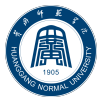
广度优先搜索算法

一层一层往外走，先到的一定最近





HGNU-ACM 实验室算法组

基础篇：队列、逐层扩展、判重、网格模板

截图配套：2 道例题，10 道练习题

关键词：队列、先进先出、最短步数、多源 BFS

本章目录 | 知识点

从迷宫到广度优先搜索

队列：BFS 的发动机

逐层扩展与最短步数

网格上的 BFS 模板

判重与状态设计

本章目录 | 例题与练习

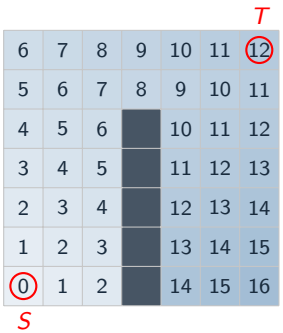
例题精讲

总结与易错点

本章全部练习题索引

先看一个迷宫：为什么需要 BFS

- 迷宫只能上下左右走，求 S 到 T 的最少步数。
- 广度优先搜索（BFS）按距离一层层扩展：先走 1 步能到的格子，再走 2 步能到的格子，依次向外。
- 右图数字是从 S 出发的最少步数；颜色越深，离 S 越远。
- 灰色格子是墙，只能绕行，不能穿过。



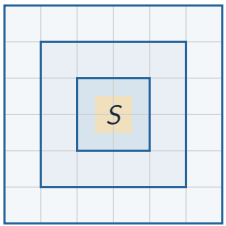
数字 = 最少步数；深色 = 距离大

本页要点

BFS 不急着走远，而是把“走 1 步能到哪、走 2 步能到哪”依次算清楚。

什么是广度优先搜索

- 广度优先搜索（Breadth-First Search，BFS）：从起点出发，先访问所有距离为 1 的状态，再访问距离为 2 的状态，像水波一样一圈一圈向外扩散。
- 核心问题：从起点到每个状态最少要走几步。
- 三个关键词：逐层、队列、判重。
- 只有每条边代价相同时，BFS 才能求最短路；边权不同要用 Dijkstra。



由内向外：第 $0, 1, 2, 3$ 层

每一圈就是“多走一步”能到的范围

本页要点

BFS 把“最短步数”变成“数圈数”：第 k 圈就是距离 k 的全部状态。

BFS 的适用条件：三条缺一不可

能用 BFS 的三个条件

1. 状态有限且可枚举：能用一个或几个整数表示，能开数组判重。
2. 每次转移代价相同：走一步都算 1，或都算同样的代价。
3. 只问最少步数或可达性：不要求按某种顺序枚举全部方案。

- 条件 2 不满足时，BFS 的“第一次到达”不再是最短，应改用 Dijkstra。
- 条件 1 不满足时，状态无限多，必须先用题目范围把状态限制住。
- 条件 3 不满足时（要枚举所有路径），应使用 DFS 与回溯。
- 三条都满足时，BFS 的时间复杂度只与状态数和转移数有关。

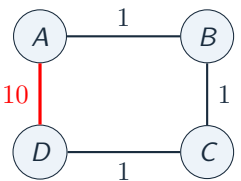
本页要点

先判断“能不能用 BFS”，再动手写；三条适用条件比代码本身更重要。

反例：边权不同，BFS 会选错

- 图中每条边上的数字是边权，不是步数。
- BFS 只看“走了几条边”，会选 $A \to D$：1 条边，总代价 10。
- 真正的最短路是 $A \to B \to C \to D$：3 条边，总代价 3。
- 边权不同时，必须用 Dijkstra、Bellman-Ford 等最短路算法。

真正最短路 $A \to B \to C \to D$：3 条边，代价 3



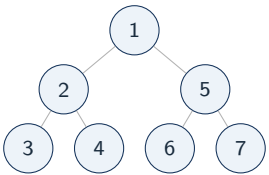
BFS 选 $A \to D$：1 条边，代价 10

本页要点

BFS 优化的是“边数”，不是“边权和”；边权不同必须先换成最短路算法。

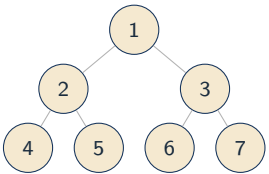
BFS 与 DFS：两种搜索顺序

DFS：一条路走到底



访问顺序 $1, 2, 3, 4, 5, 6, 7$

BFS：一层一层



访问顺序 $1, 2, 3, 4, 5, 6, 7$

	DFS	BFS
数据结构	栈 / 递归	队列
扩展顺序	一条路走到底，再回溯	按距离逐层扩展
最短路	无权图不保证第一次最优	第一次到达即最短
空间	与递归深度有关	与最宽的一层有关
适用	枚举方案、回溯、连通块	最少步数、多源扩散

BFS 能解决哪些问题

四类常见问题

1. 无权最短路：迷宫、棋盘、数轴上的最少操作。
2. 连通块：细胞、水洼、城堡房间。
3. 层数 / 扩散：火灾蔓延、病毒传播、多源最近距离。
4. 状态最少操作：把“局面”看成一个点，一次操作看成一条边。

拿到题先问三句

1. 状态是什么？一维、二维还是三维？
2. 一次转移的代价都相同吗？
3. 题目要的是“最少”还是“全部方案”？

本页要点

先判断“能不能用 BFS”，再动手写；三条适用条件比代码本身更重要。

本章题目地图：五种外壳

外壳	状态	本章题目
网格	(x, y)	细胞、Lake Counting、The Castle、仙岛求药、走迷宫
三维	(层, 行, 列)	Dungeon Master
数轴	位置 x	抓住那头牛
棋盘	(x, y)	Knight Moves、最少步数
路径	(x, y) 加前驱	迷宫问题

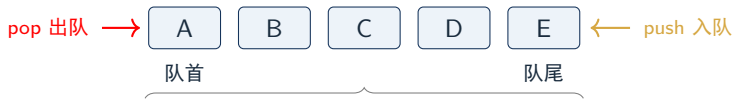
- 五种外壳的 BFS 骨架完全一样，变的只是状态和转移。
- 先把题目归到某一种外壳，再套对应的方向数组和边界条件。
- 路径题多一个 `pre` 数组，其余步骤与普通 BFS 相同。

本页要点

题目千变万化，状态和转移只有这几种；归类之后再动手。

队列是什么：先进先出

- 队列（queue）是只允许在队尾加入、在队首取出的线性表。
- 特点：先进先出（First In, First Out，FIFO）。
- 生活中的例子：食堂排队，先来的人先打到饭；后来的人只能排在队尾。
- 两个基本动作：push 入队（放到队尾）；pop 出队（删掉队首）。
- 队列不能从中间插入，也不能随机访问第 k 个元素。



本页要点

队列的“先进先出”正好对应 BFS 的“先发现、先扩展”。

STL queue 的基本用法

```cpp
#include <queue>
#include <iostream>
using namespace std;

int main() {
    queue<int> q; // 空队列
    q.push(10); // 入队：10
    q.push(20); // 入队：10, 20
    q.push(30); // 入队：10, 20, 30
    cout << q.front() << '\n'; // 队首 10
    cout << q.back() << '\n'; // 队尾 30
    q.pop(); // 弹出队首，剩 20, 30
    cout << q.size() << '\n'; // 元素个数 2
    cout << q.empty() << '\n'; // 0 表示不空
    return 0;
}
```

queue 操作速查与两个坑

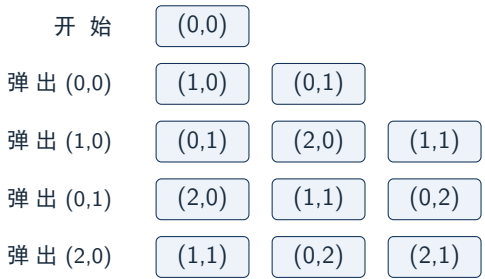
操作	作用	注意
`q.push(x)`	把 x 放到队尾	入队
`q.front()`	返回队首元素	只查看，不删除
`q.back()`	返回队尾元素	只查看，不删除
`q.pop()`	删除队首元素	不返回被删的值
`q.empty()`	队列为空返回 `true`	循环条件常用
`q.size()`	返回元素个数	类型是 `size_t`

- 坑 **1**：对空队列调用 `front()` 或 `pop()` 是未定义行为。写循环时一定先写 `while (!q.empty())`。
- 坑 **2**：只写 `front()` 不写 `pop()`，队首永远不变，程序死循环。
- 正确顺序：先 `front()` 取出元素，再 `pop()` 删除它；需要元素本身时先存到局部变量。

本页要点

`front` 与 `pop` 必须成对出现：一个取，一个删。

队列里到底发生了什么



以 3×3 全开放网格、起点 $(0,0)$ 为例，方向顺序为右、左、下、上。每一行左边是队首，右边是队尾；每次只从队首取一个，把新邻居放到队尾。

本页要点

老元素出完，才轮到下一层的新元素，所以队列天然按距离排序。

队列里存什么：两种常见写法

写法一：只存坐标，距离用数组

```
int dist[N][M];
dist[x][y] = dist[px][py] + 1;
```

- 代码短，任意格子的距离都能查。
- dist 同时充当“是否访问”的标记。
- 本章绝大多数题用这种写法。

写法二：结构体带步数

```
struct Node {
    int x, y, d;
};
queue<Node> q;
q.push({sx, sy, 0});
```

- 适合只关心终点距离、不想开距离数组的场合。
- 判重仍需 vis 数组，否则会重复入队。

本页要点

推荐写法一：dist 数组既存答案又判重，逻辑最清楚。

结构体元素怎么取出来

```
struct Node {
    int x, y, step; // 坐标与步数
};

queue<Node> q;
q.push({sx, sy, 0}); // 起点入队，步数为 0

while (!q.empty()) {
    Node cur = q.front(); // 先取整个元素
    q.pop(); // 再弹出队首
    int x = cur.x;
    int y = cur.y;
    int d = cur.step;
    // 用 x, y, d 扩展邻居
}
```

- cur 是队首的副本；pop() 之后 cur 仍然有效。

手写数组队列：head 与 tail

- 有些考场或老教材不允许用 STL，可以用数组模拟队列。
- 用两个下标：`head` 指向队首，`tail` 指向“下一个空位”。
- 队列区间是 $[\texttt{head}, \texttt{tail})$，为空时 `head == tail`。
- 入队：写 `q[tail]`，再 `++tail`。
- 出队：取 `q[head]`，再 `++head`。



灰色 = 已出队；
有效元素是 C, D, E

本页要点

`head` 之前是已出队，`tail` 之后是还没用到的空位。

数组队列的代码骨架（1/2）

```
const int N = 100005;
int qx[N], qy[N];
int head = 0, tail = 0; // 队列区间 [head, tail)

qx[tail] = sx; // 起点入队
qy[tail] = sy;
++tail;
```

- head 和 tail 一开始都为 0，表示空队列。
- 入队先写 qx[tail]、qy[tail]，再 ++tail。

数组队列的代码骨架（2/2）

```
while (head < tail) { // 队列不空
    int x = qx[head]; // 取队首
    int y = qy[head];
    ++head; // 弹出队首
    for (int k = 0; k < 4; ++k) {
        int nx = x + dx[k];
        int ny = y + dy[k];
        // 判断合法、未访问后：
        qx[tail] = nx; // 新状态放到队尾
        qy[tail] = ny;
        ++tail;
    }
}
```

队列的常见错误

1. 空队列取队首：没有先判 `empty()` 就调用 `front()`。
2. 忘记弹出：只取 `front()` 不 `pop()`，队首永远不变。
3. 弹出后才取元素：先 `pop()` 再 `front()`，拿到的是下一个元素。
4. 把队列当数组：`q[0]` 这种写法不存在，队列不能随机访问。
5. 队列开太小：手写数组队列时 `tail` 超出数组范围。
6. 多组数据共用队列：上一组残留元素混进下一组。

自检方法

在每次 `pop()` 前后打印队首和队列大小，观察是否按“先进先出”变化。

本页要点

队列的错误大多来自“空、忘弹、弹早了”这三件事。

距离数组 dist：把“第几层”记下来

- `dist[x][y]` 表示从起点走到 (x, y) 的最少步数。
- 初始化：起点为 0，其余为 -1（表示还没访问）。
- 扩展时：`dist[nx][ny] = dist[x][y] + 1`。
- 因为 BFS 按层扩展，第一次写下的值就是最短距离，之后不再修改。

```
vector<vector<int>> dist(n,
    vector<int>(m, -1));
dist[sx][sy] = 0;
q.push({sx, sy});
// 扩展到一个合法邻居
dist[nx][ny] = dist[x][y] + 1;
q.push({nx, ny});
```

本页要点

`dist` 的值就是层号；同层同距离，层层递增。

dist 的数值长什么样

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

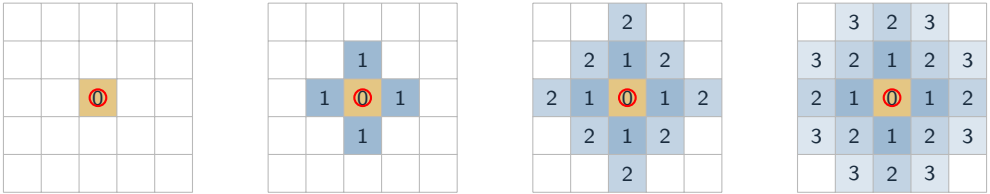
S

- 起点 `dist` 为 0；与起点相邻的格子为 1，再外一圈为 2。
- 颜色越深表示距离越大；同一个数字的所有格子属于同一层。
- 某格若始终是 -1，说明从起点走不到它。

本页要点

把 `dist` 打印出来，BFS 的“层”就一眼可见。

逐层扩展：四张图看懂 BFS



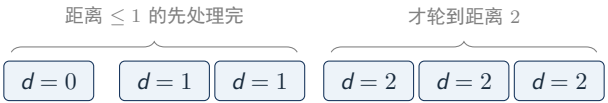
- 起点先入队；每弹出一个距离 d 的格子，就把它的未访问邻居（距离 $d+1$）入队。
- 四张图依次是：只有起点、扩完第 1 层、扩完第 2 层、扩完第 3 层。
- 同一张图里所有新格子的距离都相同；颜色由浅到深就是层号增加。

本页要点

BFS 的执行过程就是“把当前层全部处理完，再处理下一层”。

为什么第一次到达就是最短

- 队列中的距离单调不降：开始只有距离 0 的起点；每弹出一个距离 d 的状态，它未访问的邻居距离都是 $d+1$，放到队尾后仍然排在所有距离 $\le d$ 的状态后面。
- 因此某个格子第一次被访问时，不可能存在更短的路线：若存在更短路线，它的前驱应当更早出现在队列中，并更早把这个格子入队。
- 这个结论依赖一个前提：每条边的代价相同，这里每走一步都算 1。如果边权不同，就要用最短路算法（如 Dijkstra）。
- 所以 BFS 能在 $O(\text{状态数} \times \text{转移数})$ 内求出无权图最短路。



本页要点

第一次到达即最短，是 BFS 最重要的性质；它把“搜索”变成了“按层数数”。

最短步数 BFS 的五个固定动作

1. 初始化：起点距离置 0，其余置 -1；起点入队。
2. 取队首：在队列非空时，取出并弹出队首状态 (x, y)。
3. 枚举邻居：用方向数组生成所有一步能到的 (nx, ny)。
4. 过滤：依次判断越界、障碍、是否已访问；不合法就跳过。
5. 标记入队：先写 `dist[nx][ny] = dist[x][y] + 1`，再 `push`。

循环结束后的答案

- 起点到终点的最少步数：`dist[ex][ey]`。
- 若 `dist[ex][ey] == -1`，说明终点不可达，按题目要求输出。

本页要点

五个动作背下来，剩下的只是“地图怎么读、答案怎么输出”。

多源 BFS：让多个起点同时出发

- 多源 **BFS**：一开始把所有源点都入队，距离都设为 0。
- 等价于加一个到所有源点距离为 0 的“超级源点”。
- 适用：求每个点到最近源点的距离，如火灾蔓延、病毒传播、多仓库配送。
- 正确性：所有源点距离为 0，仍按层扩展，第一次到达就是最近。

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

红圈 = 源点；数字 = 到最近源点的距离

本页要点

多源 BFS 的关键：所有源点在第一轮就全部入队，而不是逐个跑 BFS。

多源 BFS：初始化所有源点

```
vector<vector<int>> dist(n, vector<int>(m, -1));
queue<pair<int, int>> q;

for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
        if (a[i][j] == 1) { // 1 表示源点
            dist[i][j] = 0; // 源点距离为 0
            q.push({i, j}); // 所有源点先入队
        }
    }
}
```

- 所有源点都在 BFS 开始前入队，队列的第一层就是全部源点。
- 源点数量多时，这个初始化本身就是 $O(\text{源点数})$。

多源 BFS：扩展部分

```
while (!q.empty()) {
    auto [x, y] = q.front();
    q.pop();
    for (int k = 0; k < 4; ++k) {
        int nx = x + dx[k];
        int ny = y + dy[k];
        if (nx < 0 || nx >= n || ny < 0 || ny >= m
            || a[nx][ny] == 1 || dist[nx][ny] != -1) {
            continue; // 越界、障碍、已访问
        }
        dist[nx][ny] = dist[x][y] + 1;
        q.push({nx, ny});
    }
}
```

多源 BFS 小例：火灾蔓延

- 一片 5×5 的草地，两个位置同时起火，火每单位时间向上下左右蔓延一格。
- 问每个格子第几时刻被烧到。
- 把两个火源都作为源点，距离都为 0，做一次多源 BFS。
- 图中数字就是被烧到的时刻；两个火源在中间“相遇”，相遇处的数字是两边距离的较小值。

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

红圈 = 起火点；数字 = 被烧到的时刻

本页要点

多源 BFS 的答案不是“到某个源点的距离”，而是“到所有源点距离的最小值”。

多源 BFS 的适用条件与易错点

适用条件

- 有多个起点，且它们同时开始扩散；
- 每次扩散代价相同；
- 要求每个位置到最近起点的距离。

易错点

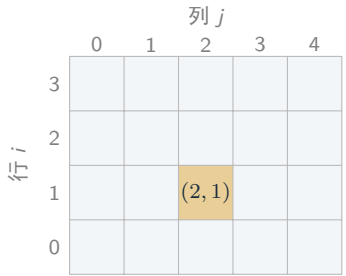
- 只把第一个源点入队，漏掉其余源点；
- 源点距离没置 0，答案整体偏移；
- 把多源 BFS 写成对每个源点各跑一次，时间超限；
- 障碍物判断与源点判断顺序写反。

本页要点

多源 BFS 是“一次 BFS，多个起点”，不是“多次 BFS”。

网格怎么表示：行列坐标

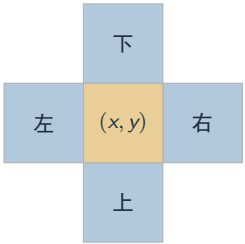
- 网格有 n 行、m 列，行从上到下编号，列从左到右编号。
- 本章统一使用 0 基下标：行 $0 \sim n-1$，列 $0 \sim m-1$。
- 字符地图用 `vector<string> g(n)` 存，`g[i][j]` 是第 i 行第 j 列的字符。
- 坐标 (i, j) 对应第 i 行、第 j 列；不要把行列写反。



本页要点

先约定“行是 i、列是 j、都从 0 开始”，后面所有代码才不会写反。

四连通：方向数组



上下左右四个邻居，斜对角不算

k	dx[k]	dy[k]	方向
0	1	0	右
1	−1	0	左
2	0	1	下
3	0	−1	上

- dx 与 dy 必须按下标配对。
- 方向顺序不影响最短距离，只影响同层内的访问顺序。

本页要点

方向数组把“四种走法”变成一次循环，是最不容易写错的写法。

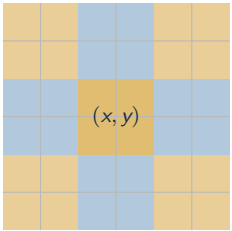
方向数组的代码写法

```
const int dx[4] = {1, -1, 0, 0}; // 右、左、下、上
const int dy[4] = {0, 0, 1, -1};

for (int k = 0; k < 4; ++k) {
    int nx = x + dx[k];
    int ny = y + dy[k];
    // (nx, ny) 就是 (x, y) 的四个邻居之一
}
```

- 四连通就写四个方向；要八连通再另写一套，不要混用。
- 方向数组的下标从 0 到 3，循环写成 `k < 4`。

八连通：水洼与扫雷



蓝：上下左右；金：斜对角；共 8 个邻居

- 八连通：上下左右加四个斜对角，共 8 个邻居。
- 写法：双重循环 `dx=-1..1`、`dy=-1..1`，跳过 $(0,0)$。
- 适用：Lake Counting 水洼、扫雷、图像连通区域。
- 易错：忘记跳过 $(0,0)$，会把当前格当成自己的邻居。

本页要点

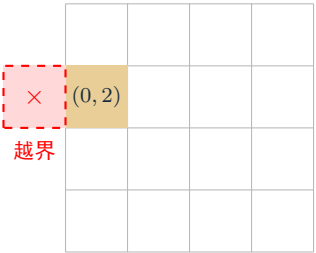
四连通写数组，八连通写双重循环；两者不能混用。

八连通的代码写法

```
for (int dx = -1; dx <= 1; ++dx) {
    for (int dy = -1; dy <= 1; ++dy) {
        if (dx == 0 && dy == 0) {
            continue; // 跳过自己
        }
        int nx = x + dx;
        int ny = y + dy;
        // 再判断越界、障碍、是否访问
    }
}
```

- 变量名 dx、dy 会遮蔽全局的方向数组，八连通代码里要另起名字。
- 八连通的邻居有 8 个，判重和边界判断一个都不能少。
- 若题目只要四连通，把循环范围改成四个方向数组即可。

边界判断：先越界，再障碍，再判重



左边界外：$nx = -1$

1. 越界：$nx < 0$、$nx \geq n$、$ny < 0$、$ny \geq m$。
2. 障碍：`g[nx][ny] == '#'` 或数字矩阵的障碍值。
3. 已访问：`dist[nx][ny] != -1` 或 `vis[nx][ny]`。

- 顺序不能反：越界判断必须放在访问数组之前，否则会读到负下标。
- 三个判断用 `continue` 跳过，不要写成嵌套 `if`。
- 逻辑或 `||` 有短路特性，越界成立时后面的下标不会执行。

边界判断的代码

```
for (int k = 0; k < 4; ++k) {
    int nx = x + dx[k];
    int ny = y + dy[k];
    if (nx < 0 || nx >= n || ny < 0 || ny >= m) {
        continue; // 越界
    }
    if (g[nx][ny] == '#') {
        continue; // 障碍
    }
    if (dist[nx][ny] != -1) {
        continue; // 已访问
    }
    dist[nx][ny] = dist[x][y] + 1;
    q.push({nx, ny});
}
```

网格 BFS 完整模板（1/3）

```cpp
#include <bits/stdc++.h>
using namespace std;

const int dx[4] = {1, -1, 0, 0}; // 右、左、下、上
const int dy[4] = {0, 0, 1, -1};

int main() {
    int n, m;
    cin >> n >> m;
    vector<string> g(n);
    for (int i = 0; i < n; ++i) {
        cin >> g[i]; // 逐行读地图
    }
```

网格 BFS 完整模板（2/3）

```
int sx, sy, ex, ey;
cin >> sx >> sy >> ex >> ey; // 起点、终点
vector<vector<int>> dist(n, vector<int>(m, -1));
queue<pair<int, int>> q;
dist[sx][sy] = 0; // 起点距离为 0
q.push({sx, sy});

while (!q.empty()) {
    auto [x, y] = q.front(); // 取队首
    q.pop(); // 弹队首
    for (int k = 0; k < 4; ++k) {
        int nx = x + dx[k];
        int ny = y + dy[k];
```

网格 BFS 完整模板（3/3）

```
            if (nx < 0 || nx >= n || ny < 0 || ny >= m
                || g[nx][ny] == '#' || dist[nx][ny] != -1) {
                continue; // 越界、障碍或已访问
            }
            dist[nx][ny] = dist[x][y] + 1;
            q.push({nx, ny}); // 先标记，再入队
        }
    }
    cout << dist[ex][ey] << '\n'; // 不可达时为 -1
    return 0;
}
```

模板的五个空位

1. 地图大小：n 行 m 列，行列下标从 0 开始。
2. 方向数组：四连通、八连通还是十二连通？
3. 起点：题目给的是坐标、字符 `S`，还是多个源点？
4. 可走条件：字符 `'#'`、数字 1、还是别的障碍值？
5. 答案：输出 `dist`、`dist`$+1$、还是固定字符串？

套模板的正确姿势

先把这五个空位在纸上写清楚，再往模板里填；不要一上来就复制代码，否则很容易把行列、起点、障碍物改错。

- 模板中的 `int` 足够存 $10^4 \sim 10^6$ 级别的步数；更大范围或累加代价时改用 `long long`。
- 如果地图是数字矩阵，`vector<string>` 要换成 `vector<vector<int>>`，读入方式也随之改变。

步数、方格数与输出格式

- dist 的含义是移动次数：从起点走到终点用了几步。
- 有的题要求“经过的方格数”，此时答案 $=$ dist $+1$（起点和终点都算）。
- 有的题按样例输出步数，不加 1。一定先用样例核对。

三个真实例子

走迷宫：5×5 样例，最短路 8 步，输出 9（方格数，含起点终点）。
仙岛求药：8×8 样例，最短路 10 步，输出 10（按样例，输出步数）。
抓住那头牛：输出分钟数，恰好等于步数。

- 多组数据时，每组输出各占一行，不要多输出空行。
- 不可达时按题目要求输出 -1、oop!、Trapped! 等固定字符串。

本页要点

写完先跑样例；样例对不上，先检查“要不要 $+1$”。

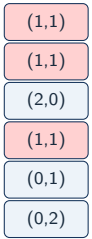
多组数据与读入细节

- 三种结束方式：读到 `0 0 0`（Dungeon Master）、`0 0`（仙岛求药）、先给组数 T（花束、Knight Moves）。
- 每组数据都要重新初始化 `dist/vis`，不能沿用上一组。

```
int T;
cin >> T;
while (T--) {
    int r, c;
    cin >> r >> c;
    vector<string> g(r);
    for (int i = 0; i < r; ++i) {
        cin >> g[i]; // 每行一个字符串
    }
    // 在这里重新建 dist，跑 BFS
}
```

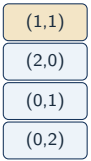
没有判重会怎样

出队才标记：重复入队



同一格出现 3 次

入队就标记：每格只一次



无重复

- 没有判重时，同一个格子会被不同邻居反复入队，队列大小可能指数级膨胀。
- 出队才标记，只能保证“处理时不重复”，不能保证“入队时不重复”。
- 标准做法：入队之前就把距离/访问标记写好。

本页要点

判重的时机：入队之前；判重的对象：整个状态，而不只是坐标的一部分。

入队时标记的正确写法

```
dist[sx][sy] = 0;
q.push({sx, sy});
while (!q.empty()) {
    auto [x, y] = q.front();
    q.pop();
    for (int k = 0; k < 4; ++k) {
        int nx = x + dx[k];
        int ny = y + dy[k];
        if (nx < 0 || nx >= n || ny < 0 || ny >= m
            || g[nx][ny] == '#' || dist[nx][ny] != -1) {
            continue; // 越界、障碍、已访问
        }
        dist[nx][ny] = dist[x][y] + 1; // 先标记
        q.push({nx, ny}); // 再入队
    }
}
```

三种常见的判重写法

写法	代码	优点	注意
距离数组	`dist[nx][ny] != -1`	距离兼作标记，最常见	需要开 int 数组
访问数组	`vis[nx][ny]`	语义清楚，不关心距离时更省	多组数据要清空
修改地图	`g[nx][ny] = '#'`	省一个数组，代码短	不能恢复地图,适合只跑一次

- 本章推荐“距离数组”：一个数组同时回答“来没来过”和“走了几步”。
- 如果只求连通块、不需要距离，用 `bool vis` 或改地图更简洁。
- 三维、带方向、带钥匙的状态，判重数组也要跟着加维度。
- 无论哪种写法，都必须在入队之前完成标记。

本页要点

判重数组的维度 = 状态的维度；状态有几个分量，数组就开几维。

状态是什么：不止是二维坐标

题目	状态	转移数	状态规模
细胞	(x, y)	4	$nm \le 10^4$
Dungeon Master	(层, 行, 列)	6	$lrc \le 27000$
抓住那头牛	数轴位置 x	3	$\le 2 \times 10^5$
Knight Moves	(x, y)	8	$L^2 \le 9 \times 10^4$
最少步数	(x, y)	12	10^4
迷宫问题	(x, y)	4	25

- 状态设计的原则：把“决定下一步能怎么走”的信息全部放进状态。
- 状态越多，数组越大；先估算状态数，再决定开多大的数组。
- 若状态还带方向、钥匙、剩余步数等，就要开更多维（后续课程）。
- 判重数组的每一维都要和状态分量一一对应，不能少开。

本页要点

先写清楚“状态里有哪些量”，再决定队列元素和判重数组的维度。

多组数据：清空要彻底

- 每组数据重新申请 `vector` 或重新 `memset`，不要沿用上一组。
- 队列在上一组 BFS 结束时已经为空，但最好在每组开头重新定义。
- 全局数组适合单组数据；多组数据用局部 `vector` 更安全。
- 输入结束条件要读对：三个零、两个零、还是先读组数 T。
- 输出格式统一：不可达输出题目指定的字符串，不要混用。

检查方法

把同一组数据复制两遍接在一起，若两遍输出相同，说明清空基本正确。

本页要点

多组数据最怕“上一组的地图”混进下一组；局部变量 + 重新初始化最稳。

判重与状态设计的常见错误

1. 标记太晚：出队才标记，同一状态多次入队。
2. 标记错对象：只标记坐标，忘了标记方向、钥匙等状态分量。
3. 数组开小：状态规模估算错误，越界访问。
4. 多组不清空：上一组的 `dist/vis` 残留。
5. 起点没标记：起点可能被邻居再次入队。
6. 障碍物当可走：地图字符判断写反，走进墙里。

自检

把队列里每次入队的元素打印出来，检查同一个状态是否出现两次；如果出现，问题一定在判重时机或判重维度。

本页要点

判重是 BFS 正确性的半条命；另外半条是边界和方向。

例 8.2 | 题意与样例

题意

给定一个 $n \times m$ 的数字矩阵，$1 \sim 9$ 表示细胞，0 表示空白；上下左右相邻的细胞属于同一个细胞。求细胞个数。

样例输入

```
4 10
0234500067
1034560500
2045600671
0000000089
```

样例输出：4

- 每个非零格是一个点，相邻的非零格之间连一条边。
- 细胞个数就是这张图的连通块个数。
- $1 \leq n, m \leq 100$，最多 10^4 个格子。

例 8.2 | 样例手推与思路

0	2	3	4	5	0	0	0	6	7
1	0	3	4	5	6	0	5	0	0
2	0	4	5	6	0	0	6	7	1
0	0	0	0	0	0	0	0	8	9

蓝、金、绿、红 $=$ 四个细胞；灰色 $=0$

1. 按行、按列扫描每个格子。
2. 遇到未访问的非零格：答案加一，把它入队并标记。
3. 从它出发 BFS，把上下左右相邻的非零格全部标记。
4. 队列为空时这一块结束，继续扫描下一格。

为什么不会重复

每个格子只在第一次被发现时入队，入队即标记；之后任何方向都不会再访问它。

例 8.2 | 完整程序（1/3）

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    vector<string> g(n);
    for (int i = 0; i < n; ++i) {
        cin >> g[i]; // 每行一个字符串
    }
    vector<vector<char>> vis(n, vector<char>(m, 0));
    const int dx[4] = {1, -1, 0, 0};
    const int dy[4] = {0, 0, 1, -1};
    int cells = 0; // 细胞个数
```

例 8.2 | 完整程序（2/3）

```
for (int si = 0; si < n; ++si) {
    for (int sj = 0; sj < m; ++sj) {
        if (g[si][sj] == '0' || vis[si][sj]) {
            continue; // 空白或已访问
        }
        ++cells; // 发现一个新细胞
        queue<pair<int, int>> q;
        q.push({si, sj});
        vis[si][sj] = 1; // 起点先标记
        while (!q.empty()) {
            auto [x, y] = q.front();
            q.pop();
            for (int k = 0; k < 4; ++k) {
                int nx = x + dx[k];
                int ny = y + dy[k];
```

例 8.2 | 完整程序（3/3）

```
                if (nx < 0 || nx >= n || ny < 0 || ny >= m) {
                    continue; // 越界
                }
                if (g[nx][ny] == '0' || vis[nx][ny]) {
                    continue; // 空白或重复
                }
                vis[nx][ny] = 1; // 入队前标记
                q.push({nx, ny});
            }
        }
    }
    cout << cells << '\n';
    return 0;
}
```

例 8.2 | 复杂度与易错点

- 时间 $O(nm)$，空间 $O(nm)$；$n, m \le 100$ 时非常轻松。
- 地图每行是一个字符串，字符是 `'0'`，不要和数字 0 混淆。
- 只向上下左右扩展，不能把斜对角算进同一个细胞。
- 标记必须放在入队之前，否则同一格可能被多个邻居重复入队。
- 外层循环要扫描每一个格子，不能只从 $(0, 0)$ 开始一次 BFS。
- 输入的行内没有空格，用 `cin >> string` 一次读一整行。

本页要点

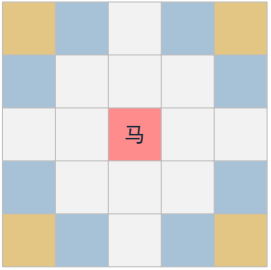
连通块题的固定套路：扫描 $\rightarrow$ 发现新块 $\rightarrow$ BFS 染色 $\rightarrow$ 计数。

例 8.3 | 题意与走法

题意

在 100×100 的棋盘上，马从 A 点或 B 点出发，走向左上角 $(1,1)$。马可以走“日”字，也可以走“田”字。分别求 A、B 到 $(1,1)$ 的最少步数。

- “日”字：$(\pm 1, \pm 2)$ 与 $(\pm 2, \pm 1)$，共 8 种。
- “田”字：$(\pm 2, \pm 2)$，共 4 种。
- 合计 12 种走法，每步代价 1；坐标 $1 \sim 100$。
- 输出两行，先 A 后 B。



蓝：日　金：田

例 8.3 | 样例手推与思路

	0	1	2	3	4	5	6	7	8	步数
A	$(12,16)$	$(13,14)$	$(12,12)$	$(11,10)$	$(9,9)$	$(7,7)$	$(5,5)$	$(3,3)$	$(1,1)$	8
B	$(18,10)$	$(17,12)$	$(15,13)$	$(13,12)$	$(11,11)$	$(9,9)$	$(7,7)$	$(5,5)$	$(3,3)$	9

- 表中相邻两点的坐标差都属于“日”或“田”；A 用 8 步，B 用 9 步。
- 两条路线在 $(9,9)$ 后汇合，提示可以从终点反向一次算出所有距离。
- 走法可逆，从 $(1,1)$ BFS 一次预处理，查询 A、B 只需读 `dist`，$O(1)$。

本页要点

反向预处理把“两次 BFS”变成“一次 BFS + 两次查询”。

例 8.3 | 完整程序（1/3）

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    const int N = 100;
    // 前 8 个是“日”，后 4 个是“田”
    const int dx[12] = {1, 1, -1, -1, 2, 2, -2, -2, 2, 2, -2, -2};
    const int dy[12] = {2, -2, 2, -2, 1, -1, 1, -1, 2, -2, 2, -2};
    vector<vector<int>> dist(N + 1, vector<int>(N + 1, -1));
    queue<pair<int, int>> q;
    dist[1][1] = 0; // 从终点反向出发
    q.push({1, 1});
```

本页要点

方向数组 12 项要按顺序配对；“田”是 $(\pm 2, \pm 2)$，不是 $(\pm 2, 0)$。

例 8.3 | 完整程序（2/3）

```
while (!q.empty()) {
    auto [x, y] = q.front();
    q.pop();
    for (int k = 0; k < 12; ++k) {
        int nx = x + dx[k];
        int ny = y + dy[k];
        if (nx < 1 || nx > N || ny < 1 || ny > N) {
            continue; // 越界
        }
        if (dist[nx][ny] != -1) {
            continue; // 已访问
        }
        dist[nx][ny] = dist[x][y] + 1;
        q.push({nx, ny});
    }
}
```

例 8.3 | 完整程序（3/3）

```
    int ax, ay, bx, by;
    cin >> ax >> ay >> bx >> by;
    cout << dist[ax][ay] << '\n';
    cout << dist[bx][by] << '\n';
    return 0;
}
```

- 查询顺序：先 A 后 B，每行一个答案。
- 题目保证 A、B 在棋盘内，且一定能走到 $(1,1)$。
- 若存在走不到的点，`dist` 为 -1，要按题目要求处理。

本页要点

预处理一次，查询 $O(1)$；这是“多询问最短路”的常用套路。

例 8.3 | 复杂度与易错点

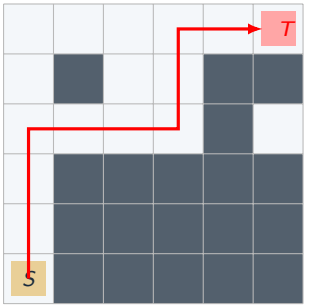
- 状态数 $100 \times 100 = 10^4$，每状态 12 条转移，时间 $O(10^4 \times 12)$；空间 $O(10^4)$，非常宽裕。
- 易错一：把“田”写成 $(\pm 2, 0)$、$(0, \pm 2)$。田是斜着走两格，即 $(\pm 2, \pm 2)$。
- 易错二：坐标从 0 开始循环，导致 $(1, 1)$ 被忽略或越界。
- 易错三：只输出一行，或先输出 B 再输出 A。
- 易错四：方向数组只写 8 个“日”，漏掉“田”，答案会偏大。
- 易错五：忘记把 `dist` 初始化为 -1，已访问判断失效。

本页要点

先枚举全部 12 种走法，再用 100×100 的表格核对一遍。

拓展 | BFS 怎么记录路径

- BFS 的 `dist` 只告诉“最少几步”，不告诉“怎么走”。
- 增加一个 `pre[x][y]`，记录 (x, y) 是从哪个格子走过来的。
- 扩展时：`pre[nx][ny] = {x, y}`。
- 到达终点后，从终点沿 `pre` 一路回溯到起点，再把路径反转。
- 图中红线就是一条最短路径；每个格子只记录一个前驱，所以路径唯一。



灰色 = 墙；红线 = 一条最短路径

本页要点

路径记录 = BFS + 前驱数组；回溯时从终点倒着走回起点。

拓展 | 记录前驱

```
if (dist[nx][ny] == -1) {
    dist[nx][ny] = dist[x][y] + 1;
    pre[nx][ny] = {x, y}; // 从 (x, y) 走来
    q.push({nx, ny});
}
```

- `pre[nx][ny]` 只在第一次访问时写入，之后不再修改。
- 每个格子只记录一个前驱，所有前驱连起来就是一条最短路径。
- `pre` 的元素类型与队列相同，都是 `pair<int, int>`。

本页要点

前驱数组只在第一次访问时写入；这是路径唯一的保证。

拓展｜回溯路径

```
vector<pair<int, int>> path;
int x = ex, y = ey;
while (x != sx || y != sy) {
    path.push_back({x, y});
    auto [px, py] = pre[x][y];
    x = px;
    y = py;
}
path.push_back({sx, sy});
reverse(path.begin(), path.end()); // 反转成从起点到终点
```

- 从终点出发，每次跳到 pre 记录的前驱，直到起点。
- 回溯得到的是“终点 $\rightarrow$ 起点”，必须 reverse 才是正向路径。
- 若终点不可达，pre 中没有有效前驱，应直接输出无解。

八个高频错误

1. 忘记判重：同一格反复入队，程序变慢甚至死循环。
2. 出队才标记：重复入队，距离可能被覆盖。
3. 边界写反：`nx >= n` 写成 `nx > n`，越界访问。
4. 方向数组漏项：八连通只写四方向，或“田”字漏写。
5. 步数含义混淆：该输出方格数时忘了 $+1$，该输出步数时多加了 1。
6. 多组数据不清空 `dist`，第二组答案错误。
7. 读入方式不对：整行字符串用 `cin >> char` 会读错。
8. 输出格式不对：`oop!`、`Trapped!`、两行坐标等细节。

本页要点

每一条都对应一个可以自己构造的小数据；写完程序先自测这八种情况。

BFS 五步检查表

1. 状态：队列里存什么？坐标是一维、二维还是三维？
2. 转移：方向数组写全了吗？每条边代价都是 1 吗？
3. 边界：越界、障碍、已访问三个判断的顺序对吗？
4. 标记：是在入队之前标记的吗？
5. 答案：输出 `dist` 还是 `dist+1`？不可达怎么输出？

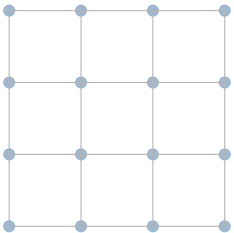
万能骨架

初始化起点 → 入队 → 取队首 → 枚举邻居 → 合法且未访问则标记并入队 → 队列空则结束。

本页要点

把这张表贴在显示器旁，每道题写完逐项打勾。

复杂度：把网格看成一张图



4×4 网格：$V = 16$ 个状态，$E = 24$ 条边

- BFS 中每个状态最多出队一次，每条边最多被检查两次。
- 时间 $O(V + E)$；网格里 $E \le 4V$，所以 $O(V) = O(nm)$。
- 空间：`dist`/`vis` 数组 $O(V)$，队列最多 $O(V)$。
- 三维、数轴、棋盘只是把 V 换成对应的状态数。

本页要点

BFS 的复杂度与路径长度无关，只与状态数和转移数有关。

复杂度速查表

题目	状态数	每状态转移	时间复杂度
细胞	nm	4	$O(nm)$
Lake Counting	NM	8	$O(NM)$
The Castle	mn	4	$O(mn)$
Dungeon Master	lrc	6	$O(lrc)$
仙岛求药	MN	4	$O(MN)$ 每组
走迷宫 / 走出迷宫	RC	4	$O(RC)$
抓住那头牛	K	3	$O(K)$
献给阿尔吉侬的花束	RC	4	$O(RC)$ 每组
Knight Moves	L^2	8	$O(L^2)$ 每组

- BFS 的时间与“状态数 $\times$ 转移数”成正比，与路径长度无关。
- 空间主要花在 `dist`/`vis` 数组和队列上，同为 O(状态数)。

一页纸总结：BFS 的四个固定动作

1. 定状态：队列里存什么？一维位置、二维坐标，还是三维 (层, 行, 列)？先估算状态数，再决定数组开多大。
2. 列转移：把走法写成方向数组；四方向、六方向、八方向、十二方向都要写全，并先判越界、再判障碍。
3. 判重：用 $\texttt{dist} \neq -1$ 或 vis 标记；必须在入队时标记，保证每个状态只进队一次。
4. 读答案：输出 dist 还是 $\texttt{dist}+1$？不可达输出什么？多组数据是否清空、是否每组换行？

本页要点

按“起点入队 $\rightarrow$ 取队首 $\rightarrow$ 枚举邻居 $\rightarrow$ 标记入队”的顺序写，BFS 只剩方向数组和输出格式两个变量。

练习路线：从模板到变式

1. 先套模板：走迷宫、走出迷宫、仙岛求药、献给阿尔吉侬的花束。
2. 再练连通块：Lake Counting、The Castle、细胞（例题）。
3. 再换状态：Dungeon Master（三维）、抓住那头牛（数轴）、Knight Moves（棋盘）、最少步数（例题）。
4. 最后练路径：迷宫问题，需要记录前驱并输出整条路线。

刷题建议

每题写完先跑样例，再自己造三组：最小、最大、不可达。把“状态、转移、判重、答案”四行写在纸上，再开始写代码。

本章练习题索引 | 1–5 / 10

1. P5259 | Dungeon Master（一本通 1248）
   三维迷宫，六方向，多组数据读到 0 0 0。
2. P5260 | Lake Counting（一本通 1249）
   八连通水洼计数，照搬连通块模板。
3. P5261 | The Castle（一本通 1250）
   用二进制位判断四面墙，求房间数与最大面积。
4. P5262 | 仙岛求药（一本通 1251）
   多组数据，输出步数，不可达输出 -1。
5. P5263 | 走迷宫（一本通 1252）
   模板题，输出经过的方格数（含起点终点）。

点击蓝色题名进入学校 OJ；本章全部 10 道练习都列在索引中。

本章练习题索引 | 6–10 / 10

6. P5264 | 抓住那头牛（一本通 1253）
   数轴上的 BFS，转移为 $X-1$、$X+1$、$2X$。
7. P5265 | 走出迷宫（一本通 1254）
   从 S 到 T 的最少步数，模板题。
8. P5266 | 迷宫问题（一本通 1255）
   记录前驱，输出从 $(0,0)$ 到 $(4,4)$ 的整条路径。
9. P5267 | 献给阿尔吉侬的花束（一本通 1256）
   T 组数据，不可达输出 oop!。
10. P5268 | Knight Moves（一本通 1257）
    棋盘 $0 \sim L-1$ 坐标，八个“日”字方向。

点击蓝色题名进入学校 OJ；本章全部 10 道练习都列在索引中。