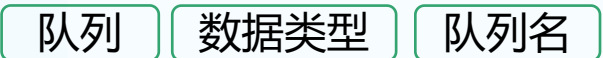
确定格子位置

行值 x
列值 y

如: (1,1)
x=1 y=1

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

创建队列



```
queue < node > q; //队列数据类型为结构体
```

node
↑
变量a类型



变量a入队

回顾队列操作函数

- q.push(a) 入队

q.pop() 出队

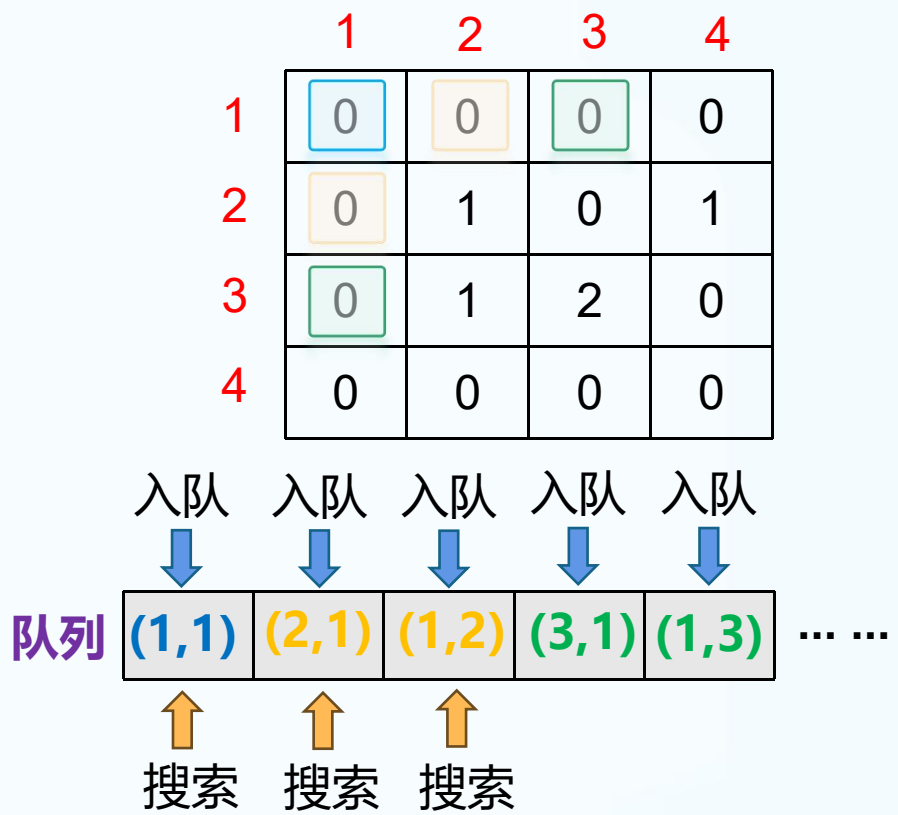
q.size() 队列元素数
- q.front() 取队首

q.back() 取队尾

q.empty() 判队空

广搜执行流程

- 1. 起点入队
node a={1,1};
q.push(a); //起点入队
- 2. 获队首元素, 判断是否为目标
(1,1) (2,1) (1,2)
新队首 新队首
- 3. 与队首相邻的格子入队
(2,1) (1,2) (3,1) (1,3)
- 4. 队首出队
(1,1) (2,1) 循环处理完所有格子





广搜核心代码

1. 起点入队

```
node a={1,1};
```

```
q.push(a); //起点入队
```

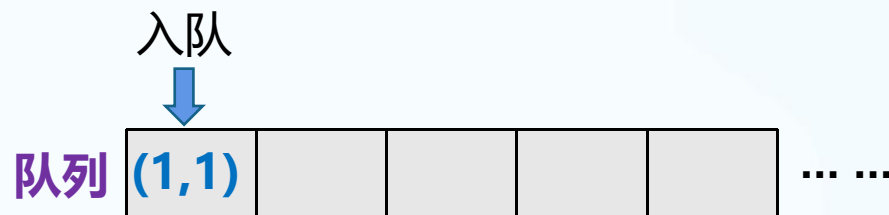
```
while( ) {
```

2. 取队首元素，判断是否为目标

3. 与队首相邻的格子入队

4. 队首出队

```
}
```



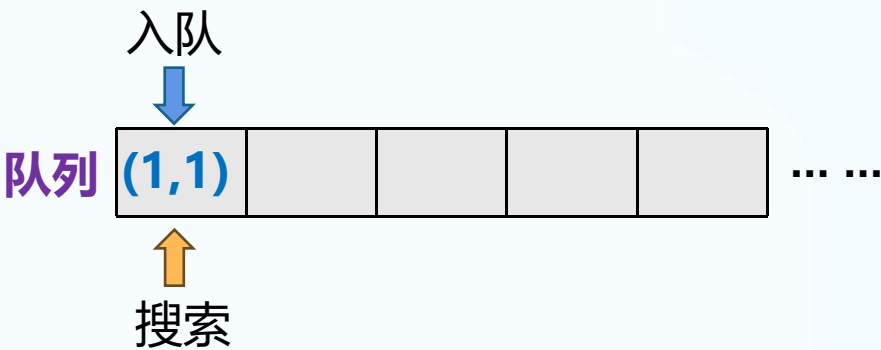
广搜核心代码

```
int mp[5][5]={}; //地图
```

```
while(           ){  
    2. 取队首元素，判断是否为目标  
    node f=q.front(); //取队首元素  
    if( mp[f.x][f.y]==2 ){ //判断是否为目标  
        cout<< "YES" ;  
        return 0;  
    }  
}
```

return作用: 结束当前函数程序, 并把0返回给外部

```
3. 与队首相邻的格子入队  
4. 队首出队  
}
```



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

广搜核心代码

```
int mp[5][5]={}; //地图
int dx[4] ={-1,0,1,0}, dy[4] ={0,1,0,-1};
while( ) {
```

2. 取队首元素, 判断是否为目标

3. 与队首相邻的格子入队

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

```
    int nx = f.x + dx[i];
```

```
    int ny = f.y + dy[i];
```

在地图范围内

不为障碍物

没有被搜索过

```
    if (nx>0&& nx<=4&& ny>0&& ny<=4&&mp[nx][ny]!=1&&vis[nx][ny]==0) {
```

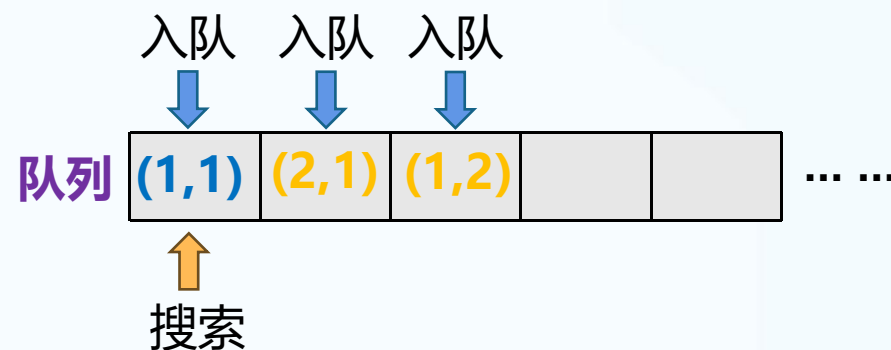
```
        vis[nx][ny]=1;    node r={nx,ny};    q.push(r);
```

标记入格子

相邻格子添加到队尾

```
    }
}
4. 队首出队
```

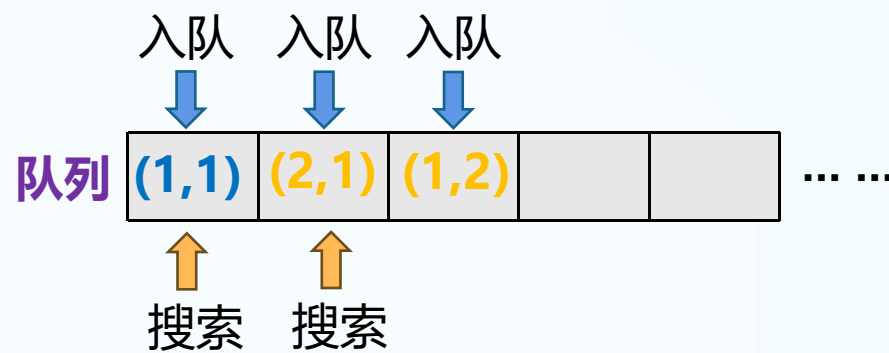
```
}
```





广搜核心代码

```
int mp[5][5]={}; //地图
int dx[4] ={-1,0,1,0}, dy[4] ={0,1,0,-1};
int vis[5][5]={}; //标记数组
while(      ){
    2. 取队首元素, 判断是否为目标
    3. 与队首相邻的格子入队
    for(int i=0; i<4; i++){
        4. int nx = f.x + dx[i];
        int ny = f.y + dy[i];
        if (nx>0&& nx<=4&& ny>0&& ny<=4&&mp[nx][ny]!=1&&vis[nx][ny]==0) {
            vis[nx][ny]=1; node r={nx,ny}; q.push(r);
        }
    }
    4. 队首出队
    q.pop( );
}
```



广搜核心代码

```
int mp[5][5]={}; //地图
int dx[4] ={-1,0,1,0}, dy[4] ={0,1,0,-1};
int vis[5][5]={}; //标记数组
while(q.empty()!=1){ //队列不为空
    2. 取队首元素, 判断是否为目标
    3. 与队首相邻的格子入队
    for(int i=0; i<4; i++){
        int nx = f.x + dx[i];
        int ny = f.y + dy[i];
        if (nx>0&& nx<=4&& ny>0&& ny<=4&&mp[nx][ny]!=1&&vis[nx][ny]==0){
            vis[nx][ny]=1; node r={nx,ny}; q.push(r);
        }
    }
    4. 队首出队
    q.pop( );
}
```

重复执行步骤**234**, 直到队列为空, 为空时结束广搜

队列



假设没有格子可以继续添加
第3步的if条件不再满足



广搜核心代码

```
int mp[5][5]={}; //地图
int dx[4] ={-1,0,1,0}, dy[4] ={0,1,0,-1};
int vis[5][5]={}; //标记数组
while( q.empty()!=1 ){ //队列不为空
    2. 取队首元素，判断是否为目标
    node f=q.front(); //取队首元素
    if(mp[f.x][f.y]==2){ //判断是否为目标
        cout<< "YES" ;
        return 0;
    }
    3. 与队首相邻的格子入队
    4. 队首出队
    q.pop( );
}
```

广搜结束方式

队列

--	--	--	--	--

① 队列为空

队列

...	...	...	(3,3)	...
-----	-----	-----	-------	-----

找到宝箱

② 找到目标直接退出

广搜核心代码

```
int mp[5][5]={}; //地图
int dx[4] ={-1,0,1,0}, dy[4] ={0,1,0,-1};
int vis[5][5]={}; //标记数组
while( q.empty()!=1 ){ //队列不为空
    2. 取队首元素, 判断是否为目标
    node f=q.front(); //取队首元素
    if(mp[f.x][f.y]==2){ //判断是否为目标
        4. cout<< "YES" ; ← 找到宝箱, 输出 "YES", 程序直接结束
        return 0;
    }
    3. 与队首相邻的格子入队
    4. 队首出队
    q.pop(); 找不到宝箱, 循环结束, 程序顺序向下执行
}
cout<< "NO" ;
```

题目要求找不到宝箱输出 "NO"

写一写

```
#include<bits/stdc++.h>
using namespace std;
struct node {
    int x; //行下标
    int y; //列下标
};
queue<node> q; //队列
int mp[5][5] = {}; //地图
int vis[5][5] = {}; //标记数组
int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};

int main() {
    //输入地图信息
    for (int i = 1; i <= 4; i++) {
        for (int j = 1; j <= 4; j++) {
            cin >> mp[i][j];
        }
    }
}
```

```
//1. 起点入队
node a = {1, 1};
q.push(a);
vis[1][1] = 1;
while (q.empty() != 1) {
    //2. 取队首元素, 判断是否为目标
    node f = q.front();
    if (mp[f.x][f.y] == 2) {
        cout << "YES";
        return 0;
    }
}
```

写一写

```
//3. 与队首相邻的格子入队
for (int i = 0; i < 4; i++) {
    int nx = f.x + dx[i];
    int ny = f.y + dy[i];
    if (nx >= 1 && nx <= 4 && ny >= 1 && ny <= 4 && mp[nx][ny] != 1 && vis[nx][ny] == 0) {
        vis[nx][ny] = 1;
        node r = {nx, ny};
        q.push(r);
    }
}
//4. 队首出队
q.pop();
}
cout << "NO";
return 0;
}
```



广度优先搜索的概念

从某一个点开始，首先搜索所有与该点相邻的点，然后再进一步搜索这些点的相邻点，以此类推找到目标。这种**全面扩散，逐层递进**的搜索方法，称之为**广度优先搜索**简称**BFS**。

广度优先搜索的模板

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int mp[N][M]={};
int vis[N][M]={};
int dx[4] ={-1,0,1,0};
int dy[4] ={0,1,0,-1};
```

核心搜索代码，
封装成函数。

```
void bfs(){
    ① 起点入队 → node a={1,1}; vis[1][1]=1; q.push(a);
    while( q.empty()!=1 ){
        ② 取队首元素，判断是否为目标 → node f=q.front();
        if(mp[f.x][f.y]==2){
            cout<< "YES" ; return;
        }
        ③ 当前队首相邻的格子入队 → for(int i=0; i<4; i++){
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx>0&& nx<=n&& ny>0&&ny<=m
                &&mp[nx][ny]!=1&&vis[nx][ny]==0) {
                vis[nx][ny]=1;
                node r={nx,ny};
                q.push(r);
            }
        }
        ④ 队首出队 → q.pop( );
    }
    cout<< "NO" ;}
```

广搜函数

广度优先搜索应用



棋盘寻宝

【问题描述】

有一个 $n*m$ 的棋盘 ($1 \leq n, m \leq 100$)，棋盘上有侍卫和宝藏，在棋盘的左上角 $(1,1)$ 开始寻找宝藏，如果能避开侍卫找到宝藏输出YES，否则输出NO。注意：左上角可能有侍卫，此题广度优先搜索解决。

【输入格式】

两个非零整数 n 和 m ， n 表示迷阵行数， m 表示迷阵列数。接下来有 n 行，每行包含 m 个符号，不同字符分别代表不同含义。

“.”：可以安全通行的方格，“#”：有守卫的方格，“*”：宝藏所在位置。

【输出格式】

找到宝藏输出YES，否则输出NO。

【输入样例1】

```
4 4
#...
....
...*
....
```

【输出样例1】

NO

【输入样例2】

```
4 4
....
.#.#
*...
....
```

【输出样例2】

YES

问题分析

【输入样例1】

【输出样例1】

【输入样例2】

【输出样例2】

4 4

NO

4 4

YES

不能通行

. . .
. . . .
. . . *
. . . .

. . . .
. # . #
. * . .
. . . .

	1	2	3	4
1	#	.	.	.
2	.	.	.	.
3	.	.	.	* 
4	.	.	.	.

	1	2	3	4
1	✓ →	✓ →	✓ →	✓
2	↓ ✓	#	↓ ✓	#
3	↓ ✓ →	* 	.	.
4	↓ ✓	.	.	.

解题步骤

1. 创建相关局部变量。

```
char mp[101][101]; //地图
int n, m; //n*m的地图
```

字符类型

起点搜索前先
判断起点

	1	2	3	4
1	.	.	.	.
2	.	#	.	#
3	.	*	.	.
4	.	.	.	.

2. 输入数据。

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

3. 判断起点。

```
if(mp[1][1]== '#' ){
    cout<< "NO" ;
    return 0;
}
```

开始广搜

```
return 0;
```

```
}
```

解题步骤

4. 修改广度优先搜索模板。

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int vis[101][101];
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
```

① 起点入队

② 取队首元素,
判断是否为目标

③ 当前队首相
邻的格子入队

④ 队首出队

```
void bfs(){
    node a={1,1}; vis[1][1]=1; q.push(a);
    while( q.empty()!=1 ){
        node f=q.front();
        if(mp[f.x][f.y]==2){
            cout<< "YES" ; return;
        }
        for(int i=0; i<4; i++){
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx>0&& nx<=n&& ny>0&&ny<=m
                &&mp[nx][ny]!=1&&vis[nx][ny]==0) {
                vis[nx][ny]=1;
                node r={nx,ny};
                q.push(r);
            }
        }
        q.pop( );
    }
    cout<< "NO" ;}
```

广搜函数

解题步骤

4. 修改广度优先搜索模板。

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int vis[101][101];
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
```

① 起点入队

② 取队首元素，
判断是否为目标

③ 当前队首相
邻的格子入队

④ 队首出队

```
void bfs(){
    node a={1,1}; vis[1][1]=1; q.push(a);
    while( q.empty()!=1 ){
        node f=q.front();
        if(mp[f.x][f.y]=='*'){
            cout<< "YES" ; return;
        }
        for(int i=0; i<4; i++){
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx>0&& nx<=n&& ny>0&&ny<=m
                &&mp[nx][ny]!='#'&&vis[nx][ny]==0) {
                vis[nx][ny]=1;
                node r={nx,ny};
                q.push(r);
            }
        }
    }
    q.pop( );
    cout<< "NO" ;}
```

修改为 '*'

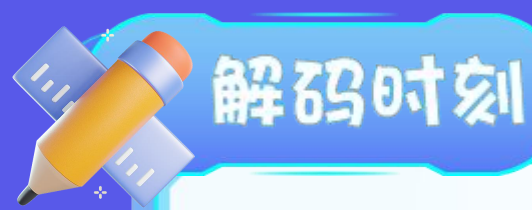
修改为 '#'，表
示不可以通行

广搜函数

完整代码-主函数

```
#include <bits/stdc++.h>
using namespace std;
struct node {
    int x, y;
};
queue <node> q; // 队列
char mp[101][101]; // 地图
int vis[101][101]; // 标记数组
int dx[4] = {1, 0, -1, 0}, // 下右上左
int dy[4] = {0, 1, 0, -1};
int n, m; // n*m的地图
void bfs() { ... }
```

```
int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> mp[i][j];
        }
    }
    if(mp[1][1] == '#'){
        cout << "NO";
        return 0;
    }
    bfs();
    return 0;
}
```



完整代码-BFS函数

```
void bfs() {  
    node a = {1, 1}; vis[1][1] = 1;  
    q.push(a);  
    while (q.empty() != 1) { // 队列不为空  
        node f = q.front();  
        if (mp[f.x][f.y] == '*') {  
            cout << "YES"; return;  
        }  
        for (int i = 0; i < 4; i++) {  
            int nx = f.x + dx[i], ny = f.y + dy[i];  
            if (nx > 0 && nx <= n && ny > 0 && ny <= m && mp[nx][ny] != '#' && vis[nx][ny] == 0) {  
                vis[nx][ny] = 1;  
                node r = {nx, ny};  
                q.push(r);  
            }  
        }  
        q.pop(); // 队首出队  
    }  
    cout << "NO";  
}
```

岛屿面积

【问题描述】

小童是个游戏迷，这天他和小伙伴相约刺激战场。游戏地图是 $n*m$ 的大小，游戏开局小童降落到场地的 sx,sy 点。根据地图提示发现，这片区域的陆地和海水是交错排布的，其中数字**1表示陆地**，**0表示海水**。被水围起来的区域就是“岛屿”。为了选择合适的逃生装备，小童需要计算一下所在岛屿的面积，你能帮一帮他吗？（注意：**数据保证降落点为1**）。

【输入格式】

四个整数 n,m,sx,sy ， n 和 m 表示图的行和列(**$1 \leq n,m \leq 50$**)， sx 和 sy 表示小童所处的横纵坐标。（一个1：表示陆地的面积为1，并且每个格子之间只能“上下左右”移动。）

【输出格式】

你所在岛屿的面积。

【输入样例】

```
5 6 3 4
0 1 0 1 1 0
0 0 0 1 0 1
1 1 0 1 0 1
1 0 0 1 0 0
1 1 0 1 0 0
```

【输出样例】

```
6
```

问题分析

游戏开局小童降落到地图的 (3,4) 点，每个格子之间只能 “上下左右” 移动，一共可以访问6个格子。

格子元素相同，任意2个格子都能直接或间接相互到达，格子数量尽可能的多，这种区域就是**连通块**。

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	0	0	0	1	0	1
3	1	1	0		0	1
4	1	0	0	1	0	0
5	1	1	0	1	0	0

从(3,4)点开始往周围四个方向进行搜索，尝试把所有能访问的格子全部访问一遍，这种解题思路符合广度优先搜索。

问题分析

1. 规定每个格子的访问顺序，例如：**下右上左**
2. 访问全部**相邻**格子
3. 访问过的格子不再访问

开始点： 降落(3,4)点，标记已访问

搜索**第一层**点的**相邻**格子

第一层(1步)：

搜索**第二层**点的**相邻**格子

第二层(2步)：

第三层(3步)：

3,4

下

右

上

左

4,4

1,2
海水

2,4

3,3
海水

5,4

4,5
海水

3,4
已访问

4,3
海水

3,4
已访问

3,5
海水

1,4

2,3
海水

越界

5,5
海水

4,4
已访问

5,3
海水

2,4
已访问

1,5

越界

2,3
海水

2,5
海水

1,6
海水

越界

2,4
已访问

无法继续搜索时，标记过的格子就是答案

	1	2	3	4	5	6
1	0	1	0	✓	✓	0
2	0	0	0	✓	0	1
3	1	1	0	✓	0	1
4	1	0	0	✓	0	0
5	1	1	0	✓	0	0

解题步骤

1. 创建相关局部变量。

```
int mp[51][51]; //地图
int n, m; //n行m列
int sx, sy; //(sx,sy)降落点
```

2. 主函数内读入数据。

```
int main(){
    cin>>n>>m>>sx>>sy;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>mp[i][j];
        }
        开始广搜
    }
    return 0;
}
```

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	0	0	0	1	0	1
3	1	1	0		0	1
4	1	0	0	1	0	0
5	1	1	0	1	0	0

解题步骤

3. 修改广度优先搜索模板。

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int vis[101][101];
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
```

① 起点入队

② 取队首元素,
判断是否为目标

③ 当前队首相
邻的格子入队

④ 队首出队

```
void bfs(){
    node a={1,1}; vis[1][1]=1; q.push(a);
    while( q.empty()!=1 ){
        node f=q.front();
        if(mp[f.x][f.y]==2){
            cout<< "YES" ; return;
        }
        for(int i=0; i<4; i++){
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx>0&& nx<=n&& ny>0&&ny<=m
                &&mp[nx][ny]!=1&&vis[nx][ny]==0) {
                vis[nx][ny]=1;
                node r={nx,ny};
                q.push(r);
            }
        }
        q.pop( );
    }
    cout<< "NO" ;}
```

广搜函数

解题步骤

统计标记格子的数量

3. 修改广度优先搜索模板。

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int vis[51][51];
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
int ans;
```

小童所在连通块的数量

① 起点入队

```
void bfs(){
    node a={sx,sy}; vis[sx][sy]=1; q.push(a); ans++;
```

起始点为(sx,sy)

② 取队首元素, 判断是否为目标

```
while( q.empty()!=1 ){
    node f=q.front();
```

本题统计已搜索格子的数量, 不需要判断队首元素

③ 当前队首相邻的格子入队

```
for(int i=0; i<4; i++){
    int nx = f.x + dx[i], ny = f.y + dy[i];
    if (nx>0&& nx<=n&& ny>0&&ny<=m
        &&mp[nx][ny]!=1&&vis[nx][ny]==0) {
        vis[nx][ny]=1;
        node r={nx,ny};
        q.push(r); ans++;
    }
}
```

可以通行

统计标记格子的数量

④ 队首出队

```
q.pop( );
```

```
cout<< "NO" ; }
```

不需要输出NO

广搜函数

解题步骤

2. 主函数调用bfs()函数。

```
int main(){
    cin>>n>>m>>sx>>sy;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>mp[i][j];
        }
        bfs();
        cout<< ans;
        return 0;
    }
}
```

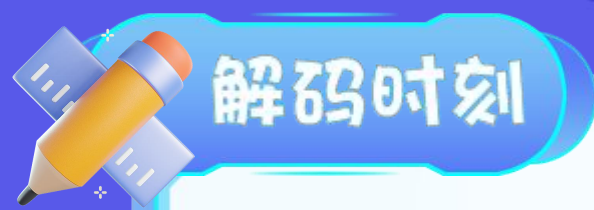
	1	2	3	4	5	6
1	0	1	0	1	1	0
2	0	0	0	1	0	1
3	1	1	0	1	0	1
4	1	0	0	1	0	0
5	1	1	0	1	0	0

完整代码-主函数

```
#include <bits/stdc++.h>
using namespace std;
struct node {
    int x, y;
};
queue <node> q; // 队列
int mp[51][51]; // 地图
int vis[51][51]; // 标记数组
int dx[4] = {1, 0, -1, 0}; // 下右上左
int dy[4] = {0, 1, 0, -1};
int n, m, sx, sy; // n行m列的迷宫(sx,sy)降落点
int ans; // 1表示的连通块数量
void bfs() { ... }
```

```
int main() {
    cin >> n >> m >> sx >> sy;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> mp[i][j];
        }
    }

    bfs();
    cout << ans;
    return 0;
}
```



完整代码-BFS函数

```
void bfs() {  
    node a = {sx, sy};  
    vis[sx][sy] = 1;  
    q.push(a);  
    ans = 1;  
    while (q.empty() != 1) {    //队列不为空  
        node f = q.front();  
        for (int i = 0; i < 4; i++) {  
            int nx = f.x + dx[i], ny = f.y + dy[i];  
            if (nx > 0 && nx <= n && ny > 0 && ny <= m && mp[nx][ny] == 1 && vis[nx][ny] == 0) {  
                vis[nx][ny] = 1;  
                node r = {nx, ny};  
                q.push(r);  
                ans++;  
            }  
        }  
        q.pop();    //队首出队  
    }  
}
```

能养几只公羊

【问题描述】

有一个农夫有个 $n*m$ 大小的农场，农场的四周有墙把农场跟外面隔离开，同时农场里也有一些隔离墙，其余的是空地。

农夫想养一些公羊，但是公羊好斗，而且公羊在农场里会“上下左右”四处走到（但不能穿过墙）两只公羊不能见面（见面就会顶角受伤），为了不让公羊受伤，农夫想知道他的农场最多能养几只公羊？

为了方便起见，农场的描述用**0表示空地**，**1表示墙**。

【输入格式】

第一行输入 n 和 m ，表示农场大小是 $n*m$ 。（ $1 \leq n, m \leq 1000$ ），

接下来是一个 $n*m$ 的矩阵，矩阵里每个值是0或者1。

【输出格式】

一行，一个整数，表示农夫能养的公羊数量。

【输入样例】

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

【输出样例】

```
3
```

问题分析

0表示空地，1表示墙，“上下左右”相连的空地只能存在一只公羊。

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

只能养三只公羊

此题求0表示连通块的数量。

问题分析

1. (1,1)位置开始遍历矩阵（从左到右，从上到下）寻找数字0。
2. 搜索当前数字0所在的连通块，标记这个连通块所有格子为已搜索。
3. 标记为已搜索的格子不再搜索。

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

连通块数量: 3

继续寻找未标记的数字0，
并标记这个连通块

重复操作，直
到遍历完矩阵。

解题步骤

1. 创建相关局部变量。

```
int mp[1001][1001]; //地图
int vis[1001][1001]; //标记数组
int n, m; //n行m列的迷宫
```

2. 读入数据。

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

        return 0;
    }
```

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

解题步骤

3. 遍历矩阵，统计0表示连通块的数量。

`int sx, sy;` 全局变量

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

```
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
```

`if (mp[i][j]==0 && vis[i][j]==0) {` 数字为0且未标记

`sx=i, sy=j;` //从当前点(sx, sy)开始搜索

开始广搜

```
    }
}
return 0;
}
```

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

解题步骤

4. 修改广度优先搜索模板。

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int vis[101][101];
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
```

① 起点入队

② 取队首元素,
判断是否为目标

③ 当前队首相
邻的格子入队

④ 队首出队

```
void bfs(){
    node a={1,1}; vis[1][1]=1; q.push(a);
    while( q.empty()!=1 ){
        node f=q.front();
        if(mp[f.x][f.y]==2){
            cout<< "YES" ; return;
        }
        for(int i=0; i<4; i++){
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx>0&& nx<=n&& ny>0&&ny<=m
                &&mp[nx][ny]!=1&&vis[nx][ny]==0) {
                vis[nx][ny]=1;
                node r={nx,ny};
                q.push(r);
            }
        }
        q.pop( );
    }
    cout<< "NO" ;}
```

广搜函数

解题步骤

起始点(sx,sy)

4. 修改广度优先搜索模板。

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
```

① 起点入队

② 取队首元素, 判断是否为目标

③ 当前队首相邻的格子入队

④ 队首出队

```
void bfs(){
    node a={sx,sy}; vis[sx][sy]=1; q.push(a);
    while( q.empty()!=1 ){
        node f=q.front();
        for(int i=0; i<4; i++){
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx>0&& nx<=4&& ny>0&&ny<=4
                &&mp[nx][ny]!=1&&vis[nx][ny]==0) {
                vis[nx][ny]=1;
                node r={nx,ny};
                q.push(r);
            }
        }
        q.pop( );
    }
    cout<< "NO" ; }
```

本题统计已搜索格子的数量, 不需要判断队首元素

0表示空地

不需要输出NO

广搜函数

解题步骤

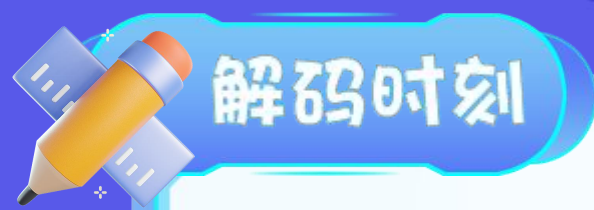
```
int main(){
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            cin>>mp[i][j];
        }
        int ans=0; //ans:连通块数量
        for(int i=1; i<=n; i++){
            for(int j=1; j<=m; j++){
                if (mp[i][j]==0&&vis[i][j]==0) {
                    sx=i, sy=j;
                    bfs();
                    ans++;
                }
            }
        }
        cout<<ans;
        return 0;
    }
}
```

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

完整代码-主函数

```
#include <bits/stdc++.h>
using namespace std;
struct node {
    int x, y;
};
queue <node> q; //队列
int mp[1001][1001]; //地图
int vis[1001][1001]; //标记数组
int dx[4] = {1, 0, -1, 0}; //下右上左
int dy[4] = {0, 1, 0, -1};
int n, m; //n行m列的迷宫
int sx, sy; //(sx,sy)开始搜索的点
void bfs() { ... }
```

```
int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> mp[i][j];
        }
    }
    int ans = 0; //连通块数量
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (mp[i][j] == 0 && vis[i][j] == 0) {
                sx = i; sy = j;
                bfs();
                ans++;
            }
        }
    }
    cout << ans;
    return 0;
}
```



完整代码-BFS函数

```
void bfs() {  
    node a = {sx, sy};  
    vis[sx][sy] = 1;  
    q.push(a);  
    while (q.empty() != 1) {    //队列不为空  
        node f = q.front();  
        for (int i = 0; i < 4; i++) {  
            int nx = f.x + dx[i], ny = f.y + dy[i];  
            if (nx > 0 && nx <= n && ny > 0 && ny <= m && mp[nx][ny] == 0 && vis[nx][ny] == 0) {  
                vis[nx][ny] = 1;  
                node r = {nx, ny};  
                q.push(r);  
            }  
        }  
        q.pop();    //队首出队  
    }  
}
```



森林探险

【问题描述】

小童在森林里探险的时候不小心走入了一个迷宫，迷宫可以看成是由 $n*n$ 的格点组成，每个格点只有2种状态.和#，.表示可以通行，#表示不能通行。同时当小童处在某个格点时，他只能移动到上下左右四个相邻格点上，小童想要从点A走到点B，问在不走出迷宫的情况下能不能办到。注意：起点不为#号，此题广度优先搜索解决。

【输入格式】

第1行表示有t组测试数据，后面跟着t组输入。每组测试数据的第1行是一个正整数n，表示迷宫的规模是 $n*n$ 的。接下来是一个 $n*n$ 的矩阵，矩阵中的元素为.或者#。再接下来一行是4个整数sx,sy,ex,ey，A点(sx,sy)，B点(ex,ey)。(1 ≤ t,n ≤ 50) 注意：地图左上角为 (1,1)

【输出格式】

t行，每行输出对应一组输入。能办到则输出“YES”，否则输出“NO”。

【输入样例】

```
2
3
.##
..#
#..
1 1 3 3
5
.....
###.#
..#..
###..
...#.
1 1 5 1
```

【输出样例】

```
YES
NO
```

问题分析

【输入样例】

2

3

..##

..#

#..

1 1 3 3

5

....

####.#

..#..

####..

...#.

1 1 5 1

表示有两组数据

3行3列, "." 通行, "#" 不通行

起点(1,1), 终点(3,3)

5行5列, "." 通行, "#" 不通行

起点(1,1), 终点(5,1)

【输出样例】

YES

NO

	1	2	3
1	.	#	#
2	.	.	#
3	#	.	.

可以到达

此题求多组迷宫是否能搜到宝箱。

for(循环t次){ t组测试数据

搜索一个迷宫问题

我们已经解决

	1	2	3	4	5
1	.	.	.	.	.
2	#	#	#	.	#
3	.	.	#	.	.
4	#	#	#	.	.
5	.	.	.	#	.

不可以到达

解题步骤

1. 创建相关局部变量。

```
char mp[51][51]; //地图
int t,n; //t组数据 n*n的地图
int sx,sy,ex,ey; //A点(sx,sy) B点(ex,ey)
```

字符类型

2. 输入数据。

```
int main(){
    cin>>t;
    for(int k=1; k<=t; k++){ //循环t次
        cin>>n;
        for(int i=1; i<=n; i++){
            for(int j=1; j<=n; j++){
                cin>>mp[i][j];
            }
        }
        cin>>sx>>sy>>ex>>ey;
        开始广搜
    }
    return 0;
}
```

解题步骤

3. 修改广度优先搜索模板。

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int vis[101][101];
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
```

① 起点入队

② 取队首元素,
判断是否为目标

③ 当前队首相
邻的格子入队

④ 队首出队

```
void bfs(){
    node a={1,1}; vis[1][1]=1; q.push(a);
    while( q.empty()!=1 ){
        node f=q.front();
        if(mp[f.x][f.y]==2){
            cout<< "YES" ; return;
        }
        for(int i=0; i<4; i++){
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx>0&& nx<=n&& ny>0&&ny<=m
                &&mp[nx][ny]!=1&&vis[nx][ny]==0) {
                vis[nx][ny]=1;
                node r={nx,ny};
                q.push(r);
            }
        }
        q.pop( );
    }
    cout<< "NO" ;}
```

广搜函数



解题步骤

3. 修改广度优先搜索模板。

全局变量

```
struct node{
    int x, y;
};
queue<node> q;
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
```

① 起点入队

② 取队首元素,
判断是否为目标

③ 当前队首相
邻的格子入队

④ 队首出队

```
void bfs(){
    node a={sx,sy}; vis[sx][sy]=1; q.push(a);
    while( q.empty()!=1 ){
        node f=q.front();
        if( f.x==ex&&f.y==ey){
            cout<<"YES"<<endl; return;
        }
        for(int i=0; i<4; i++){
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx>0&& nx<=n&& ny>0&&ny<=m
                &&mp[nx][ny]!='1' &&vis[nx][ny]==0) {
                vis[nx][ny]=1;
                node r={nx,ny};
                q.push(r);
            }
        }
        q.pop( );
    }
    cout<<"NO"<<endl; }
```

A点入队

判断是否到达B 点

n*n地图

.可通行

广搜函数

注意: 每次输出要有换行符。

解题步骤

4. 初始化vis数组和队列。

```
int main(){
    cin>>t;
    for(int k=1; k<=t; k++){ //循环t次
        cin>>n;
        for(int i=1; i<=n; i++){
            for(int j=1; j<=n; j++){
                cin>>mp[i][j];
            }
        }
        cin>>sx>>sy>>ex>>ey;

        bfs();

        return 0;
    }
```

搜索之前，确保vis数组为0同时队列为空

	1	2	3
1	.	#	#
2	.	.	#
3	#	.	.

mp

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

vis

每次搜索都会改变vis数组的值，

队列

		目标点	其它点	
--	--	-----	-----	--

队列同理，其它点没有清除掉

解题步骤

4. 初始化vis数组和队列。

```
int main(){
    cin>>t;
    for(int k=1; k<=t; k++){ //循环t次
        cin>>n;
        for(int i=1; i<=n; i++){
            for(int j=1; j<=n; j++){
                cin>>mp[i][j];
            }
        }
        cin>>sx>>sy>>ex>>ey;

        bfs();

        return 0;
    }
```

```
for(int i=1; i<=n; i++){
    for(int j=1; j<=n; j++)
        vis[i][j]=0;
}
while( q.empty()!=1 ){
    q.pop( );
}
```

初始化vis为0

清空队列



完整代码-主函数

```
#include <bits/stdc++.h>
using namespace std;
struct node {
    int x, y;
};
queue <node> q; // 队列
char mp[51][51]; // 地图
int vis[51][51]; // 标记数组
int dx[4] = {1, 0, -1, 0}; // 下右上左
int dy[4] = {0, 1, 0, -1};
int t, n; // t组数据 n*n的地图
int sx, sy, ex, ey; // A点(sx,sy) B点(ex,ey)
void bfs() { ... }
```

```
int main() {
    cin >> t;
    for (int k = 1; k <= t; k++) { // 循环t次
        while(q.empty() != 1) { // 清空队列
            q.pop();
        }
        for (int i = 1; i <= n; i++) { // 还原标记
            for (int j = 1; j <= n; j++)
                vis[i][j] = 0;
        }
        cin >> n;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++)
                cin >> mp[i][j];
        }
        cin >> sx >> sy >> ex >> ey;
        bfs();
    }
    return 0;
}
```



完整代码-BFS函数

```
void bfs() {
    node a = {sx, sy};
    q.push(a); vis[sx][sy] = 1;
    while (q.empty() == 0) {    // 队列不为空
        node f = q.front();
        if(f.x == ex && f.y == ey){
            cout << "YES" << endl;
            return;
        }
        for (int i = 0; i < 4; i++) {
            int nx = f.x + dx[i], ny = f.y + dy[i];
            if (nx > 0 && nx <= n && ny > 0 && ny <= n && mp[nx][ny] == '.' && vis[nx][ny] == 0) {
                vis[nx][ny] = 1; node r = {nx, ny}; q.push(r);
            }
        }
        q.pop();
    }
    cout << "NO" << endl;
}
```