BFS 最短路与双向 A 星

队列层序波纹扩散、多源与 0-1 BFS、双向搜索与启发式估价

HGNU-ACM 实验室算法组

配套：无权最短路铁律 + 多源/0-1 BFS + 双向/A* + 3 道真题精讲 + 7 题练习





本章目录（1/2）

广度优先搜索与无权最短步

进阶变形一：多源 BFS 与 0-1 BFS

进阶变形二：双向 BFS 与启发式 A* 搜索

前三节承接《DFS 回溯与剪枝优化》，讲清“求最少步数为什么要换队列 BFS”、多源与双端队列 0-1 BFS，以及双向 BFS 与 A* 估价函数。

本章目录（2/2）

三大 BFS 经典真题精讲

常见错误、检查清单与练习安排

后两节逐题精讲《走迷宫》《抓住那头牛》《细胞》，附上常见报错定位、六必查清单与下一篇《栈与单调栈》预告。

承接上一章：求“最少走几步”时，为什么 DFS 不灵了？

小例子：在 3×3 空旷网格里，从左上角 $(0,0)$ 走到 $(1,0)$ 最少几步？

答案显然是往下走 1 步就到！但如果我们用上一章的 DFS（比如优先“向右 $\rightarrow$ 向下 $\rightarrow$ 向左 $\rightarrow$ 向上”）：

- DFS 会先一口气向右走到 $(0,2)$，再往下走到 $(2,2)$，再往左、再往上……绕了一个大圈走了 7 步才碰到 $(1,0)$！
- 而且拿到这个 7 步之后，DFS 根本不敢停，因为它不知道别的岔路里还有没有 5 步、3 步甚至 1 步的走法，只能退回去把整张地图所有走法全试一遍！
- 上一章 DFS 擅长回答“共有多少种走法”；而问“最少走几步到达”时，必须换一种搜索顺序！

本页要点

DFS 像闭眼钻胡同的老鼠，第一次撞到终点往往绕了远路；求最少步数需要按距离一圈一圈找！

生活类比：往湖面扔一块石头，水波怎么扩散？

- 想象在平静的湖面扔进一块石头：
  - 第 0 秒：只有落水点（起点）湿了，距离 0；
  - 第 1 秒：水波传到所有离起点恰好 1 步的位置；
  - 第 2 秒：水波传到所有离起点恰好 2 步的位置……
- 如果目标小船停在湖面上，水波第一次碰见小船的时刻，一定是它离落水点最近的步数，根本不需要等后面的波浪再撞它一次！

这就是广度优先搜索（BFS）的直觉

先找所有 1 步能到的点，再找所有 2 步能到的点，再找 3 步……首次碰到终点时直接停下，一定是全局最少步数！

本页要点

每步代价相等（都是 1 步）时，水波按“离起点步数”从小到大铺开，先到即最短！

暴力 DFS 为何 TLE：算算网格里的路径数量

在没有障碍物的 $R \times C$ 网格上，如果硬用 DFS 枚举所有不重复路径来找最短步数：

- 每个格子最多有 4 个方向（去掉刚走来的方向还有 3 个）。
- 当网格只有 $20 \times 20 = 400$ 格时，简单路径数量已经突破 3^{100} 量级，哪怕加上最优性剪枝，只要一开始方向选歪，就会在大片空地上疯狂兜圈，远远超过 1 秒约 10^8 次运算的极限，必定 **TLE**！
- 反过来，如果改用 BFS 按层扩散，规定每个格子最多只进队列 1 次：
  - 20×20 网格最多只处理 400 个格子；
  - 1000×1000 网格也只需约 4×10^6 次判断，几十毫秒轻松 AC！

本页要点

每个格子只进队 1 次，BFS 时间复杂度严格是 $O(R \cdot C)$，和路径总数无关！

新术语大白话：BFS、队列与“层（距离）”

- 广度优先搜索（**BFS**）：按“离起点远近”一层一层往外找的搜索方法，先把当前距离的格子全看完，再去下一距离。
- 队列（queue）：就像食堂打饭排队，先排进来的先出去（先进先出）。第 1 层的点排在前面，它们扩展出的第 2 层的点乖乖排到队尾，天然保证不会插队！
- 层（距离 dist）：从起点走到某个格子最少需要走几步，它就属于第几层。
- 无权图：每走一步代价都一样（比如都是 1 分钟或 1 步），没有哪条路更贵或免费。

本页要点

用 queue 排队，就自动把“距离近的先处理、距离远的排后面”的层序关系维持住了！

TikZ 图解：4×4 网格水波按层扩散图

起点 S 在 $(0,0)$，终点 E 在 $(2,3)$，灰色格子 # 是障碍物不能走：

$S(d=0)$	#		#
$d=1$	#		
$d=2$	$d=3$	$d=4$	$E(d=5)$
$d=3$	$d=4$	$d=5$	

第 0 层：起点 $(0,0)$

第 1 层：$(1,0)$

第 2 层：$(2,0)$

一路波纹推到 E，$d=5$！

本页要点

绕开墙 # 后，水波在第 5 层第一次触及 E，立刻得到最短步数 5！

最致命铁律：必须在 push 入队时标记 dist，绝不等到 pop！

很多初学者最容易在这里翻车，我们看一个 2×2 网格从 $(0,0)$ 到 $(1,1)$ 的对比：

错误做法：出队 pop 时才标记

- $(0,0)$ 弹出，把 $(1,0)$ 和 $(0,1)$ 推进队列；
- $(1,0)$ 弹出时，发现 $(1,1)$ 还没标记，把 $(1,1)$ 推进队列；
- 接着 $(0,1)$ 弹出时，发现 $(1,1)$ 还是没标记，又把 $(1,1)$ 推了一遍！
- 在 100×100 网格中，重复次数会翻倍成 2^{100} 次，瞬间 **MLE / TLE**！

正确铁律：push 入队那一刻就标记

- $(1,0)$ 把 $(1,1)$ 塞入队列的那一行，立刻把 `dist[1][1] = 2` 刻上去！
- 等轮到 $(0,1)$ 查看邻居 $(1,1)$ 时，发现 `dist[1][1] != -1`，直接跳过！
- 保证每个格子从始至终最多入队 1 次！

本页要点

口诀：“入队即刻标距离，判重再进队”——把标记写在 `q.push(...)` 紧前面那一行！

手算跟踪表：2×2 网格 BFS 队列变化（含终止步）

从 $(0,0)$ 到 $(1,1)$（四连通无障碍），跟踪队列 q 与 dist 变化，直到队列为空：

步序	弹出队头	队列当前内容（操作后）	dist 修改与说明
初始	-	[(0,0)]	起点入队并立刻置 dist[0][0]=0
1	$(0,0)$	[(1,0), (0,1)]	扩展下、右两格，立即设 dist[1][0]=1，dist[0][1]=1
2	$(1,0)$	[(0,1), (1,1)]	扩展 $(1,1)$，立即设 dist[1][1]=2 并入队
3	$(0,1)$	[(1,1)]	看到 $(1,1)$ 的 dist 已非 -1，跳过不重复入队
4	$(1,1)$	[]	邻居 $(0,1),(1,0)$ 均已访问，弹出后队列为空
5	条件为假	[]	判断 !q.empty() 为假，循环终止！

本页要点

若题目只要到 $(1,1)$ 的最短距离，第 4 步弹出 $(1,1)$ 时就能直接 break 结束；全图遍历则到第 5 步队空结束。

网格方向数组 dx/dy：用循环代替 4 份重复代码

在网格上往上下左右移动时，千万别写 4 个几乎一模一样的 if（既长又容易改漏一个分支）：

方向编号 k	dx[k]（行变化）	dy[k]（列变化）	含义
0	$+1$	0	向下走一格
1	-1	0	向上走一格
2	0	$+1$	向右走一格
3	0	-1	向左走一格

- 新坐标直接写：`int nx = x + dx[k], ny = y + dy[k];`，用 `for (int k = 0; k < 4; ++k)` 一遍扫完四个方向！
- 如果是八连通（含斜对角），只需把方向数组扩展成长度为 8 即可。

本页要点

先写好 dx，dy 常量数组，再通过下标 k 统一处理所有方向，避免复制粘贴出错！

模板代码：标准二维网格 BFS（最短步数）

```
const int dx[4] = {1, -1, 0, 0}, dy[4] = {0, 0, 1, -1};
vector<vector<int>> dist(R, vector<int>(C, -1)); // -1 表示尚未访问过
queue<pair<int, int>> q;
dist[sx][sy] = 0; q.push({sx, sy});              // 起点入队前先标距离 0
while (!q.empty()) {
    auto [x, y] = q.front(); q.pop();
    if (x == tx && y == ty) break;               // 首次出队即保证是最短路，提前停
    for (int k = 0; k < 4; ++k) {
        int nx = x + dx[k], ny = y + dy[k];
        if (nx < 0 || nx >= R || ny < 0 || ny >= C) continue; // 先判越界！
        if (g[nx][ny] == '#' || dist[nx][ny] != -1) continue; // 再判墙和已访问
        dist[nx][ny] = dist[x][y] + 1; q.push({nx, ny}); // 入队前立刻标 dist！
    }
}
```

要点：判断顺序不能反：必须先判 nx，ny 没越界，再读 g[nx][ny]；且入队前立刻标 dist！

为什么对？队列单调性不变量

用三个阶段验证“为什么第一次到达终点的 dist 一定是最少的步数”：

阶段	要验证的不变量
初始化	开始时队里只有起点 (sx, sy)，距离为 0，队内元素距离非递减，且所有距离 0 的可达点都已入队。
保持	若当前队列头部的元素距离全为 d（后面最多跟着若干 $d+1$），弹出距离为 d 的点后，它扩展出的新点距离都是 $d+1$，全部排在队尾；因此队列里的元素距离始终呈现单调不降（一段 d 紧接一段 $d+1$）。
终止	所有步数 $< d^*$ 的点一定比步数为 d^* 的点更早出队。因此目标点首次被标记并出队时，它的 dist 必为全局最小值！

本页要点

队列的“先进先出”和“每次只 $+1$”配合，就像年轮一样严格按 $0 \to 1 \to 2 \to \dots$ 往外长！

上机验证与反事实追问：去掉 break 会怎样？

测试下面 3 组小网格（起点 S，终点 E，输出 dist[tx][ty]）：

- 普通可达：3 3 地图（S.. / .#. / ..E）→ 输出 4。
- 最小边界（起点即终点）：1 1 地图，S 和 E 同在 $(0,0)$ → 刚弹出 $(0,0)$ 就触发 break，输出 0。
- 无解死胡同：2 2 地图（S# / #E）→ 队列变空也到不了 E，保留初值输出 -1。

反事实追问

1. 如果把模板第 7 行的 `if (x == tx && y == ty) break;` 删掉，最后输出的 dist[tx][ty] 还会对吗？
   答：依然对！因为每个点只标一次 dist，只是多扩展了剩余无用格子，稍微多花一点点时间。
2. 如果有些格子的代价是 2 分钟，有些是 1 分钟，还能直接套用这个模板吗？
   答：不能！因为加 2 的点可能排在前面而加 1 的后到，破坏了“首次到达即最短”性质（后面讲 0-1 BFS 和 Dijkstra 会解决）。

易错对照：普通 BFS 四大高频踩坑

错误表现	常见错误写法	正确做法与原因
队列爆内存（**MLE/TLE**）	等到 `q.pop()` 后才写 `dist[x][y]=...`	同一个点被多个邻居重复推进队，必须写在 `q.push` 前
数组越界崩溃（**RE**）	先写 `if (g[nx][ny]=='#')` 再判越界	逻辑短路先算左边的，必须把 `nx<0 || nx>=R...` 放最前
差 1 错误（**WA**）	题目问“经过格子数”却设起点 `dist=0`	问“移动步数”初值 0；问“经过格子数（含起终点）”初值 1
分不清未访问（**WA**）	`dist` 初值设成 0 又把起点也设 0	统一把 `dist` 全初始化为 `-1`，无解一目了然

本页要点

写完 BFS 第一眼先核对这 4 行：初值 -1 了吗？起点 0 还是 1？先判界了吗？入队就打标了吗？

生活引入（多源 BFS）：仓库有两个起火点，火势怎么蔓延？

场景：3×4 仓库里同时有两个位置起火，火每分钟向四邻扩展 1 格

想知道仓库里每个格子最早几分钟被烧到。如果只有一个火源，跑一遍普通 BFS 就够；现在有 k 个火源（甚至上千个），怎么办？

- 笨办法：对第 1 个火源跑一遍完整 BFS，再对第 2 个火源跑一遍……最后每个格子取所有结果的 min。当有 1000 个火源时，复杂度变成 $1000 \times O(RC)$，直接 **TLE**！
- 聪明办法（多源 **BFS**）：既然它们同时在第 0 分钟着火，为什么不一开始把所有火源一起塞进队列，全部设成 `dist = 0`，只跑一次 BFS？

本页要点

多个起点同时出发求“到任意起点的最短距离”，只需把所有起点初始全部入队！

TikZ 图解：多源 BFS（超级源点直觉与同步波纹）

把“多个火源 S_1, S_2 同时入队”想象成天上有一个虚构的“超级源点”，同时点亮所有火源：

$S_1(0)$	1	2	2
1	2	2	1
2	3	1	$S_2(0)$

初始队列：[S1，S2]，都是 $d = 0$

第 1 轮两波同时扩出所有 $d = 1$

中间交界处谁先踩到就算谁，绝不重入

本页要点

因为两个火源同时从 0 出发，谁先碰到格子 (x, y)，谁离它就最近，后面的火源不必再更新！

多源 BFS 手算跟踪表（2×3 网格两火源同步扩）

假设 2×3 网格火源在 $(0,0)$ 和 $(1,2)$，手算同步扩展全过程（含队列为空终止步）：

步序	弹出队头	队列状态（操作后）	dist 更新与说明
初始	-	[(0,0), (1,2)]	两起点同时入队，dist=0
1	$(0,0)$	[(1,2), (1,0), (0,1)]	扩展 $(1,0),(0,1)$ 标 dist=1
2	$(1,2)$	[(1,0), (0,1), (0,2), (1,1)]	扩展 $(0,2),(1,1)$ 标 dist=1
3∼6	依次弹出 $d=1$	[]	发现所有格子均已被标过 dist，全部跳过
7	条件为假	[]	判断 !q.empty() 为假，循环终止

本页要点

整个 2×3 网格每个格子只进队 1 次，总复杂度依然严格 $O(R \cdot C)$！

多源 BFS 模板代码与反事实追问

```
vector<vector<int>> dist(R, vector<int>(C, -1));
queue<pair<int, int>> q;
for (int i = 0; i < R; ++i)
    for (int j = 0; j < C; ++j)
        if (g[i][j] == 'S') { dist[i][j] = 0; q.push({i, j}); } // 所有起点一起入
    队!
while (!q.empty()) {
    auto [x, y] = q.front(); q.pop();
    for (int k = 0; k < 4; ++k) {
        int nx = x + dx[k], ny = y + dy[k];
        if (nx<0 || nx>=R || ny<0 || ny>=C || dist[nx][ny]!=-1) continue;
        dist[nx][ny] = dist[x][y] + 1; q.push({nx, ny});
    }
}
```

- 反事实追问：如果某题有 2 个火源，却写成“先只把 S_1 入队跑完 BFS，再把 S_2 入队但不重置 dist 再跑一次”，会怎样？答：第一次已经把全图 dist 填满了，第二次 dist[nx][ny] != -1 全跳过，结果漏掉 S_2！

生活引入（0-1 BFS）：有些路免费（0 步），有些路走 1 步

小例子：迷宫里有“顺风传送门”（走过去花 0 分钟）和普通走廊（走过去花 1 分钟）

起点 1 到 3 有一条费用为 1 的直接走廊；但 $1 \to 2$ 花 1 分钟，$2 \to 3$ 是传送门花 0 分钟……不对，如果 $1 \to 2$ 花 0 分钟，$2 \to 3$ 也花 0 分钟，而 $1 \to 3$ 花 1 分钟：

- 如果还用普通 queue 并且首次入队就写死：从 1 先看到 3（花 1 分钟）和 2（花 0 分钟），3 被排在 2 前面先标成了 1，等 2 出来发现 $2 \to 3$ 只要 $0+0=0$ 分钟时却因为“已标过”被跳过——直接 **WA**！
- 而且普通 queue 只能塞队尾，走 0 分钟的点被迫排在走 1 分钟的点后面，打破了“距离小的先出队”的秩序！

本页要点

当边权只有 0 和 1 两种时，普通队列会乱序；我们需要能两头插队的“双端队列”！

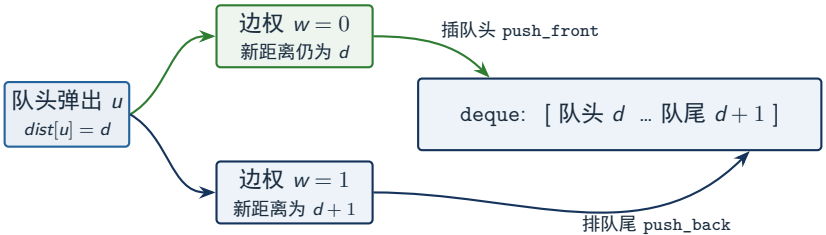
新术语大白话：双端队列 deque 与“0 塞队头、1 塞队尾”

- 双端队列（deque）：既可以从队尾排进去（push_back），也可以直接插到队头最前面（push_front）的升级版队列。
- 为什么 **0-1 BFS** 用 deque 就能保持单调？
  1. 每次从队头弹出当前距离最小的点 u（设它的距离是 d）。
  2. 如果通过权值为 0 的边到达 v，新距离还是 d，跟刚弹出的 u 同一层，把它插到队头 push_front，保证和所有距离为 d 的点排在最前面处理完！
  3. 如果通过权值为 1 的边到达 v，新距离变成 $d+1$，乖乖放到队尾 push_back，留给下一层！
- 松弛更新：一个点可能先被 1 边塞进队尾，后来又被 0 边发现更近的路，所以必须写 if (dist[u] + w < dist[v]) 允许刷新！

本页要点

口诀：“**0** 放队头立刻算，**1** 放队尾排队等；判重不能只判 **-1**，要判距离更小再更新！”

TikZ 图解：0-1 BFS 双端队列插入规则



本页要点

整个过程不需要 $O(n \log n)$ 的优先队列（堆），每个点最多被松弛进队几次，复杂度线性 $O(V+E)$！

手算跟踪表：0-1 BFS 必须“松弛更新”的 4 点小图

有向边：$1 \xrightarrow{1} 3$，$1 \xrightarrow{0} 2$，$2 \xrightarrow{0} 3$。从起点 1 出发跟踪 deque 与 dist[1..3]（初值 INF）：

步序	弹出队头	deque 状态	dist[1,2,3]	松弛更新动作
初始	-	[1]	[0, INF, INF]	起点入队
1	$1(d=0)$	[2, 3]	[0, 0, 1]	$1 \to 3$ 权 1 放队尾；$1 \to 2$ 权 0 插队头！
2	$2(d=0)$	[3, 3]	[0, 0, 0]	发现 $0+0<1$！把 3 刷新为 0 再插队头
3	$3(d=0)$	[3]	[0, 0, 0]	弹出最优的 3（距离 0），无出边
4	3(旧)	[]	[0, 0, 0]	旧的距离 1 的 3 弹出（可判 vis 跳过）
5	条件为假	[]	[0, 0, 0]	判断 !dq.empty() 为假，循环终止

本页要点

第 2 步清清楚楚展示了为什么 3 虽然已经在队列里，但遇到更小的 0 边时依然必须再更新并插到队头！

模板代码：0-1 BFS（双端队列标准写法）

```
deque<int> dq;
vector<int> dist(n + 1, 1e9);                    // 初值设为无穷大，方便做 min 比较
dist[s] = 0; dq.push_back(s);
while (!dq.empty()) {
    int u = dq.front(); dq.pop_front();
    for (auto [v, w] : g[u]) {                   // w 只能是 0 或 1
        if (dist[u] + w < dist[v]) {             // 核心：距离变小才松弛更新！
            dist[v] = dist[u] + w;
            if (w == 0) dq.push_front(v);        // 0 权边不加距离，插到队头
            else        dq.push_back(v);         // 1 权边距离 +1，放到队尾
        }
    }
}
```

要点：可额外在出队时加 `if (vis[u]) continue; vis[u]=1;`，保证每个点最多只向外扩展一次！

为什么对？0-1 BFS 双端队列单调性论证

阶段	要验证的不变量
初始化	只有起点 s 在队头，`dist[s]=0`，队列满足单调非降且首尾差不超过 1。
保持	若当前弹出队头元素 u 时，队内元素距离只有 d 和 $d+1$ 且所有 d 都在 $d+1$ 前面：从 u 走 $w=0$ 边得新距离 d 并插到队头，仍在最前；走 $w=1$ 边得新距离 $d+1$ 并塞到队尾，仍在最后。因此单调性永远保持！
终止	当配合出队判重 `vis[u]` 时，每个点第一次从队头弹出一定是它的真实最短距离。

本页要点

正因为队列中任何时刻最多只存在 d 和 $d+1$ 两种距离，才无需排序就能直接线性求解！

上机验证与反事实追问：写死 dist[v] != -1 会怎样？

测试下面 3 组图输入（边表分别为上述 3 边小图、全 0 边链、无路可达）：

- 普通图：边 $1 \xrightarrow{1} 3$，$1 \xrightarrow{0} 2$，$2 \xrightarrow{0} 3 \rightarrow$ `dist[3] = 0`。
- 最小边界：$n = 1, m = 0, s = 1 \rightarrow$ `dist[1] = 0`。
- 不可达：$n = 3, m = 1$（只有 $1 \xrightarrow{1} 2$）$\rightarrow$ `dist[3]` 保持极大值 `1e9`。

反事实追问

1. 如果把第 7 行的 `if (dist[u] + w < dist[v])` 改写成普通 BFS 的 `if (dist[v] == 1e9)`，上面的 3 点小图输出什么？
   答：因为 1 先遍历出边 $1 \xrightarrow{1} 3$ 把 `dist[3]` 标成了 1，轮到 $2 \xrightarrow{0} 3$ 时发现 `dist[3] != 1e9` 直接跳过，错误输出 **1**（WA）！
2. 如果某条边权变成了 2，还能继续用 `push_front / push_back` 吗？
   答：不能！因为加 2 的点放到队尾会和加 1 的点混在一起破坏单调性，此时应改用图论篇的 Dijkstra 算法！

易错对照：多源与 0-1 BFS 常见误区

问题场景	常见错误写法	正确做法与原因
0-1 BFS 判重写死（**WA**）	写成 `if (dist[v] != -1) continue;`	0 边可以刷新 1 边入队的点，必须写 `dist[u] + w < dist[v]`
多源只入队一个（**WA**）	找到第一个起点就 `break` 去跑 BFS	必须扫描整张图把所有起点全入队后再 `while`
0-1 BFS 混入权 2（**WA**）	边权有 $\{0,1,2\}$ 仍用 `deque`	边权超过两种值请用优先队列 Dijkstra
多源输出错（**WA**）	忘记把自身起点设 `dist = 0`	所有源点出队前 `dist` 必须显式置 0

本页要点

普通 BFS 只要 `-1` 判重；0-1 BFS 必须改用 `dist[u] + w < dist[v]` 松弛！

为什么单向 BFS 也会爆？状态空间指数级爆炸

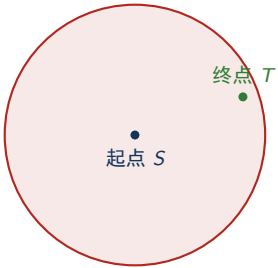
很多状态转换题（如 4 位密码锁每次拨动有 8 种操作、八数码每次移动有 3 种选择）步数稍微多一点，单向 BFS 就会撑爆内存或超时：

- 假设每一步平均分叉 $b = 10$ 个新状态，从起点到终点需要走 $d = 8$ 步：
  - 单向 **BFS**：第 8 层需要扩展约 10^8 个状态，耗时且内存轻松爆表（**TLE / MLE**）！
  - 双向 **BFS**：从起点和终点同时出发，各自只走 $d/2 = 4$ 步在中间碰头，两边合计只需要扩展约 $2 \times 10^4 = 20{,}000$ 个状态，直接快出 5000 倍！
- 适用前提：必须同时知道明确起点和明确终点，而且操作可以倒着推回去。

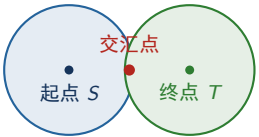
本页要点

当每步分叉很大、且起点和终点都唯一确定时，双向 BFS 把指数底数上的步数直接折半！

TikZ 图解：单向大圆 vs 双向两小圆碰头



单向 BFS：面积 $\propto b^d$（巨大）



双向 BFS：面积 $\propto 2 \cdot b^{d/2}$（极小）

本页要点

半径减半，搜索面积呈几何级数缩小；一旦两侧状态在地图中间重叠，两边步数相加就是答案！

双向 BFS 两大核心规则：按层扩 + 每次挑小的扩

写双向 BFS 不能胡乱从头尾各弹出一个点，必须守住两个规矩：

1. 每次挑队列元素更少的那一侧扩展：如果 `qa.size() <= qb.size()` 就扩 `qa`，否则扩 `qb`。这样两边扩展面尽量均衡，不会出现一头已经膨胀到 10^5 个点还在硬扩的情况！
2. 每次必须扩完当前“一整层”：记住当前队列大小 `int sz = q.size();`，用 `while (sz--)` 把这一层的点全部弹完。否则同一层先碰到的点可能不是最短交汇点！
3. 交汇判定：当一侧扩展出新状态 v，查发现另一侧距离表 `db` 里已经存过 v，答案就是 `da[u] + 1 + db[v]`！

本页要点

只扩较小一侧 + 按层整批推进 + 撞见对方标记立刻返回两边距离和！

手算跟踪表：数轴小例双向 BFS 碰头过程

起点 $S=1$，终点 $T=10$，每步可从 x 走到 $x+1$ 或 $2x$（逆向等价于 $x-1$ 或 $x/2$ 当 x 为偶）。跟踪交汇全过程：

步序	本轮扩哪侧	qa（正向 S）	qb（反向 T）	交汇检查与距离
初始	-	[1](d=0)	[10](d=0)	两队列均非空，大小均为 1
1	扩 qa（同大）	[2](d=1)	[10](d=0)	$1+1=2$，$2\times1=2$（去重），未遇 qb
2	扩 qb（较小）	[2](d=1)	[9，5](d=1)	$10-1=9$，$10/2=5$，未遇 qa
3	扩 qa（较小）	[3，4](d=2)	[9，5](d=1)	$2+1=3$，$2\times2=4$，未遇 qb
4	扩 qa（同大）	扩出 5！	[9，5](d=1)	$4+1=5$ 在 qb 中（$db[5]=1$）！命中！总步数 $2+1+1=4$？等一下！

本页要点

验算：$1\to2\to4\to5\to10$ 共 4 步（第 4 步由 da[4]=2 加 1 加上 db[5]=1 刚好等于 4）！若某侧队列空仍未碰头则无解。

模板代码：双向 BFS 核心主循环骨架

```
int bfs_bi(int S, int T) {
    if (S == T) return 0;                          // 特判起点等于终点
    queue<int> qa, qb; unordered_map<int, int> da, db;
    qa.push(S); da[S] = 0; qb.push(T); db[T] = 0;
    while (!qa.empty() && !qb.empty()) {           // 任一侧空了说明绝对不连通
        int res = (qa.size() <= qb.size())
            ? extend(qa, da, db)                   // 总是扩展状态更少的那一侧
            : extend(qb, db, da);
        if (res != -1) return res;                 // 一旦两侧碰头，直接返回总步数
    }
    return -1;
}
```

本页要点

用 `extend(q, d_self, d_other)` 按层循环 `sz = q.size()` 推进一层，碰到 `d_other.count(v)` 即返回 `d_self[u] + 1 + d_other[v]`。

双向 BFS 易错与反事实追问

- 反事实追问 **1**：如果把 `while (!qa.empty() && !qb.empty())` 误写成 `||`（或），当起点被一圈墙彻底围死时，会发生什么？
  答：qa 空了以后，qb 会继续把剩下整张大图全部搜完直到也变空，白白浪费大量时间！任一侧队列空了，两边就绝不可能相遇，必须用 `&&`！
- 反事实追问 **2**：如果题目给了一个起点，但要求找“任意一个出口 `E`（地图里有多个出口）”的最短距离，适合用双向 BFS 吗？
  答：多个出口更适合从所有出口一起出发跑多源 **BFS**，双向 BFS 主要用于“单起点对单终点”的庞大状态空间（如八数码、字符串变换）。

本页要点

记住两点：循环条件用 `&&`；每次必须用 `sz = q.size()` 扩完一整层再换边！

生活引入（A* 算法）：迷宫里先往终点方向走，别乱往反方向扩

类比开车看导航：你要从武汉去北京，会先往南边的广州开吗？

普通 BFS 像没装导航的瞎子，只要是 1 步能到的路口（不管往北还是往南）全按相同优先级扩；而 **A***（读作 **A-Star**）给每个点多算一个“大概离终点还有多远”的估计值，优先把朝终点方向走、估计总路程最短的点放进优先队列最前面！

- 什么时候用 **A***？普通网格/图有明确终点、但空间极大时（或求第 k 短路），靠估价指引方向大幅减少无效扩展！
- 如果把估价直接设为 0：A* 就退化成了普通的 BFS / Dijkstra！

本页要点

A* = 普通最短路（已走步数）+ 方向指南针（离终点还有多远的估计）！

新术语大白话：$g(x)$、估价函数 $h(x)$、$f(x)$ 与可采纳性

- $g(x)$（已经花的代价）：从起点走到当前状态 x 实打实已经花掉的步数（确定的）。
- 估价函数 $h(x)$（猜还要花多少）：从 x 走到终点估计还需要多少步（比如网格里假设没墙的直线距离）。
- 总估价 $f(x) = g(x) + h(x)$：优先队列（小根堆）按 $f(x)$ 从小到大排序，每次弹出 f 最小的！
- 可采纳性（**Admissible**）：$h(x) \le h^*(x)$（$h^*(x)$ 是从 x 到终点的真实最短距离）。大白话：你猜的剩余距离，永远不能比实际最省的距离还大（宁可乐观少猜，绝不悲观高估）！

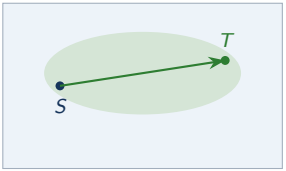
本页要点

网格四方向移动时，曼哈顿距离 $h = |x - tx| + |y - ty|$ 永远 $\le$ 绕墙后的真实步数，天然满足 $h \le h^*$！

TikZ 图解：普通 BFS 盲目铺开 vs A* 瞄准终点



普通 BFS：四面八方平均扩散



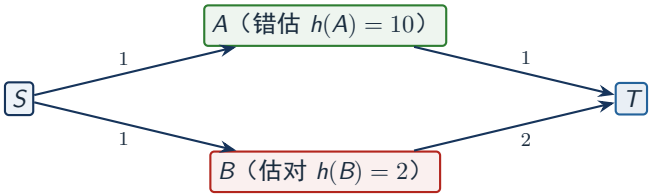
A*（$f = g + h$）：集中向 T 推进

本页要点

当周围有大量往反方向延伸的空地时，A* 靠 $h(x)$ 把反方向的 f 拉大，几乎不去碰它们！

为什么必须 $h \le h^*$？反例图解高估会错失最优解！

假设从 S 到 T 有两条路：上面一条 $S \xrightarrow{1} A \xrightarrow{1} T$（真实总长 2）；下面一条 $S \xrightarrow{1} B \xrightarrow{2} T$（真实总长 3）。



- 因为把 $h(A)$ 猜高成了 $10 > h^*(A) = 1$，导致 $f(A) = 1 + 10 = 11$，而 $f(B) = 1 + 2 = 3$。
- 小根堆先弹出 B 并一路把 T 以总距离 3 弹出来直接结束，真正的最短路 2 还没出堆就被抛弃了（**WA**）！
- 反过来只要所有点都满足 $h(x) \le h^*(x)$，当 T 出堆时，堆里所有还没扩展完的路估价都 $\ge f$，真实长度也一定 $\ge$ 当前 T 的距离，因此首次弹出终点必是最优！

模板代码：A* 优先队列核心骨架

```
struct Node {
    int f, g, u;                                    // f = g + h(u)
    bool operator>(const Node& o) const { return f > o.f; } // 小根堆：f 小的先出
};
priority_queue<Node, vector<Node>, greater<Node>> pq;
dist[S] = 0; pq.push({h(S), 0, S});
while (!pq.empty()) {
    auto [f, g, u] = pq.top(); pq.pop();
    if (u == T) { cout << g << '\n'; break; }   // h <= h* 时，首次出堆必最优！
    if (g > dist[u]) continue;                  // 跳过已经被更小 g 刷新的旧状态
    for (auto [v, w] : adj[u])
        if (g + w < dist[v]) { dist[v] = g + w; pq.push({dist[v] + h(v), dist[v], v
    }); }
}
```

要点：与 Dijkstra 相比只多一个 + h(v)；网格四方向常用曼哈顿距离作 h(v)。

五大 BFS/最短路搜法选型速查与易错对照

方法	边权 / 场景	数据结构	判重与常见坑
普通 **BFS**	全为 1，单起点	`queue`	`push` 时立即标 `dist`；防 `pop` 后标爆内存
多源 **BFS**	全为 1，多起点	`queue`	初始所有起点全部入队设 0，不要分多次跑
0-1 BFS	边权仅 $\{0,1\}$	`deque`	必须写 `dist[u]+w < dist[v]`，不能用 `-1` 写死
双向 **BFS**	已知唯一终起点	双 `queue`	每次扩小队列且整层推，循环用 `&&`
A* 搜索	状态多、有坐标估价	`priority_queue`	必须保证 $h(x) \le h^*(x)$，高估会直接 **WA**

本页要点

先看边权（全 1 / 0-1），再看起终点个数和状态大小，10 秒钟就能挑中对的搜索框架！

例 1 | 走迷宫（一本通 1252）[入门]：先读懂要求

大白话题意

给你一个 R 行 C 列的迷宫字符网格（$1 \le R, C \le 40$），其中 `'.'` 是可以走的空地，`'#'` 是障碍物。只能上下左右四方向走，不能斜着走。求从左上角 $(0,0)$ 走到右下角 $(R-1, C-1)$ 最少需要经过多少个空地格子（注意：计算时包括起点和终点）！

- 输入：第 1 行 R, C；下面 R 行每行 C 个字符（保证左上角和右下角都是 `'.'` 且一定能走到）。
- 处理：用普通网格 BFS 求出最少移动步数，并把起点初始值计为 1 格。
- 输出：一个整数（样例 5×5 迷宫输出 9）。

本页要点

看清题目最后一句话：“计算时要包括起点和终点”——这意味着起点 `dist[0][0]` 必须初始化为 1 而非 0！

例 1 | 怎么想到的？样例网格与水波推进建模

把原题样例 5×5 迷宫画出来，从左上角 $(0,0)$ 开始按层水波扩散（数字代表到达时经过的格子数）：

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

绿色格子即最短路径：共 9 格！

注意中间 $(1,4)$ 也能到 $(d=6)$，

但往下走撞到墙 $(2,4)$ 进不去。

本页要点

每步权值相等都是 $+1$ 格，标准二维网格 BFS 一次水波刚好在 9 步命中 $(4,4)$！

例 1 | 样例手推表（追踪样例前 6 层推进）

步序	弹出点	新入队格	写入的 dist 值与说明
初始	-	$(0,0)$	`dist[0][0] = 1`（算上起点本身）
1	$(0,0)$	$(0,1)$	下方 $(1,0)$ 是 `'#'` 跳过，右方 $(0,1)$ 标 2
2	$(0,1)$	$(1,1)$	下方 $(1,1)$ 标 3
3	$(1,1)$	$(2,1),(1,2)$	向下标 4，向右标 4（进入分叉）
4∼7	沿路推进	推到 $(3,3)$	分别经过 $(1,3)\to 5$、$(2,3)\to 6$、$(3,3)\to 7$；另支 $(2,1)\to(3,1)\to(4,1)$ 被墙围死
8∼9	$(3,3)\to(4,3)$	$(4,4)$	$(4,3)$ 标 8，接着扩展 $(4,4)$ 标 9，出队即 `break`！

本页要点

其中左下死胡同 $(2,1)\to(3,1)\to(4,1)$ 走完自动停止，完全不会干扰右边正路！

例 1 | 核心步骤与相对模板改动

按顺序计算	纸上得到	核对问题
设 $(0,0)$ 初值为 1 并入队	`dist[0][0] = 1`	题目算格子总数还是移动步数？
弹出检查是否终点	`x==R-1 && y==C-1`	坐标从 0 算起，右下角是 $(R-1, C-1)$ 吗？
四方向扩展合法空格	`dist[nx][ny]=dist[x][y]+1`	是否在 `q.push` 前立刻标了 `dist`？

相对标准网格 BFS 模板改了哪里？

1. 起点和终点坐标直接固定为 $(0,0)$ 与 $(R-1, C-1)$，不需要在地图里扫描 `'S'` 和 `'E'`。
2. 起点初始距离从 0 改为 1（因为含起终点格子总数 $=$ 移动步数 $+1$）。

例 1 | 完整可编译代码（1/3）：读入地图

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

// 一本通 1252 走迷宫：二维网格 BFS 求最少经过格子数（入队即标记 dist 防重入队列！）
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int R, C;
    if (!(cin >> R >> C)) return 0;
    vector<string> g(R);
    for (auto &row : g) cin >> row;
```

要点：与 code/maze_shortest.cpp 一致；用 vector<string> 按行读字符免去开大数组烦恼。

例 1 | 完整可编译代码（2/3）：队列初始化与终点退出

接上一页 code/maze_shortest.cpp:

```
const int dx[4] = {1, -1, 0, 0}, dy[4] = {0, 0, 1, -1};
vector<vector<int>> dist(R, vector<int>(C, -1));
queue<pair<int, int>> q;
dist[0][0] = 1;                                    // 本题要求含起点与终点格子总
数，初值 1
q.push({0, 0});

while (!q.empty()) {
    auto [x, y] = q.front(); q.pop();
    if (x == R - 1 && y == C - 1) break;           // 首次出队即保证是最短路！
```

本页要点

第 4 行 dist[0][0] = 1; 是本题唯一和“算移动步数”不同的地方，做题时务必再看一遍题面最后一句！

例 1 | 完整可编译代码（3/3）：四方向扩展与输出

接上一页 code/maze_shortest.cpp 结尾：

```
        for (int k = 0; k < 4; ++k) {
            int nx = x + dx[k], ny = y + dy[k];
            if (nx < 0 || nx >= R || ny < 0 || ny >= C) continue;
            if (g[nx][ny] == '#' || dist[nx][ny] != -1) continue;
            dist[nx][ny] = dist[x][y] + 1;          // 必须在 push 时立刻打标！
            q.push({nx, ny});
        }
    }
    cout << dist[R - 1][C - 1] << '\n';
    return 0;
}
```

本页要点

先判 nx，ny 越界，再读 g[nx][ny] 和 dist[nx][ny]，最后先标 dist 再 q.push！

例 1 | 易错点与 3 组自测边界

- 易错 **1**（**WA** 差 **1**）：照背模板写了 `dist[0][0] = 0`，样例输出 8 而不是 9。
- 易错 **2**（**RE** 越界）：把 `if (g[nx][ny] == '#')` 写在了 `nx >= R` 判断前面。
- 易错 **3**（**TLE/MLE**）：忘了在 `push` 前给 `dist[nx][ny]` 赋值，改成出队才赋值。

3 组必测边界数据

1. 最小 1×1 网格：`1 1 / .` → 刚弹出 $(0,0)$ 就 `break`，输出 `1`。
2. 2×2 无障碍：`2 2 / .. / ..` → 输出 `3`（$(0,0) \to (1,0) \to (1,1)$ 共 3 格）。
3. 原题样例：`5 5` 迷宫 → 输出 `9`。

例 2 | 抓住那头牛（一本通 1253）[基础]：先读懂要求

大白话题意

农夫和牛都在数轴上，农夫在坐标 N，牛在坐标 K（$0 \le N, K \le 100{,}000$），牛不动。农夫每分钟可以从位置 X 走到 $X-1$、$X+1$，或者瞬间传送到 $2X$。求农夫抓到牛最少需要多少分钟？

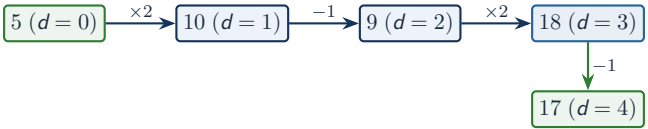
- 输入：一行两个整数 N 和 K（如 `5 17`）。
- 处理：把数轴上的每个整数坐标看成一个节点，每次向 $x-1,\ x+1,\ 2x$ 三条边扩展跑一维 BFS。
- 输出：最小分钟数（样例 `5 17` $\to$ `4`，路线为 $5 \to 10 \to 9 \to 18 \to 17$）。

本页要点

看一眼样例路线：$5 \to 10 \to 9 \to 18 \to 17$！注意 18 已经超过了终点 17，然后再向左退回 1 步！

例 2 | 怎么想到的？一维状态图建模与 2×10^5 上界推导

不是只有迷宫网格才能用 BFS！任何“状态每次能变出几个新状态、每步代价为 1”的题都能用 BFS：



- 关键问题：状态数组 `dist` 开到 10^5 够吗？绝对不够！
- 万一 $K = 99{,}999$，农夫完全可能先乘 2 冲到 100,000 以上（甚至接近 2×10^5）再 -1 退回来！但绝对不需要冲超过 200,000，因为如果超过 $2K$，直接先 -1 再 $\times 2$ 一定比先 $\times 2$ 再退 -1 更省步数！

本页要点

上界安全区：`LIM = 200000`（即 2×10^5），下界 0（不会走到负数再加回来）！

例 2 | 样例手推表（5 17 队列扩张过程）

步序	弹出 x	当前 dist[x]	入队的新坐标（跳过已标记或越界）
初始	-	-	dist[5]=0，队列 [5]
1	5	0	入队 $4, 6, 10$（均标 dist=1）
2∼4	$4, 6, 10$	1	4 扩 $3, 8$；6 扩 $7, 12$；10 扩 $9, 11, 20$（标 dist=2）
5∼11	$3, 8, \ldots, 9$	2	轮到 9 时扩出 18（标 dist=3）；11 扩 22；20 扩 $19, 21, 40$
12	18 附近轮	3	18 弹出后扩出 $18 - 1 = 17$，标 dist[17] = 4
13	17	4	出队命中 x == K (17)，直接 break 输出 4！

本页要点

虽然第 12 步就标出了 dist[17]=4，等到第 13 步出队检查 if (x == K) break; 时安全退出！

例 2 | 核心步骤与相对模板改动

按顺序计算	纸上得到	核对问题
把数轴范围设为 $[0, 200000]$	`LIM = 200000`	开到 10^5 为什么会漏掉超车折返？
`dist[N]=0` 入队，遇 `K` 停	循环枚举 `{x-1, x+1, 2*x}`	是算分钟数（初值 0）吗？
先判 `y<0 || y>LIM`	再判 `dist[y] != -1`	如果 $N = K$，不进扩展循环输出几？

相对二维网格 BFS 模板改了哪里？

1. 状态从二维坐标 `pair<int, int>` 简化为一维整数 `int x`，队列也改为 `queue<int>`。
2. 四方向 `dx/dy` 循环替换为范围 `for (int y : {x - 1, x + 1, 2 * x})` 三种一维跳跃。

例 2 | 完整可编译代码（1/3）：读入与状态数组初始化

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

// 一本通 1253 抓住那头牛：一维状态空间 BFS，支持 x-1, x+1, 2*x 三种转移
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int N, K;
    if (!(cin >> N >> K)) return 0;
    const int LIM = 200000;                          // 上界开到 2*max(N,K) 防向右越
    过后再左跳
    vector<int> dist(LIM + 1, -1);
    queue<int> q;
```

要点：与 code/catch_cow.cpp 一致；开 LIM + 1 涵盖 $0 \sim 200000$ 全部坐标。

例 2 | 完整可编译代码（2/3）：起点入队与主循环

```
    dist[N] = 0;
    q.push(N);
    while (!q.empty()) {
        int x = q.front(); q.pop();
        if (x == K) break;
```

本页要点

遇到 `x == K` 立即 `break`；当起点就是终点 $N == K$ 时，第 1 轮出队即停并输出 0！

例 2 | 完整可编译代码（3/3）：一维三分支转移与输出

```
        for (int y : {x - 1, x + 1, 2 * x}) {
            if (y < 0 || y > LIM || dist[y] != -1) continue;
            dist[y] = dist[x] + 1;
            q.push(y);
        }
    }
    cout << dist[K] << '\n';
    return 0;
}
```

本页要点

C++17 支持直接写 `for (int y : {x - 1, x + 1, 2 * x})`，简洁不易漏分支！

例 2 | 易错点与 3 组自测边界

- 易错 **1**（**WA/RE** 上界不够）：LIM 误写 100000，当输入 `50001 99999` 等需要超车折返的数据时，直接漏掉最优解！
- 易错 **2**（**RE** 负下标）：当 $x = 0$ 时，$x - 1 = -1$，如果不先判 `y < 0` 直接查 `dist[-1]` 就会越界崩溃！
- 易错 **3**（**TLE** 没判重）：忘了写 `dist[y] != -1`，数轴上反复横跳瞬间爆队列。

3 组必测边界数据

1. 起点即终点：`10 10` $\to$ `0`（刚弹出 10 就 `break`）。
2. 人在牛右边：`20 3` $\to$ `17`（加 `1` 和翻倍只会更远，只能一直 -1）。
3. 原题样例：`5 17` $\to$ `4`。

例 3 | 细胞（一本通 1329）[基础]：先读懂要求

大白话题意

给你一个 n 行 m 列的数字矩阵（$1 \le n, m \le 100$），由字符 `'0'` 到 `'9'` 组成。其中 `'0'` 表示背景（非细胞），`'1'` 到 `'9'` 都表示细胞。只要两个非零数字在上下左右四个方向相邻，它们就属于同一个细胞（不要求数字相等）。求矩阵里总共有多少个细胞？

- 输入：第 1 行 n, m；接下来 n 行每行 m 个连续数字字符（中间无空格）。
- 处理：用“外层扫全图 + 内层 BFS 洪水填充（Flood Fill）”统计四连通块个数。
- 输出：细胞个数（样例 4×10 矩阵输出 4）。

本页要点

相邻只算上下左右四连通，斜向挨着不算同一个细胞；而且读入的是连成一行的字符！

例 3 | 怎么想到的？Flood Fill 洪水填充建模

把原题 4×10 样例矩阵画成格子图，用不同颜色标出连在一起的非零数字：

```
0234500067
1034560500
2045600671
0000000089
```

1. 从上到下、从左到右逐格扫描：
2. 只要撞见一个没访问过的非 `'0'` 格，
   计数 `++cells`，并启动一次 BFS，
   像“放水淹掉整块地”一样把它连通的
   全部标记 `vis = 1`，以后绝不重复数！

本页要点

这种“遇新起点就计数 $+1$ 并放水标记整片连通块”的套路，叫做洪水填充（**Flood Fill**）！

例 3 | 样例手推表（扫描 4×10 矩阵发现 4 个细胞）

扫描位置	字符与 vis	cells	本步动作与 BFS 洪水淹没范围
$(0,0)$	'0', 0	0	非细胞，跳过
$(0,1)$	'2', 0	$0 \to \mathbf{1}$	细胞 **1**：BFS 淹掉左上蓝色 11 格设 vis=1
$(0,2)..(0,7)$	'3' 或'0'	1	要么是 '0'，要么 vis=1，跳过！
$(0,8)$	'6', 0	$1 \to \mathbf{2}$	细胞 **2**：BFS 淹掉右上角 (0,8),(0,9) 2 格
$(1,0)$	'1', 0	$2 \to \mathbf{3}$	细胞 **3**：BFS 淹掉左下角 (1,0),(2,0) 2 格
$(1,7)$	'5', 0	$3 \to \mathbf{4}$	细胞 **4**：BFS 淹掉右下角 6 格；后续全跳过，输出 4

本页要点

虽然外层循环套了一个 BFS，但每个格子最多只被放水淹没 1 次，总复杂度依然是 $O(nm)$！

例 3 | 核心步骤与相对模板改动

按顺序计算	纸上得到	核对问题
双重循环扫每格 (i, j)	`g[i][j] != '0' && !vis`	判断的是字符 `'0'` 还是数字 0？
发现新块计数器 $+1$ 并设 `vis=1`	`++cells; vis[i][j]=1;`	起点入队前立刻设了 `vis=1` 吗？
内部 `while(!q.empty())` 扩四邻	连通全部非零格标 `vis=1`	是否只用了上下左右 4 个方向？

相对标准最短路 BFS 模板改了哪里？

1. 这道题不问步数，所以把 `dist` 数组简化为布尔/字符标记 `vis`，遇到目标也不需要 `break`。
2. 外面多套了一层 $n \times m$ 的双重循环，对每一个未访问过的连通块各自启动一次 BFS。

例 3 | 完整可编译代码（1/4）：读入矩阵与初始化

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

// 一本通 1329 细胞：BFS 洪水填充（Flood Fill）统计四连通块个数
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    if (!(cin >> n >> m)) return 0;
    vector<string> g(n);
    for (auto &row : g) cin >> row;
```

要点：与 code/cells.cpp 一致；用 vector<string> 读紧凑数字串，切勿当整数读入。

例 3 | 完整可编译代码（2/4）：外层扫描触发新连通块

```
vector<vector<char>> vis(n, vector<char>(m, 0));
const int dx[4] = {1, -1, 0, 0}, dy[4] = {0, 0, 1, -1};
int cells = 0;

for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
        if (g[i][j] == '0' || vis[i][j]) continue;
        ++cells;                                    // 发现一个新连通块起点
        queue<pair<int, int>> q;
        vis[i][j] = 1; q.push({i, j});
```

要点：用 `vector<char>` 存标记避开 `vector<bool>` 位压缩陷阱；起点入队即标 1。

例 3 | 完整可编译代码（3/4）：BFS 四方向扩展淹没

```
        while (!q.empty()) {
            auto [x, y] = q.front(); q.pop();
            for (int k = 0; k < 4; ++k) {
                int nx = x + dx[k], ny = y + dy[k];
                if (nx >= 0 && nx < n && ny >= 0 && ny < m && !vis[nx][ny] && g
[nx][ny] != '0') {
                    vis[nx][ny] = 1;
                    q.push({nx, ny});
                }
            }
        }
```

本页要点

入队即标 `vis[nx][ny] = 1;`，防止同一个细胞格子被周围邻居反复推进队列！

例 3 | 完整可编译代码（4/4）：闭合循环与输出总数

```
            }
        }
        cout << cells << '\n';
        return 0;
}
```

本页要点

外层扫描结束后，`cells` 恰好就是四连通非零区域的总块数！

例 3 | 易错点与 3 组自测边界

- 易错 **1**（**WA** 字符与整数混淆）：写成 `g[i][j] == 0` 而不是 `'0'`！字符 `'0'` 的 ASCII 码是 48，写成 0 会把所有 `'0'` 都当成细胞！
- 易错 **2**（**WA** 错用八连通）：多加了 4 个斜方向，把斜角挨着的两个细胞合并成了一个。
- 易错 **3**（**WA** 忘记外层判重）：只判 `g[i][j] != '0'` 而漏判 `vis[i][j]`，同一个细胞会被算很多次。

3 组必测边界数据

1. 全 **0** 矩阵：`2 2 / 00 / 00` $\rightarrow$ `0`（一个细胞都没有）。
2. 最小单细胞：`1 1 / 5` $\rightarrow$ `1`。
3. 原题样例：`4 10` 矩阵 $\rightarrow$ `4`。

BFS 最短路高频错误对照表

题目 / 场景	常见错误写法	正确做法与原因
出队时才打标记	把 `vis[nx][ny]=1` 放到 `pop()` 后	同一点会被多个邻居重复入队，队列爆内存 **MLE/TLE**
数轴跳步数组小	《抓住那头牛》`dist` 只开 10^5	跳 $2x$ 跨过 K 再左退很常见，上界开 200000
起点距离初值错	题目问“经过格子数”却写了 `dist=0`	分清“经过格子数”（初值 1）还是“移动步数”（初值 0）
字符数字弄混	《细胞》里判 `g[i][j] == 0`	字符数组比较必须带单引号 `'0'`
0-1 BFS 判重写死	首次入队就写死不再松弛	权 1 先入队的点会被权 0 边刷新，应判 `dist[u]+w < dist[v]`

本页要点

“入队即标记，判界才进队”；每次写网格题前先确认初值是 0 还是 1！

常见错误：看懂评测结果再修改

提交代码看到 OJ 返回结果时，按下面顺序快速定位：

- 编译错误（**CE**）：检查 `queue`、`deque` 头文件是否包含，C++17 结构化绑定 `auto [x, y]` 是否开了 `-std=c++17`。
- 答案错误（**WA**）：检查起点初值是 0 还是 1、字符 `'0'` 有没有写成 0、0-1 BFS 有没有用 `dist[u]+w < dist[v]` 松弛、多组数据有没有清空队列或 `dist`。
- 格式错误（**PE**）：检查像《Dungeon Master》这类题目的句末句号、空格和感叹号拼写。
- 运行错误（**RE**）：检查是不是先访问 `g[nx][ny]` 后才判越界、或者《抓住那头牛》中 $2x$ 超过数组上限、$x-1$ 变成负数。
- 内存超限 / 超时（**MLE/TLE**）：99% 是忘了在 `q.push` 前立刻标记 `dist` 或 `vis`！

本页要点

定位口诀：**MLE/TLE** 先查标记有没有写在 `push` 前，**RE** 查越界与负下标，**WA** 查初值与松弛！

提交前检查清单（六必查）

1. 是否在把新状态塞进队列 `q.push` 的那一行立刻设置了 `dist` 或 `vis`？
2. 访问新坐标 `nx`，`ny` 或新状态值 `y` 前，有没有先拦截 < 0 或 $\geq$ 数组上界？
3. 题面要求的是“最少移动步数”（起点初值 0）还是“经过格点总数”（起点初值 1）？
4. 《抓住那头牛》一类可超出目标后再折返的问题，状态数组上限是否开到了足够大（如 $2 \times \max$）？
5. 若起点本身就是终点（如 $N == K$），代码是否不进扩展循环也能直接输出正确答案？
6. 存在无解情况时，是否正确初始化 `dist = -1` 并按题面要求输出 `-1` 或 `oop!`？

最后一步

对起点即终点、周围全是墙无解、以及绕路越界折返三类小样例跑一遍再提交！

本章全部练习与完成标准

1. [入门] 走迷宫（一本通 1252，code/maze_shortest.cpp）：练二维网格 BFS 与含起终点格子数计数。
2. [入门] 献给阿尔吉侬的花束（一本通 1256）：练多组数据找 'S' 到 'E' 最短步数及无解输出 oop!。
3. [基础] 仙岛求药（一本通 1251）：练多组输入以 0 0 结束、避开 '#' 到 '*' 最少步数（-1 无解）。
4. [基础] 抓住那头牛（一本通 1253，code/catch_cow.cpp）：练一维状态空间建模与 2×10^5 上界防 RE。
5. [基础] 细胞（一本通 1329，code/cells.cpp）：练 BFS 洪水填充（Flood Fill）求四连通块数。
6. [基础] **Knight Moves**：练 300×300 棋盘马跳八方向最短步（对比上一章马走日）。
7. [提高] **Dungeon Master**：练三维迷宫 (z, x, y) 六方向扩展（上下层 + 四邻）。

完成标准

理解为什么 BFS 层序扩展能命中无权最短路；熟记入队即标铁律；独立 AC 前 5 题并挑战后 2 题。

课后安排与下一篇预告

- 复习：合上课件默写带 dx/dy 方向数组的网格 BFS 模板、多源入队写法与 0-1 BFS 双端队列写法。
- 练习：独立复写 code/ 下 3 份源码；把《走迷宫》里的 dist[nx][ny]=... 故意挪到 pop() 后对比体会爆队列报错。
- 下一篇预告：正式开启《04_ 数据结构》第 1 章《栈与单调栈》——认识后进先出（LIFO）的栈结构，并看一个单调栈如何在 $O(n)$ 时间内一趟扫完所有“左边第一个比自己大/小”的难题！

本章一句话

一层一层往外扩，入队时刻就把距离刻下来，第一次摸到终点就是最优答案。

谢谢！