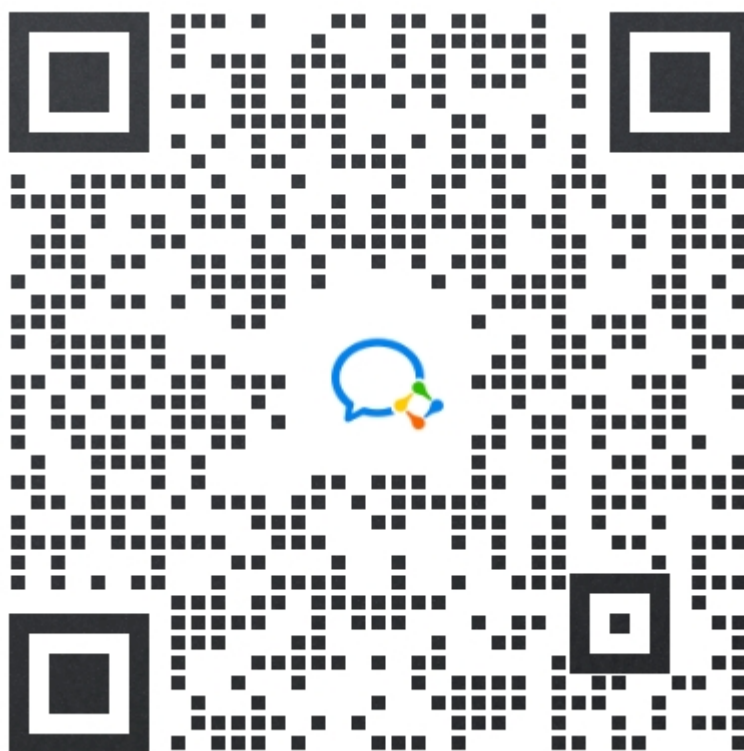


# 优选算法精品课(No.082~No.100)

## 版权说明

本“**比特就业课**”优选算法精品课(No.082~No.100)（以下简称“本精品课”）的所有内容，包括但不限于文字、图片、音频、视频、软件、程序、数据库、设计、布局、界面等，均由本精品课的开发或授权方拥有版权。我们鼓励个人学习者使用本精品课进行学习和研究。在遵守相关法律法规的前提下，个人学习者可以下载、浏览、学习本精品课的内容，并为了个人学习、研究或教学目的而使用其中的材料。但请注意，**未经我们明确授权，个人学习者不得将本精品课的内容用于任何商业目的**，包括但不限于销售、转让、许可或以其他方式从中获利。此外，个人学习者也不得擅自修改、复制、传播、展示、表演或制作本精品课内容的衍生作品。任何未经授权的使用均属侵权行为，我们将依法追究法律责任。如果您希望以其他方式使用本精品课的内容，包括但不限于引用、转载、摘录、改编等，请事先与我们联系，获取书面授权。感谢您对“比特就业课”优选算法精品课(No.082~No.100)的关注与支持，我们将持续努力，为您提供更好的学习体验。特此说明。比特就业课版权所有方。

对比特算法感兴趣，可以联系这个微信。



## 板书链接

## BFS 解决 FloodFill 算法

### 78. 图像渲染 (medium)

#### 1. 题目链接: [733. 图像渲染](#)

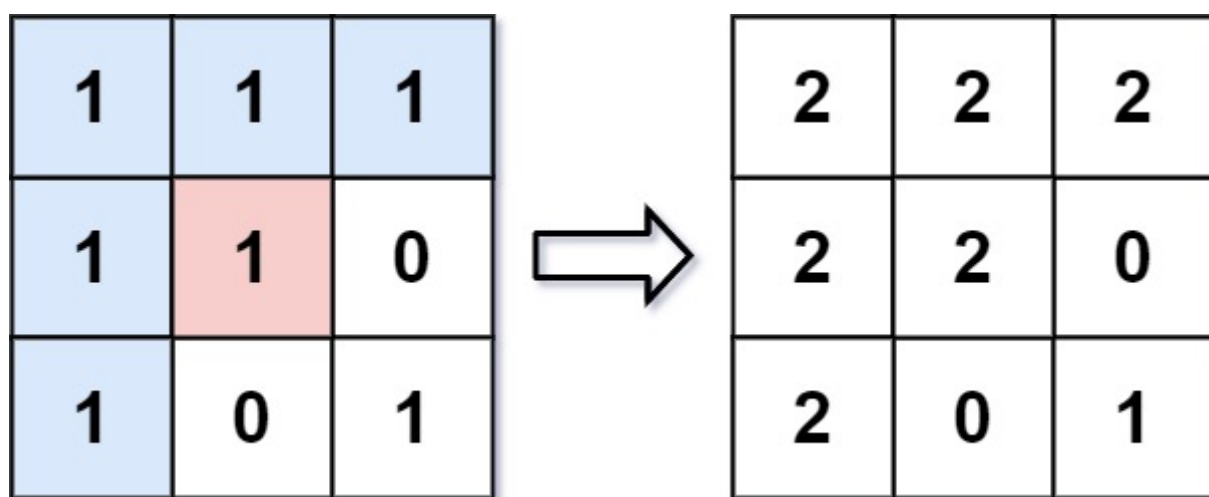
#### 2. 题目描述:

有一幅以  $m \times n$  的二维整数数组表示的图画 `image`，其中 `image[i][j]` 表示该图画的像素值大小。你也被给予三个整数 `sr`，`sc` 和 `newColor`。你应该从像素 `image[sr][sc]` 开始对图像进行上色填充。

为了完成上色工作，从初始像素开始，记录初始坐标的上下左右四个方向上像素值与初始坐标相同的相连像素点，接着再记录这四个方向上符合条件的像素点与他们对应四个方向上像素值与初始坐标相同的相连像素点，……，重复该过程。将所有有记录的像素点的颜色值改为 `newColor`。

最后返回 经过上色渲染后的图像。

示例 1:



输入: `image = [[1,1,1],[1,1,0],[1,0,1]]`, `sr = 1`, `sc = 1`, `newColor = 2`

输出: `[[2,2,2],[2,2,0],[2,0,1]]`

解析: 在图像的正中间, (坐标(`sr,sc`)=(1,1)),在路径上所有符合条件的像素点的颜色都被更改成 2。

注意, 右下角的像素没有更改为 2, 因为它不是在上下左右四个方向上与初始点相连的像素点。

示例 2:

输入: image = [[0,0,0],[0,0,0]], sr = 0, sc = 0, newColor = 2

输出: [[2,2,2],[2,2,2]]

### 3. 算法思路:

可以利用「深搜」或者「宽搜」，遍历到与该点相连的所有「像素相同的点」，然后将其修改成指定的像素即可。

### C++ 算法代码:

```
1 class Solution
2 {
3     typedef pair<int, int> PII;
4
5     int dx[4] = {0, 0, 1, -1};
6     int dy[4] = {1, -1, 0, 0};
7
8 public:
9     vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc,
10     int color)
11     {
12         int prev = image[sr][sc];
13         if(prev == color) return image;
14         int m = image.size(), n = image[0].size();
15
16         queue<PII> q;
17         q.push({sr, sc});
18
19         while(!q.empty())
20         {
21             auto [a, b] = q.front();
22             image[a][b] = color;
23             q.pop();
24             for(int i = 0; i < 4; i++)
25             {
26                 int x = a + dx[i], y = b + dy[i];
27                 if(x >= 0 && x < m && y >= 0 && y < n && image[x][y] == prev)
28                 {
29                     q.push({x, y});
30                 }
31             }
32         }
33     }
```

```
32
33     return image;
34 }
35 };
```

## C++ 运行结果:

C++

时间 4 ms

击败 97.88%

内存 14.1 MB

击败 24.40%

## Java 算法代码:

```
1 class Solution
2 {
3     int[] dx = {0, 0, 1, -1};
4     int[] dy = {1, -1, 0, 0};
5
6     public int[][] floodFill(int[][] image, int sr, int sc, int color)
7     {
8         int prev = image[sr][sc]; // 统计刚开始的颜色
9         if(prev == color) return image; // 处理边界情况
10
11         int m = image.length, n = image[0].length;
12
13         Queue<int[]> q = new LinkedList<>();
14         q.add(new int[]{sr, sc});
15
16         while(!q.isEmpty())
17         {
18             int[] t = q.poll();
19             int a = t[0], b = t[1];
20             image[a][b] = color;
21             // 上下左右四个方向
22             for(int i = 0; i < 4; i++)
23             {
24                 int x = a + dx[i], y = b + dy[i];
25                 if(x >= 0 && x < m && y >= 0 && y < n && image[x][y] == prev)
26                 {
27                     q.add(new int[]{x, y});
28                 }
29             }
30         }
31     }
32 }
```

```
31         return image;
32     }
33 }
```

## Java 运行结果：



## 79. 岛屿数量 (medium)

### 1. 题目链接：200. 岛屿数量

### 2. 题目描述：

给你一个由 '1'（陆地）和 '0'（水）组成的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

示例 1：

输入：grid = [

```
["1","1","1","1","0"],
["1","1","0","1","0"],
["1","1","0","0","0"],
["0","0","0","0","0"]
]
```

输出：1

示例 2：

输入：grid = [

```
["1","1","0","0","0"],
["1","1","0","0","0"],
["0","0","1","0","0"],
]
```

```
["0","0","0","1","1"]
```

```
]
```

输出：3

提示：

```
m == grid.length
```

```
n == grid[i].length
```

```
1 <= m, n <= 300
```

```
grid[i][j] 的值为 '0' 或 '1'
```

### 算法思路：

遍历整个矩阵，每次找到「一块陆地」的时候：

- 说明找到「一个岛屿」，记录到最终结果 `ret` 里面；
- 并且将这个陆地相连的所有陆地，也就是这块「岛屿」，全部「变成海洋」。这样的话，我们下次遍历到这块岛屿的时候，它「已经是海洋」了，不会影响最终结果。
- 其中「变成海洋」的操作，可以利用「深搜」和「宽搜」解决，其实就是 [733. 图像渲染](#) 这道题~

这样，当我们，遍历完全部的矩阵的时候，`ret` 存的就是最终结果。

### C++ 算法代码：

```
1 class Solution
2 {
3     int dx[4] = {1, -1, 0, 0};
4     int dy[4] = {0, 0, 1, -1};
5     bool vis[301][301];
6     int m, n;
7
8 public:
9     int numIslands(vector<vector<char>>& grid)
10    {
11        m = grid.size(), n = grid[0].size();
12        int ret = 0;
13        for(int i = 0; i < m; i++)
14        {
15            for(int j = 0; j < n; j++)
```

```

16         {
17             if(grid[i][j] == '1' && !vis[i][j])
18             {
19                 ret++;
20                 bfs(grid, i, j); // 把这块陆地全部标记一下
21             }
22         }
23     }
24     return ret;
25 }
26
27 void bfs(vector<vector<char>>& grid, int i, int j)
28 {
29     queue<pair<int, int>> q;
30     q.push({i, j});
31     vis[i][j] = true;
32
33     while(q.size())
34     {
35         auto [a, b] = q.front();
36         q.pop();
37         for(int k = 0; k < 4; k++)
38         {
39             int x = a + dx[k], y = b + dy[k];
40             if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == '1' &&
!vis[x][y])
41             {
42                 q.push({x, y});
43                 vis[x][y] = true;
44             }
45         }
46     }
47 }
48
49 };

```

## C++ 运行结果:

C++

时间 32 ms

击败 65.90%

内存 12.2 MB

击败 41.84%

## Java 算法代码:

```

1 class Solution
2 {
3     int[] dx = {0, 0, -1, 1};
4     int[] dy = {1, -1, 0, 0};
5     boolean[][] vis = new boolean[301][301];
6     int m, n;
7
8     public int numIslands(char[][] grid)
9     {
10         m = grid.length; n = grid[0].length;
11
12         int ret = 0;
13         for(int i = 0; i < m; i++)
14         {
15             for(int j = 0; j < n; j++)
16             {
17                 if(grid[i][j] == '1' && !vis[i][j])
18                 {
19                     ret++;
20                     bfs(grid, i, j);
21                 }
22             }
23         }
24         return ret;
25     }
26
27     public void bfs(char[][] grid, int i, int j)
28     {
29         Queue<int[]> q = new LinkedList<>();
30         q.add(new int[]{i, j});
31         vis[i][j] = true;
32
33         while(!q.isEmpty())
34         {
35             int[] t = q.poll();
36             int a = t[0], b = t[1];
37             for(int k = 0; k < 4; k++)
38             {
39                 int x = a + dx[k], y = b + dy[k];
40                 if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == '1' &&
!vis[x][y])
41                 {
42                     q.add(new int[]{x, y});
43                     vis[x][y] = true;
44                 }
45             }

```



```
46      }
47      }
48  }
```

## Java 运行结果：



## 80. 岛屿的最大面积 (medium)

### 1. 题目链接：[695. 岛屿的最大面积](#)

### 2. 题目描述：

给你一个大小为  $m \times n$  的二进制矩阵 `grid`。

岛屿是由一些相邻的 1 (代表土地) 构成的组合，这里的「相邻」要求两个 1 必须在水平或者竖直的四个方向上相邻。你可以假设 `grid` 的四个边缘都被 0 (代表水) 包围着。

岛屿的面积是岛上值为 1 的单元格的数目。

计算并返回 `grid` 中最大的岛屿面积。如果没有岛屿，则返回面积为 0。

示例 1：

0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	1	0	1	0	0
0	1	0	0	1	1	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0

输入：

```
grid = [[0,0,1,0,0,0,0,1,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,1,1,0,0,0],
        [0,1,1,0,1,0,0,0,0,0,0,0,0],
        [0,1,0,0,1,1,0,0,1,0,1,0,0],
        [0,1,0,0,1,1,0,0,1,1,1,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,1,0],
        [0,0,0,0,0,0,0,0,1,1,0,0,0],
        [0,0,0,0,0,0,0,0,1,1,0,0,0]]
```

输出：6

解释：

答案不应该是 11 ， 因为岛屿只能包含水平或垂直这四个方向上的 1 。

示例 2：

输入：

```
grid = [[0,0,0,0,0,0,0]]
```

输出：0

提示：

```
m == grid.length
n == grid[i].length
1 <= m, n <= 50
grid[i][j] 为 0 或 1
```

### 3. 算法思路：

- 遍历整个矩阵，每当遇到一块土地的时候，就用「深搜」或者「宽搜」将与这块土地相连的「整个岛屿」的面积计算出来。
- 然后在搜索得到的「所有的岛屿面积」求一个「最大值」即可。
- 在搜索过程中，为了「防止搜到重复的土地」：
  - 可以开一个同等规模的「布尔数组」，标记一下这个位置是否已经被访问过；
  - 也可以将原始矩阵的 `1` 修改成 `0`，但是这样操作会修改原始矩阵。

### C++ 算法代码：

```
1 class Solution
2 {
3     int dx[4] = {0, 0, 1, -1};
4     int dy[4] = {1, -1, 0, 0};
5     bool vis[51][51];
6     int m, n;
7
8 public:
9     int maxAreaOfIsland(vector<vector<int>>& grid)
10    {
11        m = grid.size(), n = grid[0].size();
12
13        int ret = 0;
14        for(int i = 0; i < m; i++)
15        {
16            for(int j = 0; j < n; j++)
17            {
18                if(grid[i][j] == 1 && !vis[i][j])
19                {
20                    ret = max(ret, bfs(grid, i, j));
21                }
22            }
23        }
24        return ret;
25    }
26
27    int bfs(vector<vector<int>>& grid, int i, int j)
28    {
29        int count = 0;
30        queue<pair<int, int>> q;
31        q.push({i, j});
32        vis[i][j] = true;
```

```

33         count++;
34
35         while(q.size())
36         {
37             auto [a, b] = q.front();
38             q.pop();
39             for(int k = 0; k < 4; k++)
40             {
41                 int x = a + dx[k], y = b + dy[k];
42                 if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == 1 &&
!vis[x][y])
43                 {
44                     q.push({x, y});
45                     vis[x][y] = true;
46                     count++;
47                 }
48             }
49         }
50         return count;
51     }
52 };

```

## C++ 运行结果:

C++



## Java 算法代码:

```

1 class Solution
2 {
3     int[] dx = {0, 0, 1, -1};
4     int[] dy = {1, -1, 0, 0};
5     boolean[][] vis = new boolean[51][51];
6     int m, n;
7
8     public int maxAreaOfIsland(int[][] grid)
9     {
10         m = grid.length; n = grid[0].length;
11
12         int ret = 0;

```

```

13         for(int i = 0; i < m; i++)
14         {
15             for(int j = 0; j < n; j++)
16             {
17                 if(grid[i][j] == 1 && !vis[i][j])
18                 {
19                     ret = Math.max(ret, bfs(grid, i, j));
20                 }
21             }
22         }
23         return ret;
24     }
25
26     public int bfs(int[][] grid, int i, int j)
27     {
28         int count = 0;
29
30         Queue<int[]> q = new LinkedList<>();
31         q.add(new int[]{i, j});
32         vis[i][j] = true;
33         count++;
34
35         while(!q.isEmpty())
36         {
37             int[] t = q.poll();
38             int a = t[0], b = t[1];
39             for(int k = 0; k < 4; k++)
40             {
41                 int x = a + dx[k], y = b + dy[k];
42                 if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == 1 &&
!vis[x][y])
43                 {
44                     q.offer(new int[]{x, y});
45                     vis[x][y] = true;
46                     count++;
47                 }
48             }
49         }
50         return count;
51     }
52 }

```

**Java 运行结果：**

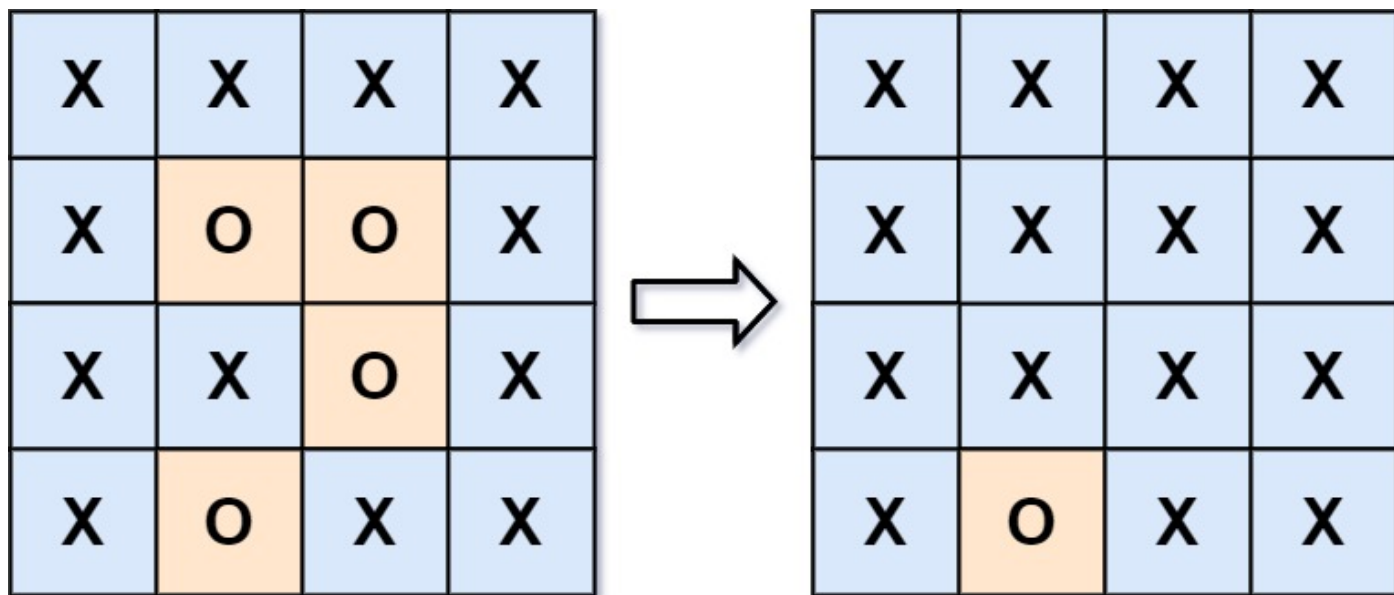
## 81. 被围绕的区域 (medium)

### 1. 题目链接: [130. 被围绕的区域](#)

### 2. 题目描述:

给你一个  $m \times n$  的矩阵 `board`，由若干字符 'X' 和 'O'，找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。

示例 1:



输入: `board = [["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]`

输出: `[["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]`

解释: 被围绕的区间不会存在于边界上, 换句话说, 任何边界上的 'O' 都不会被填充为 'X'。任何不在边界上, 或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻, 则称它们是“相连”的。

示例 2:

输入: `board = [["X"]]`

输出: `[["X"]]`

提示:

```
m == board.length
n == board[i].length
1 <= m, n <= 200
board[i][j] 为 'X' 或 'O'
```

### 3. 解法:

#### 算法思路:

正难则反。

可以先利用 `bfs` 将与边缘相连的 `'O'` 区域做上标记，然后重新遍历矩阵，将没有标记过的 `'O'` 修改成 `'X'` 即可。

#### C++ 算法代码:

```
1 class Solution
2 {
3     int dx[4] = {0, 0, 1, -1};
4     int dy[4] = {1, -1, 0, 0};
5     int m, n;
6
7 public:
8     void solve(vector<vector<char>>& board)
9     {
10         m = board.size(), n = board[0].size();
11
12         // 1. 先处理边界上的 'O' 联通块, 全部修改成 '.'
13         for(int j = 0; j < n; j++)
14         {
15             if(board[0][j] == 'O') bfs(board, 0, j);
16             if(board[m - 1][j] == 'O') bfs(board, m - 1, j);
17         }
18
19         for(int i = 0; i < m; i++)
20         {
21             if(board[i][0] == 'O') bfs(board, i, 0);
22             if(board[i][n - 1] == 'O') bfs(board, i, n - 1);
23         }
24
25         // 2. 还原
26         for(int i = 0; i < m; i++)
27             for(int j = 0; j < n; j++)
28                 if(board[i][j] == 'O') board[i][j] = 'X';
29                 else if(board[i][j] == '.') board[i][j] = 'O';
```

```

30     }
31
32     void bfs(vector<vector<char>>& board, int i, int j)
33     {
34         queue<pair<int, int>> q;
35         q.push({i, j});
36         board[i][j] = '.';
37
38         while(q.size())
39         {
40             auto [a, b] = q.front();
41             q.pop();
42             for(int k = 0; k < 4; k++)
43             {
44                 int x = a + dx[k], y = b + dy[k];
45                 if(x >= 0 && x < m && y >= 0 && y < n && board[x][y] == '0')
46                 {
47                     q.push({x, y});
48                     board[x][y] = '.';
49                 }
50             }
51         }
52     }
53 };

```

## C++ 代码结果:

C++

时间 16 ms

击败 36.8%

内存 9.5 MB

击败 97.88%

## Java 算法代码:

```

1 class Solution
2 {
3     int[] dx = {0, 0, 1, -1};
4     int[] dy = {1, -1, 0, 0};
5     int m, n;
6
7     public void solve(char[][] board)
8     {
9         m = board.length; n = board[0].length;

```



```

10
11 // 1. 先处理边界的 'O' 联通块, 全部修改成 '.'
12 for(int j = 0; j < n; j++)
13 {
14     if(board[0][j] == 'O') bfs(board, 0, j);
15     if(board[m - 1][j] == 'O') bfs(board, m - 1, j);
16 }
17
18 for(int i = 0; i < m; i++)
19 {
20     if(board[i][0] == 'O') bfs(board, i, 0);
21     if(board[i][n - 1] == 'O') bfs(board, i, n - 1);
22 }
23
24 // 2. 还原
25 for(int i = 0; i < m; i++)
26     for(int j = 0; j < n; j++)
27         if(board[i][j] == 'O') board[i][j] = 'X';
28         else if(board[i][j] == '.') board[i][j] = 'O';
29 }
30
31 public void bfs(char[][] board, int i, int j)
32 {
33     Queue<int[]> q = new LinkedList<>();
34     q.add(new int[]{i, j});
35     board[i][j] = '.';
36
37     while(!q.isEmpty())
38     {
39         int[] t = q.poll();
40         int a = t[0], b = t[1];
41         for(int k = 0; k < 4; k++)
42         {
43             int x = a + dx[k], y = b + dy[k];
44             if(x >= 0 && x < m && y >= 0 && y < n && board[x][y] == 'O')
45             {
46                 board[x][y] = '.';
47                 q.add(new int[]{x, y});
48             }
49         }
50     }
51 }
52 }

```

**Java 运行结果：**



## BFS 解决最短路问题

### 82. 迷宫中离入口最近的出口 (medium)

#### 1. 题目链接：1926. 迷宫中离入口最近的出口

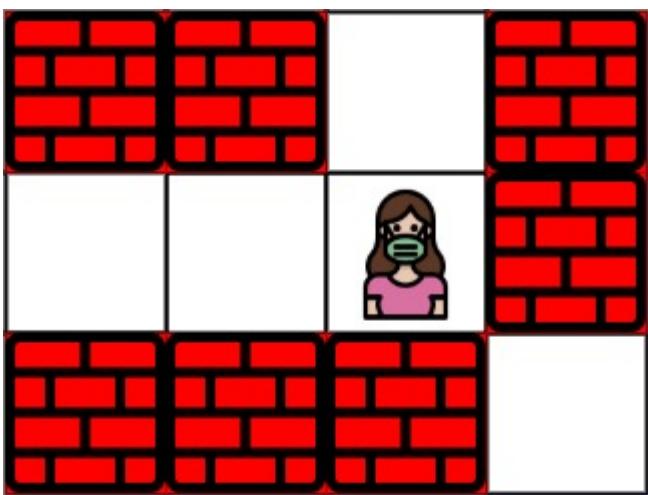
#### 2. 题目描述：

给你一个  $m \times n$  的迷宫矩阵 `maze`（下标从 0 开始），矩阵中有空格子（用 `' '` 表示）和墙（用 `'+'` 表示）。同时给你迷宫的入口 `entrance`，用 `entrance = [entrancerow, entrancecol]` 表示你一开始所在格子的行和列。

每一步操作，你可以往 上，下，左 或者 右 移动一个格子。你不能进入墙所在的格子，你也不能离开迷宫。你的目标是找到离 `entrance` 最近 的出口。出口 的含义是 `maze` 边界上的 空格子。  
`entrance` 格子 不算 出口。

请你返回从 `entrance` 到最近出口的最短路径的 步数，如果不存在这样的路径，请你返回 -1。

示例 1：



输入：`maze = [[ "+","+",";","+" ],[ "",";",";","+" ],[ "+","+","+",";" ], entrance = [1,2]`

输出：1

解释：

总共有 3 个出口，分别位于 (1,0)，(0,2) 和 (2,3)。

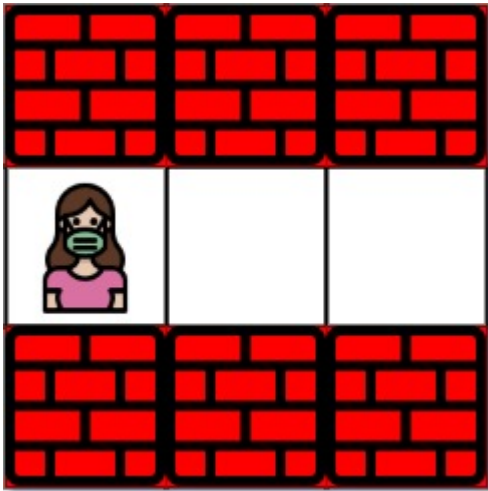
一开始，你在入口格子 (1,2) 处。

- 你可以往左移动 2 步到达 (1,0)。
- 你可以往上移动 1 步到达 (0,2)。

从入口处没法到达 (2,3)。

所以，最近的出口是 (0,2)，距离为 1 步。

示例 2：



输入：maze = [["+", "+", "+", "+"], [":", ":", ":", ":"], ["+", "+", "+", "+"]], entrance = [1,0]

输出：2

解释：

迷宫中只有 1 个出口，在 (1,2) 处。

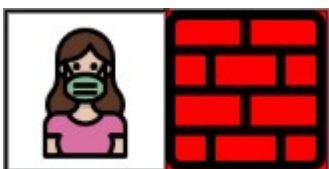
(1,0) 不算出口，因为它是入口格子。

初始时，你在入口与格子 (1,0) 处。

- 你可以往右移动 2 步到达 (1,2) 处。

所以，最近的出口为 (1,2)，距离为 2 步。

示例 3：



输入：maze = [".","+"], entrance = [0,0]

输出：-1

解释：

这个迷宫中没有出口。

提示：

maze.length == m

maze[i].length == n

1 <= m, n <= 100

maze[i][j] 要么是 '.'，要么是 '+'。

entrance.length == 2

0 <= entrancerow < m

0 <= entrancecol < n

entrance 一定是空格子。

### 3. 解法（bfs 求最短路）：

算法思路：

利用层序遍历来解决迷宫问题，是最经典的做法。

我们可以从起点开始层序遍历，并且在遍历的过程中记录当前遍历的层数。这样就能在找到出口的时候，得到起点到出口的最短距离。

C++ 算法代码：

```
1 class Solution
2 {
3     int dx[4] = {0, 0, 1, -1};
4     int dy[4] = {1, -1, 0, 0};
5
6 public:
7     int nearestExit(vector<vector<char>>& maze, vector<int>& e)
8     {
9         int m = maze.size(), n = maze[0].size();
10
11         bool vis[m][n];
12         memset(vis, 0, sizeof vis);
13
14         queue<pair<int, int>> q;
```

```

15     q.push({e[0], e[1]});
16     vis[e[0]][e[1]] = true;
17
18     int step = 0;
19     while(q.size())
20     {
21         step++;
22         int sz = q.size();
23         for(int i = 0; i < sz; i++)
24         {
25             auto [a, b] = q.front();
26             q.pop();
27             for(int j = 0; j < 4; j++)
28             {
29                 int x = a + dx[j], y = b + dy[j];
30                 if(x >= 0 && x < m && y >= 0 && y < n && maze[x][y] == '.'
&& !vis[x][y])
31                 {
32                     // 判断是否已经到达出口
33                     if(x == 0 || x == m - 1 || y == 0 || y == n - 1) return
step;
34                     q.push({x, y});
35                     vis[x][y] = true;
36                 }
37             }
38         }
39     }
40
41     return -1;
42 }
43 };

```

## C++ 运行结果:

C++



## Java 算法代码:

```

1 class Solution
2 {
3     int[] dx = {0, 0, -1, 1};

```

```

4     int[] dy = {1, -1, 0, 0};
5
6     public int nearestExit(char[][] maze, int[] e)
7     {
8         int m = maze.length, n = maze[0].length;
9
10        boolean[][] vis = new boolean[m][n];
11
12        Queue<int[]> q = new LinkedList<>();
13        q.add(new int[]{e[0], e[1]});
14        vis[e[0]][e[1]] = true;
15        int step = 0;
16
17        while(!q.isEmpty())
18        {
19            step++;
20            int sz = q.size();
21            for(int i = 0; i < sz; i++)
22            {
23                int[] t = q.poll();
24                int a = t[0], b = t[1];
25                for(int j = 0; j < 4; j++)
26                {
27                    int x = a + dx[j], y = b + dy[j];
28                    if(x >= 0 && x < m && y >= 0 && y < n && maze[x][y] == '.'
    && !vis[x][y])
29                    {
30                        // 判断是否已经到出口
31                        if(x == 0 || x == m - 1 || y == 0 || y == n - 1) return
    step;
32                        vis[x][y] = true;
33                        q.add(new int[]{x, y});
34                    }
35                }
36            }
37        }
38        return -1;
39    }
40 }

```

## Java 运行结果：

Java



## 83. 最小基因变化 (medium)

### 1. 题目链接: [433. 最小基因变化](#)

### 2. 题目描述:

基因序列可以表示为一条由 8 个字符组成的字符串，其中每个字符都是 'A'、'C'、'G' 和 'T' 之一。

假设我们需要调查从基因序列 `start` 变为 `end` 所发生的基因变化。一次基因变化就意味着这个基因序列中的一个字符发生了变化。

- 例如, "AACCGGTT" --> "AACCGGTA" 就是一次基因变化。

另有一个基因库 `bank` 记录了所有有效的基因变化，只有基因库中的基因才是有效的基因序列。（变化后的基因必须位于基因库 `bank` 中）

给你两个基因序列 `start` 和 `end`，以及一个基因库 `bank`，请你找出并返回能够使 `start` 变化为 `end` 所需的最少变化次数。如果无法完成此基因变化，返回 `-1`。

注意：起始基因序列 `start` 默认是有效的，但是它并不一定会出现在基因库中。

#### 示例 1:

输入: `start = "AACCGGTT", end = "AACCGGTA", bank = ["AACCGGTA"]`

输出: 1

#### 示例 2:

输入: `start = "AACCGGTT", end = "AAACGGTA", bank = ["AACCGGTA", "AACCGCTA", "AAACGGTA"]`

输出: 2

#### 示例 3:

输入: `start = "AAAAACCC", end = "AACCCCCC", bank = ["AAAACCCC", "AAACCCCC", "AACCCCCC"]`

输出: 3

#### 提示:

- `start.length == 8`
- `end.length == 8`
- `0 <= bank.length <= 10`
- `bank[i].length == 8`

- `start`、`end` 和 `bank[i]` 仅由字符 `['A', 'C', 'G', 'T']` 组成

### 3. 解法:

#### 算法思路:

如果将「每次字符串的变换」抽象成图中的「两个顶点和一条边」的话，问题就变成了「边权为 1 的最短路问题」。

因此，从起始的字符串开始，来一次 `bfs` 即可。

#### C++ 算法代码:

```
1 class Solution
2 {
3 public:
4     int minMutation(string startGene, string endGene, vector<string>& bank)
5     {
6         unordered_set<string> vis; // 用来标记已经搜索过的状态
7         unordered_set<string> hash(bank.begin(), bank.end()); // 存储基因库里面的
            字符串
8         string change = "ACGT";
9
10        if(startGene == endGene) return 0;
11        if(!hash.count(endGene)) return -1;
12
13        queue<string> q;
14        q.push(startGene);
15        vis.insert(startGene);
16
17        int ret = 0;
18        while(q.size())
19        {
20            ret++;
21            int sz = q.size();
22            while(sz--)
23            {
24                string t = q.front();
25                q.pop();
26                for(int i = 0; i < 8; i++)
27                {
28                    string tmp = t; // 细节问题
29                    for(int j = 0; j < 4; j++)
30                    {
31                        tmp[i] = change[j];
32                        if(hash.count(tmp) && !vis.count(tmp))
33                        {
```



```

34         if(tmp == endGene) return ret;
35         q.push(tmp);
36         vis.insert(tmp);
37     }
38 }
39 }
40 }
41 }
42 return -1;
43 }
44 };

```

## C++ 代码结果:

C++



## Java 算法代码:

```

1 class Solution
2 {
3     public int minMutation(String startGene, String endGene, String[] bank)
4     {
5         Set<String> vis = new HashSet<>(); // 用来标记已经搜索过的状态
6         Set<String> hash = new HashSet<>(); // 用来统计基因库里面的字符串
7         for(String s : bank) hash.add(s);
8         char[] change = {'A', 'C', 'G', 'T'};
9
10        if(startGene.equals(endGene)) return 0;
11        if(!hash.contains(endGene)) return -1;
12
13        Queue<String> q = new LinkedList<>();
14        q.add(startGene);
15        vis.add(startGene);
16        int step = 0;
17
18        while(!q.isEmpty())
19        {
20            step++;
21            int sz = q.size();
22            while(sz-- != 0)
23            {

```

```

24         String t = q.poll();
25         for(int i = 0; i < 8; i++)
26         {
27             char[] tmp = t.toCharArray();
28             for(int j = 0; j < 4; j++)
29             {
30                 tmp[i] = change[j];
31                 String next = new String(tmp);
32                 if(hash.contains(next) && !vis.contains(next))
33                 {
34                     if(next.equals(endGene)) return step;
35                     q.add(next);
36                     vis.add(next);
37                 }
38             }
39         }
40     }
41 }
42 return -1;
43 }
44 }

```

## Java 运行结果：



## 84. 单词接龙 (hard)

### 1. 题目链接：127. 单词接龙

### 2. 题目描述：

字典 `wordList` 中从单词 `beginWord` 和 `endWord` 的 **转换序列** 是一个按下述规格形成的序列 `beginWord -> s(1) -> s(2) -> ... -> s(k)`：

- 每一对相邻的单词只差一个字母。
- 对于  $1 \leq i \leq k$  时，每个 `s(i)` 都在 `wordList` 中。注意，`beginWord` 不需要在 `wordList` 中。
- `s(k) == endWord`

给你两个单词 `beginWord` 和 `endWord` 和一个字典 `wordList`，返回从 `beginWord` 到 `endWord` 的 **最短转换序列** 中的 **单词数目**。如果不存在这样的转换序列，返回 `0`。

### 示例 1:

输入: `beginWord = "hit"`, `endWord = "cog"`, `wordList = ["hot","dot","dog","lot","log","cog"]`

输出: 5

解释: 一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog", 返回它的长度 5。

### 示例 2:

输入: `beginWord = "hit"`, `endWord = "cog"`, `wordList = ["hot","dot","dog","lot","log"]`

输出: 0

解释: `endWord "cog"` 不在字典中，所以无法进行转换。

### 提示:

- `1 <= beginWord.length <= 10`
- `endWord.length == beginWord.length`
- `1 <= wordList.length <= 5000`
- `wordList[i].length == beginWord.length`
- `beginWord`、`endWord` 和 `wordList[i]` 由小写英文字母组成
- `beginWord != endWord`
- `wordList` 中的所有字符串 **互不相同**

## 3. 解法:

### 算法思路:

跟上题一样~

### C++ 算法代码:

```
1 class Solution
2 {
3 public:
4     int ladderLength(string beginWord, string endWord, vector<string>&
        wordList)
5     {
6         unordered_set<string> hash(wordList.begin(), wordList.end());
7         unordered_set<string> vis; // 标记已经搜索过的单词
8     }
```

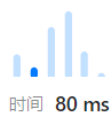
```

9         if(!hash.count(endWord)) return 0;
10
11         queue<string> q;
12         q.push(beginWord);
13         vis.insert(beginWord);
14
15         int ret = 1;
16         while(q.size())
17         {
18             ret++;
19             int sz = q.size();
20             while(sz--)
21             {
22                 string t = q.front();
23                 q.pop();
24                 for(int i = 0; i < t.size(); i++)
25                 {
26                     string tmp = t;
27                     for(char ch = 'a'; ch <= 'z'; ch++)
28                     {
29                         tmp[i] = ch;
30                         if( hash.count(tmp) && !vis.count(tmp))
31                         {
32                             if(tmp == endWord) return ret;
33                             q.push(tmp);
34                             vis.insert(tmp);
35                         }
36                     }
37                 }
38             }
39         }
40         return 0;
41     }
42 };

```

## C++ 代码结果:

C++



击败 90.48%



击败 54.86%

## Java 算法代码:

```

1 class Solution
2 {
3     public int ladderLength(String beginWord, String endWord, List<String>
wordList)
4     {
5         Set<String> hash = new HashSet<>();
6         for(String s : wordList) hash.add(s);
7         Set<String> vis = new HashSet<>(); // 标记已经搜索过的单词
8
9         if(!hash.contains(endWord)) return 0;
10
11         Queue<String> q = new LinkedList<>();
12         q.add(beginWord);
13         vis.add(beginWord);
14
15         int ret = 1;
16         while(!q.isEmpty())
17         {
18             ret++;
19             int sz = q.size();
20             while(sz-- != 0)
21             {
22                 String t = q.poll();
23                 for(int i = 0; i < t.length(); i++)
24                 {
25                     char[] tmp = t.toCharArray();
26                     for(char ch = 'a'; ch <= 'z'; ch++)
27                     {
28                         tmp[i] = ch;
29                         String next = new String(tmp);
30                         if(hash.contains(next) && !vis.contains(next))
31                         {
32                             if(next.equals(endWord)) return ret;
33                             q.add(next);
34                             vis.add(next);
35                         }
36                     }
37                 }
38             }
39         }
40         return 0;
41     }
42 }

```

**Java 运行结果：**

## 85. 为高尔夫比赛砍树 (hard)

### 1. 题目链接：675. 为高尔夫比赛砍树

### 2. 题目描述：

你被请来给一个要举办高尔夫比赛的树林砍树。树林由一个  $m \times n$  的矩阵表示，在这个矩阵中：

- 0 表示障碍，无法触碰
- 1 表示地面，可以行走
- 比 1 大的数 表示有树的单元格，可以行走，数值表示树的高度

每一步，你都可以向上、下、左、右四个方向之一移动一个单位，如果你站的地方有一棵树，那么你可以决定是否要砍倒它。

你需要按照树的高度从低向高砍掉所有的树，每砍过一颗树，该单元格的值变为 1（即变为地面）。

你将从 (0, 0) 点开始工作，返回你砍完所有树需要走的最小步数。如果你无法砍完所有的树，返回 -1。

可以保证的是，没有两棵树的高度是相同的，并且你至少需要砍倒一棵树。

### 示例 1：

1	→	2	→	3
0		0		↓
				↓
7	←	6	←	5

输入：forest = [[1,2,3],[0,0,4],[7,6,5]]

输出：6

**解释：**沿着上面的路径，你可以用 6 步，按从最矮到最高的顺序砍掉这些树。

**示例 2：**

1	2	3
0	0	0
7	6	5

**输入：**forest = [[1,2,3],[0,0,0],[7,6,5]]

**输出：**-1

**解释：**由于中间一行被障碍阻塞，无法访问最下面一行中的树。

**示例 3：**

**输入：**forest = [[2,3,4],[0,0,5],[8,7,6]]

**输出：**6

**解释：**可以按与示例 1 相同的路径来砍掉所有的树。

**提示：**

- `m == forest.length`
- `n == forest[i].length`
- `1 <= m, n <= 50`
- `0 <= forest[i][j] <= 10^9`

**3. 解法：**

**算法思路：**

- 先找出砍树的顺序；
- 然后按照砍树的顺序，一个一个的用 `bfs` 求出最短路径即可。

**C++ 算法代码：**

```

1 class Solution
2 {
3     int m, n;
4 public:
5     int cutOffTree(vector<vector<int>>& f)
6     {
7         m = f.size(), n = f[0].size();
8         // 1. 准备工作: 找出砍树的顺序
9         vector<pair<int, int>> trees;
10        for(int i = 0; i < m; i++)
11            for(int j = 0; j < n; j++)
12                if(f[i][j] > 1) trees.push_back({i, j});
13
14        sort(trees.begin(), trees.end(), [&](const pair<int, int>& p1, const
pair<int, int>& p2)
15            {
16                return f[p1.first][p1.second] < f[p2.first][p2.second];
17            });
18
19        // 2. 按照顺序砍树
20        int bx = 0, by = 0;
21        int ret = 0;
22        for(auto& [a, b] : trees)
23        {
24            int step = bfs(f, bx, by, a, b);
25            if(step == -1) return -1;
26            ret += step;
27            bx = a, by = b;
28        }
29        return ret;
30    }
31
32    bool vis[51][51];
33    int dx[4] = {0, 0, 1, -1};
34    int dy[4] = {1, -1, 0, 0};
35
36    int bfs(vector<vector<int>>& f, int bx, int by, int ex, int ey)
37    {
38        if(bx == ex && by == ey) return 0;
39
40        queue<pair<int, int>> q;
41        memset(vis, 0, sizeof vis); // 清空之前的数据
42        q.push({bx, by});
43        vis[bx][by] = true;
44
45        int step = 0;
46        while(q.size())

```



```

47     {
48         step++;
49         int sz = q.size();
50         while(sz-->0)
51         {
52             auto [a, b] = q.front();
53             q.pop();
54             for(int i = 0; i < 4; i++)
55             {
56                 int x = a + dx[i], y = b + dy[i];
57                 if(x >= 0 && x < m && y >= 0 && y < n && f[x][y] && !vis[x]
[y])
58                     {
59                         if(x == ex && y == ey) return step;
60                         q.push({x, y});
61                         vis[x][y] = true;
62                     }
63             }
64         }
65     }
66     return -1;
67 }
68 };

```

## C++ 代码结果:

C++



## Java 算法代码:

```

1 class Solution
2 {
3     int m, n;
4     public int cutOffTree(List<List<Integer>> f)
5     {
6         m = f.size(); n = f.get(0).size();
7         List<int[]> trees = new ArrayList<>();
8         for(int i = 0; i < m; i++)
9             for(int j = 0; j < n; j++)
10                 if(f.get(i).get(j) > 1)
11                     trees.add(new int[]{i, j});

```

```

12
13     Collections.sort(trees, (a, b) ->
14     {
15         return f.get(a[0]).get(a[1]) - f.get(b[0]).get(b[1]);
16     });
17
18     int bx = 0, by = 0;
19     int ret = 0;
20     for(int[] next : trees)
21     {
22         int a = next[0], b = next[1];
23         int step = bfs(f, bx, by, a, b);
24         if(step == -1) return -1;
25         ret += step;
26         bx = a; by = b;
27     }
28     return ret;
29 }
30
31 int[] dx = {0, 0, 1, -1};
32 int[] dy = {1, -1, 0, 0};
33
34 public int bfs(List<List<Integer>> f, int bx, int by, int ex, int ey)
35 {
36     if(bx == ex && by == ey) return 0;
37
38     Queue<int[]> q = new LinkedList<>();
39     boolean[][] vis = new boolean[m][n];
40     q.add(new int[]{bx, by});
41     vis[bx][by] = true;
42
43     int step = 0;
44     while(!q.isEmpty())
45     {
46         int sz = q.size();
47         step++;
48         while(sz-- != 0)
49         {
50             int[] t = q.poll();
51             int a = t[0], b = t[1];
52             for(int i = 0; i < 4; i++)
53             {
54                 int x = a + dx[i], y = b + dy[i];
55                 if(x >= 0 && x < m && y >= 0 && y < n && f.get(x).get(y)
56                 != 0 && !vis[x][y])
57                 {
58                     if(x == ex && y == ey) return step;

```

```
58         q.add(new int[]{x, y});
59         vis[x][y] = true;
60     }
61 }
62 }
63
64 }
65 return -1;
66 }
67 }
```

## Java 运行结果：



## 多源 BFS

### 86. 01 矩阵 (medium)

1. 题目链接：[542. 01 矩阵](#)

2. 题目描述：

给定一个由 0 和 1 组成的矩阵 mat，请输出一个大小相同的矩阵，其中每一个格子是 mat 中对应位置元素到最近的 0 的距离。

两个相邻元素间的距离为 1。

示例 1：

0	0	0
0	1	0
0	0	0

输入：mat = [[0,0,0],[0,1,0],[0,0,0]]

输出：[[0,0,0],[0,1,0],[0,0,0]]

示例 2：

0	0	0
0	1	0
1	1	1

输入：mat = [[0,0,0],[0,1,0],[1,1,1]]

输出：[[0,0,0],[0,1,0],[1,2,1]]

提示：

m == mat.length

n == mat[i].length

1 <= m, n <= 104

1 <= m \* n <= 104

mat[i][j] is either 0 or 1.

mat 中至少有一个 0

### 3. 解法 (bfs) (多个源头的最短路问题)

#### 算法思路:

对于求的最终结果，我们有两种方式：

- 第一种方式：从每一个 1 开始，然后通过层序遍历找到离它最近的 0。

这一种方式，我们会以所有的 1 起点，来一次层序遍历，势必会遍历到很多重复的点。并且如果矩阵中只有一个 0 的话，每一次层序遍历都要遍历很多层，时间复杂度较高。

- 换一种方式：从 0 开始层序遍历，并且记录遍历的层数。当第一次碰到 1 的时候，当前的层数就是这个 1 离 0 的最短距离。

这一种方式，我们在遍历的时候标记一下处理过的 1，能够做到只用遍历整个矩阵一次，就能得到最终结果。

但是，这里有一个问题，0 是有很多个的，我们怎样才能保证遇到的 1 距离这一个 0 是最近的呢？

其实很简单，我们可以先把所有的 0 都放在队列中，把它们当成一个整体，每次把当前队列里面的所有元素向外扩展一次。

#### C++ 算法代码:

```
1 class Solution
2 {
3     int dx[4] = {0, 0, 1, -1};
4     int dy[4] = {1, -1, 0, 0};
5
6 public:
7     vector<vector<int>> updateMatrix(vector<vector<int>>& mat)
8     {
9         int m = mat.size(), n = mat[0].size();
10
11         // dist[i][j] == -1 表示：没有搜索过
12         // dist[i][j] != -1 表示：最短距离
13         vector<vector<int>> dist(m, vector<int>(n, -1));
14         queue<pair<int, int>> q;
15
16         // 1. 把所有的源点加入到队列中
17         for(int i = 0; i < m; i++)
18             for(int j = 0; j < n; j++)
19                 if(mat[i][j] == 0)
20                     {
21                         q.push({i, j});
```

```

22         dist[i][j] = 0;
23     }
24
25     // 2. 一层一层的往外扩
26     while(q.size())
27     {
28         auto [a, b] = q.front(); q.pop();
29         for(int i = 0; i < 4; i++)
30         {
31             int x = a + dx[i], y = b + dy[i];
32             if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
33             {
34                 dist[x][y] = dist[a][b] + 1;
35                 q.push({x, y});
36             }
37         }
38     }
39     return dist;
40 }
41 };

```

## C++ 运行结果:

C++



## Java 算法代码:

```

1 class Solution
2 {
3     int[] dx = {0, 0, -1, 1};
4     int[] dy = {1, -1, 0, 0};
5
6     public int[][] updateMatrix(int[][] mat)
7     {
8         int m = mat.length, n = mat[0].length;
9         // dist[i][j] == -1: 标记当前位置没有被搜索过
10        // dist[i][j] != -1: 存的是最短距离
11        int[][] dist = new int[m][n];
12        for(int i = 0; i < m; i++)
13            for(int j = 0; j < n; j++)

```

```

14         dist[i][j] = -1;
15     Queue<int[]> q = new LinkedList<>();
16
17     // 1. 把所有的源点加入到队列里面
18     for(int i = 0; i < m; i++)
19         for(int j = 0; j < n; j++)
20             {
21                 if(mat[i][j] == 0)
22                     {
23                         dist[i][j] = 0;
24                         q.add(new int[]{i, j});
25                     }
26             }
27
28     // 2. 一层一层的往外扩
29     while(!q.isEmpty())
30     {
31         int[] t = q.poll();
32         int a = t[0], b = t[1];
33         for(int i = 0; i < 4; i++)
34             {
35                 int x = a + dx[i], y = b + dy[i];
36                 if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
37                     {
38                         dist[x][y] = dist[a][b] + 1;
39                         q.add(new int[]{x, y});
40                     }
41             }
42     }
43     return dist;
44 }
45 }

```

## Java 运行结果：

Java



## 87. 飞地的数量 (medium)

1. 题目链接：[1020. 飞地的数量](#)

2. 题目描述：

给你一个大小为 `m x n` 的二进制矩阵 `grid`，其中 `0` 表示一个海洋单元格、`1` 表示一个陆地单元格。

一次 **移动** 是指从一个陆地单元格走到另一个相邻（上、下、左、右）的陆地单元格或跨过 `grid` 的边界。

返回网格中 **无法** 在任意次数的移动中离开网格边界的陆地单元格的数量。

示例 1:

0	0	0	0
1	0	1	0
0	1	1	0
0	0	0	0

输入: `grid = [[0,0,0,0],[1,0,1,0],[0,1,1,0],[0,0,0,0]]`

输出: 3

解释: 有三个 1 被 0 包围。一个 1 没有被包围，因为它在边界上。

示例 2:

0	1	1	0
0	0	1	0
0	0	1	0
0	0	0	0



输入: grid = [[0,1,1,0],[0,0,1,0],[0,0,1,0],[0,0,0,0]]

输出: 0

解释: 所有 1 都在边界上或可以到达边界。

提示:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 500`
- `grid[i][j]` 的值为 0 或 1

### 3. 解法:

算法思路:

正难则反:

从边上的 1 开始搜索, 把与边上 1 相连的联通区域全部标记一下;

然后再遍历一遍矩阵, 看看哪些位置的 1 没有被标记即可

标记的时候, 可以用「多源 bfs」解决。

C++ 算法代码:

```
1 class Solution
2 {
3     int dx[4] = {0, 0, 1, -1};
4     int dy[4] = {1, -1, 0, 0};
5
6 public:
7     int numEnclaves(vector<vector<int>>& grid)
8     {
9         int m = grid.size(), n = grid[0].size();
10        vector<vector<bool>> vis(m, vector<bool>(n));
11        queue<pair<int, int>> q;
12
13        // 1. 把边上的 1 加入到队列中
14        for(int i = 0; i < m; i++)
15            for(int j = 0; j < n; j++)
16                if(i == 0 || i == m - 1 || j == 0 || j == n - 1)
17                    {
18                        if(grid[i][j] == 1)
19                            {
20                                q.push({i, j});
```

```

21         vis[i][j] = true;
22     }
23 }
24 // 2. 多源 bfs
25 while(q.size())
26 {
27     auto [a, b] = q.front();
28     q.pop();
29     for(int i = 0; i < 4; i++)
30     {
31         int x = a + dx[i], y = b + dy[i];
32         if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == 1 &&
!vis[x][y])
33         {
34             vis[x][y] = true;
35             q.push({x, y});
36         }
37     }
38 }
39 // 3. 统计结果
40 int ret = 0;
41 for(int i = 0; i < m; i++)
42     for(int j = 0; j < n; j++)
43         if(grid[i][j] == 1 && !vis[i][j])
44             ret++;
45 return ret;
46 }
47 };

```

## C++ 代码结果:



## Java 算法代码:

```

1 class Solution
2 {
3     int[] dx = {0, 0, 1, -1};
4     int[] dy = {1, -1, 0, 0};
5

```

```

6     public int numEnclaves(int[][] grid)
7     {
8         int m = grid.length, n = grid[0].length;
9         boolean[][] vis = new boolean[m][n];
10        Queue<int[]> q = new LinkedList<>();
11
12        // 1. 把边上的 1 全部加入到队列中
13        for(int i = 0; i < m; i++)
14            for(int j = 0; j < n; j++)
15                if(i == 0 || i == m - 1 || j == 0 || j == n - 1)
16                {
17                    if(grid[i][j] == 1)
18                    {
19                        q.add(new int[]{i, j});
20                        vis[i][j] = true;
21                    }
22                }
23        // 2. 多源 bfs
24        while(!q.isEmpty())
25        {
26            int[] t = q.poll();
27            int a = t[0], b = t[1];
28            for(int i = 0; i < 4; i++)
29            {
30                int x = a + dx[i], y = b + dy[i];
31                if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == 1 &&
!vis[x][y])
32                {
33                    q.add(new int[]{x, y});
34                    vis[x][y] = true;
35                }
36            }
37        }
38        // 3. 提取结果
39        int ret = 0;
40        for(int i = 0; i < m; i++)
41            for(int j = 0; j < n; j++)
42                if(grid[i][j] == 1 && !vis[i][j])
43                    ret++;
44        return ret;
45    }
46 }

```

**Java 运行结果：**

## 88. 地图中的最高点 (medium)

### 1. 题目链接: 1765. 地图中的最高点

### 2. 题目描述:

给你一个大小为  $m \times n$  的整数矩阵 `isWater`，它代表了一个由 **陆地** 和 **水域** 单元格组成的地图。

- 如果 `isWater[i][j] == 0`，格子  $(i, j)$  是一个 **陆地** 格子。
- 如果 `isWater[i][j] == 1`，格子  $(i, j)$  是一个 **水域** 格子。

你需要按照如下规则给每个单元格安排高度：

- 每个格子的高度都必须是非负的。
- 如果一个格子是 **水域**，那么它的高度必须为 `0`。
- 任意相邻的格子高度差 **至多** 为 `1`。当两个格子在正东、南、西、北方向上相互紧挨着，就称它们为相邻的格子。（也就是说它们有一条公共边）

找到一种安排高度的方案，使得矩阵中的最高高度值 **最大**。

请你返回一个大小为  $m \times n$  的整数矩阵 `height`，其中 `height[i][j]` 是格子  $(i, j)$  的高度。如果有多种解法，请返回 **任意一个**。

### 示例 1:

1	0
2	1

输入: isWater = [[0,1],[0,0]]

输出: [[1,0],[2,1]]

解释: 上图展示了给各个格子安排的高度。

蓝色格子是水域格, 绿色格子是陆地格。

示例 2:

1	1	0
0	1	1
1	2	2

输入: isWater = [[0,0,1],[1,0,0],[0,0,0]]

输出: [[1,1,0],[0,1,1],[1,2,2]]

解释: 所有安排方案中, 最高可行高度为 2。

任意安排方案中, 只要最高高度为 2 且符合上述规则的, 都为可行方案。

提示:

- `m == isWater.length`
- `n == isWater[i].length`
- `1 <= m, n <= 1000`
- `isWater[i][j]` 要么是 0, 要么是 1。
- 至少有 1 个水域格子。

3. 解法:

算法思路:

01矩阵的变型题, 直接用多源 bfs 解决即可。

## C++ 算法代码：

```
1 class Solution
2 {
3     int dx[4] = {0, 0, 1, -1};
4     int dy[4] = {1, -1, 0, 0};
5
6 public:
7     vector<vector<int>> highestPeak(vector<vector<int>>& isWater)
8     {
9         int m = isWater.size(), n = isWater[0].size();
10        vector<vector<int>> dist(m, vector<int>(n, -1));
11        queue<pair<int, int>> q;
12
13        // 1. 把所有的源点加入到队列里面
14        for(int i = 0; i < m; i++)
15            for(int j = 0; j < n; j++)
16                if(isWater[i][j])
17                    {
18                        dist[i][j] = 0;
19                        q.push({i, j});
20                    }
21        // 2. 多源 bfs
22        while(q.size())
23        {
24            auto [a, b] = q.front(); q.pop();
25            for(int i = 0; i < 4; i++)
26            {
27                int x = a + dx[i], y = b + dy[i];
28                if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
29                {
30                    dist[x][y] = dist[a][b] + 1;
31                    q.push({x, y});
32                }
33            }
34        }
35        return dist;
36    }
37 };
```

## C++ 代码结果：



时间 276 ms

击败 77.50%



内存 106 MB

击败 56.67%

## Java 算法代码:

```
1 class Solution
2 {
3     int[] dx = {0, 0, -1, 1};
4     int[] dy = {1, -1, 0, 0};
5
6     public int[][] highestPeak(int[][] isWater)
7     {
8         int m = isWater.length, n = isWater[0].length;
9         int[][] dist = new int[m][n];
10        for(int i = 0; i < m; i++)
11            for(int j = 0; j < n; j++)
12                dist[i][j] = -1;
13        Queue<int[]> q = new LinkedList<>();
14
15        // 1. 所有的源点加入到队列里面
16        for(int i = 0; i < m; i++)
17            for(int j = 0; j < n; j++)
18                if(isWater[i][j] == 1)
19                {
20                    q.add(new int[]{i, j});
21                    dist[i][j] = 0;
22                }
23        // 2. 多源 bfs
24        while(!q.isEmpty())
25        {
26            int[] t = q.poll();
27            int a = t[0], b = t[1];
28            for(int i = 0; i < 4; i++)
29            {
30                int x = a + dx[i], y = b + dy[i];
31                if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
32                {
33                    dist[x][y] = dist[a][b] + 1;
34                    q.add(new int[]{x, y});
35                }
36            }
37        }
38    }
39 }
```

```
38         return dist;
39     }
40 }
```

## Java 运行结果：

Java



## 89. 地图分析 (medium)

### 1. 题目链接：1162. 地图分析

### 2. 题目描述：

你现在手里有一份大小为  $n \times n$  的网格 `grid`，上面的每个单元格都用 `0` 和 `1` 标记好了。其中 `0` 代表海洋，`1` 代表陆地。

请你找出一个海洋单元格，这个海洋单元格到离它最近的陆地单元格的距离是最大的，并返回该距离。如果网格上只有陆地或者海洋，请返回 `-1`。

我们这里说的距离是「曼哈顿距离」 (Manhattan Distance)：  $(x_0, y_0)$  和  $(x_1, y_1)$  这两个单元格之间的距离是  $|x_0 - x_1| + |y_0 - y_1|$ 。

### 示例 1：

1	0	1
0	0	0
1	0	1

输入：grid = [[1,0,1],[0,0,0],[1,0,1]]

输出：2

解释：

海洋单元格 (1, 1) 和所有陆地单元格之间的距离都达到最大，最大距离为 2。

### 示例 2：

1	0	0
0	0	0
0	0	0



输入: grid = [[1,0,0],[0,0,0],[0,0,0]]

输出: 4

解释:

海洋单元格 (2, 2) 和所有陆地单元格之间的距离都达到最大，最大距离为 4。

提示:

- `n == grid.length`
- `n == grid[i].length`
- `1 <= n <= 100`
- `grid[i][j]` 不是 0 就是 1

### 3. 解法:

算法思路:

01矩阵的变型题，直接用多源 bfs 解决即可。

C++ 算法代码:

```
1 class Solution
2 {
3     int dx[4] = {0, 0, 1, -1};
4     int dy[4] = {1, -1, 0, 0};
5
6 public:
7     int maxDistance(vector<vector<int>>& grid)
8     {
9         int m = grid.size(), n = grid[0].size();
10        vector<vector<int>> dist(m, vector<int>(n, -1));
11        queue<pair<int, int>> q;
12
13        for(int i = 0; i < m; i++)
14            for(int j = 0; j < n; j++)
15                if(grid[i][j])
16                {
17                    dist[i][j] = 0;
18                    q.push({i, j});
19                }
20
21        int ret = -1;
22        while(q.size())
```

```

23     {
24         auto [a, b] = q.front(); q.pop();
25         for(int i = 0; i < 4; i++)
26         {
27             int x = a + dx[i], y = b + dy[i];
28             if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
29             {
30                 dist[x][y] = dist[a][b] + 1;
31                 q.push({x, y});
32                 ret = max(ret, dist[x][y]);
33             }
34         }
35     }
36     return ret;
37 }
38 };

```

## C++ 代码结果:

C++

时间 52 ms

击败 75.55%

内存 21.3 MB

击败 30.32%

## Java 算法代码:

```

1 class Solution
2 {
3     int[] dx = {0, 0, 1, -1};
4     int[] dy = {1, -1, 0, 0};
5
6     public int maxDistance(int[][] grid)
7     {
8         int m = grid.length, n = grid[0].length;
9         int[][] dist = new int[m][n];
10        for(int i = 0; i < m; i++)
11            for(int j = 0; j < n; j++)
12                dist[i][j] = -1;
13        Queue<int[]> q = new LinkedList<>();
14
15        for(int i = 0; i < m; i++)
16            for(int j = 0; j < n; j++)
17                if(grid[i][j] == 1)

```

```

18         {
19             dist[i][j] = 0;
20             q.add(new int[]{i, j});
21         }
22
23     int ret = -1;
24     while(!q.isEmpty())
25     {
26         int[] t = q.poll();
27         int a = t[0], b = t[1];
28         for(int i = 0; i < 4; i++)
29         {
30             int x = a + dx[i], y = b + dy[i];
31             if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
32             {
33                 dist[x][y] = dist[a][b] + 1;
34                 q.add(new int[]{x, y});
35                 ret = Math.max(ret, dist[x][y]);
36             }
37         }
38     }
39     return ret;
40 }
41 }

```

## Java 运行结果：



## BFS 解决拓扑排序

### 90. 课程表 (medium)

#### 1. 题目链接：207. 课程表

#### 2. 题目描述：

你这个学期必须选修 `numCourses` 门课程，记为 `0` 到 `numCourses - 1`。

在选修某些课程之前需要一些先修课程。先修课程按数组 `prerequisites` 给出，其中

`prerequisites[i] = [a(i), b(i)]`，表示如果要学习课程 `a(i)` 则 **必须** 先学习课程

`b(i)()`。

- 例如，先修课程对 `[0, 1]` 表示：想要学习课程 `0`，你需要先完成课程 `1`。

请你判断是否可能完成所有课程的学习？如果可以，返回 `true`；否则，返回 `false`。

### 示例 1：

输入：numCourses = 2, prerequisites = [[1,0]]

输出：true

解释：总共有 2 门课程。学习课程 1 之前，你需要完成课程 0。这是可能的。

### 示例 2：

输入：numCourses = 2, prerequisites = [[1,0],[0,1]]

输出：false

解释：总共有 2 门课程。学习课程 1 之前，你需要先完成课程 0；并且学习课程 0 之前，你还应先完成课程 1。这是不可能的。

### 提示：

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= a(i), b(i) < numCourses`
- `prerequisites[i]` 中的所有课程对 **互不相同**

## 3. 解法：

### 算法思路：

原问题可以转换成一个拓扑排序问题。

用 BFS 解决拓扑排序即可。

### 拓扑排序流程：

- 将所有入度为 0 的点加入到队列中；
- 当队列不空的时候，一直循环：
  - 取出队头元素；
  - 将队头元素相连的顶点的入度 - 1；
  - 然后判断是否减成 0，如果减成 0，就加入到队列中。

## C++ 算法代码：

```
1 class Solution
2 {
3 public:
4     bool canFinish(int n, vector<vector<int>>& p)
5     {
6         unordered_map<int, vector<int>> edges; // 邻接表
7         vector<int> in(n); // 存储每一个结点的入度
8
9         // 1. 建图
10        for(auto& e : p)
11        {
12            int a = e[0], b = e[1];
13            edges[b].push_back(a);
14            in[a]++;
15        }
16
17        // 2. 拓扑排序 - bfs
18        queue<int> q;
19        // 把所有入度为 0 的点加入到队列中
20        for(int i = 0; i < n; i++)
21        {
22            if(in[i] == 0) q.push(i);
23        }
24
25        // 层序遍历
26        while(!q.empty())
27        {
28            int t = q.front();
29            q.pop();
30            // 修改相连的边
31            for(int e : edges[t])
32            {
33                in[e]--;
34                if(in[e] == 0) q.push(e);
35            }
36        }
37
38        // 3. 判断是否有环
39        for(int i : in)
40            if(i)
41                return false;
42
43        return true;
```

```
44     }
45 };
```

## C++ 代码结果:



## Java 算法代码:

```
1 class Solution
2 {
3     public boolean canFinish(int n, int[][] p)
4     {
5         // 1. 准备工作
6         int[] in = new int[n]; // 统计每一个顶点的入度
7         Map<Integer, List<Integer>> edges = new HashMap<>(); // 邻接表存图
8
9         // 2. 建图
10        for(int i = 0; i < p.length; i++)
11        {
12            int a = p[i][0], b = p[i][1]; // b -> a
13            if(!edges.containsKey(b))
14            {
15                edges.put(b, new ArrayList<>());
16            }
17            edges.get(b).add(a);
18            in[a]++;
19        }
20
21        // 3. 拓扑排序
22        Queue<Integer> q = new LinkedList<>();
23        // (1) 先把入度为 0 的点, 加入到队列中
24        for(int i = 0; i < n; i++)
25        {
26            if(in[i] == 0) q.add(i);
27        }
28
29        // (2) bfs
30        while(!q.isEmpty())
31        {
32            int t = q.poll();
33        }
```

```

34         for(int a : edges.getOrDefault(t, new ArrayList<>()))
35         {
36             in[a]--;
37             if(in[a] == 0) q.add(a);
38         }
39     }
40
41     // 4. 判断是否有环
42     for(int x : in)
43         if(x != 0) return false;
44
45     return true;
46 }
47 }

```

## Java 运行结果：



## 91. 课程表II (medium)

### 1. 题目链接：210. 课程表 II

### 2. 题目描述：

现在你总共有 `numCourses` 门课程需要选，记为 `0` 到 `numCourses - 1`。给你一个数组 `prerequisites`，其中 `prerequisites[i] = [a(i), b(i)]`，表示在选修课程 `a(i)` 前 **必须** 先选修 `b(i)`。

- 例如，想要学习课程 `0`，你需要先完成课程 `1`，我们用一个匹配来表示：`[0,1]`。

返回你为了学完所有课程所安排的学习顺序。可能会有多个正确的顺序，你只要返回 **任意一种** 就可以了。如果不可能完成所有课程，返回 **一个空数组**。

#### 示例 1：

**输入：**`numCourses = 2, prerequisites = [[1,0]]`

**输出：**`[0,1]`

**解释：**总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 `[0,1]`。

#### 示例 2：

**输入：**`numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]`

输出: [0,2,1,3]

解释: 总共有 4 门课程。要学习课程 3, 你应该先完成课程 1 和课程 2。并且课程 1 和课程 2 都应该排在课程 0 之后。[0,1,2,3]。另一个正确的排序是 [0,2,1,3]。

### 示例 3:

输入: numCourses = 1, prerequisites = []

输出: [0]

### 提示:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= numCourses * (numCourses - 1)`
- `prerequisites[i].length == 2`
- `0 <= a(i), b(i) < numCourses`
- `a(i) != b(i)`
- 所有 `[a(i), b(i)]` 互不相同

### 3. 解法:

#### 算法思路:

和上一题一样~

#### C++ 算法代码:

```
1 class Solution
2 {
3 public:
4     vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites)
5     {
6         // 1. 准备工作
7         vector<vector<int>> edges(numCourses); // 邻接表存储图
8         vector<int> in(numCourses); // 存储每一个点的入度
9
10        // 2. 建图
11        for(auto& p : prerequisites)
12        {
13            int a = p[0], b = p[1]; // b -> a
14            edges[b].push_back(a);
15            in[a]++;
16        }
17    }
```



```

18      // 3. 拓扑排序
19      vector<int> ret; // 统计排序后的结果
20      queue<int> q;
21      for(int i = 0; i < numCourses; i++)
22      {
23          if(in[i] == 0) q.push(i);
24      }
25      while(q.size())
26      {
27          int t = q.front(); q.pop();
28          ret.push_back(t);
29          for(int a : edges[t])
30          {
31              in[a]--;
32              if(in[a] == 0) q.push(a);
33          }
34      }
35      if(ret.size() == numCourses) return ret;
36      else return {};
37  }
38  };

```

## C++ 代码结果:



## Java 算法代码:

```

1  class Solution
2  {
3      public int[] findOrder(int n, int[][] prerequisites)
4      {
5          // 1. 准备工作
6          int[] in = new int[n]; // 统计每个顶点的入度
7          List<List<Integer>> edges = new ArrayList<>();
8          for(int i = 0; i < n; i++)
9          {
10             edges.add(new ArrayList<>());
11         }
12
13         // 2. 建图
14         for(int i = 0; i < prerequisites.length; i++)

```

```

15     {
16         int a = prerequisites[i][0], b = prerequisites[i][1]; // b -> a
17         edges.get(b).add(a);
18         in[a]++;
19     }
20
21     // 3. 拓扑排序
22     Queue<Integer> q = new LinkedList<>();
23     int[] ret = new int[n];
24     int index = 0;
25     for(int i = 0; i < n; i++)
26     {
27         if(in[i] == 0) q.add(i);
28     }
29
30     while(!q.isEmpty())
31     {
32         int t = q.poll();
33         ret[index++] = t;
34         for(int a : edges.get(t))
35         {
36             in[a]--;
37             if(in[a] == 0) q.add(a);
38         }
39     }
40
41     if(index == n) return ret;
42     return new int[0];
43 }
44 }

```

## Java 运行结果：



## 92. 火星词典 (hard)

### 1. 题目链接：LCR 114. 火星词典

### 2. 题目描述：

现有一种使用英语字母的外星文语言，这门语言的字母顺序与英语顺序不同。

给定一个字符串列表 `words`，作为这门语言的词典，`words` 中的字符串已经 **按这门新语言的字母顺序进行了排序**。

请你根据该词典还原出此语言中已知的字母顺序，并 **按字母递增顺序** 排列。若不存在合法字母顺序，返回 `""`。若存在多种可能的合法字母顺序，返回其中 **任意一种** 顺序即可。

字符串 `s` **字典顺序小于** 字符串 `t` 有两种情况：

- 在第一个不同字母处，如果 `s` 中的字母在这门外星语言的字母顺序中位于 `t` 中字母之前，那么 `s` 的字典顺序小于 `t`。
- 如果前面 `min(s.length, t.length)` 字母都相同，那么 `s.length < t.length` 时，`s` 的字典顺序也小于 `t`。

**示例 1：**

输入：words = ["wrt","wrf","er","ett","rftt"]

输出："wertf"

**示例 2：**

输入：words = ["z","x"]

输出："zx"

**示例 3：**

输入：words = ["z","x","z"]

输出：""

解释：不存在合法字母顺序，因此返回 ""。

**提示：**

- `1 <= words.length <= 100`
- `1 <= words[i].length <= 100`
- `words[i]` 仅由小写英文字母组成

### 3. 解法：

**算法思路：**

将题意搞清楚之后，这道题就变成了判断有向图时候有环，可以用拓扑排序解决。

**如何搜集信息（如何建图）：**

- 两层 for 循环枚举出所有的两个字符串的组合；
- 然后利用指针，根据字典序规则找出信息。

## C++ 算法代码：

```
1 class Solution
2 {
3     unordered_map<char, unordered_set<char>> edges; // 邻接表来存储图
4     unordered_map<char, int> in; // 统计入度
5     bool check; // 处理边界情况
6
7 public:
8     string alienOrder(vector<string>& words)
9     {
10         // 1. 建图 + 初始化入度哈希表
11         for(auto& s : words)
12         {
13             for(auto ch : s)
14             {
15                 in[ch] = 0;
16             }
17         }
18         int n = words.size();
19         for(int i = 0; i < n; i++)
20         {
21             for(int j = i + 1; j < n; j++)
22             {
23                 add(words[i], words[j]);
24                 if(check) return "";
25             }
26         }
27
28         // 2. 拓扑排序
29         queue<char> q;
30         for(auto& [a, b] : in)
31         {
32             if(b == 0) q.push(a);
33         }
34         string ret;
35         while(q.size())
36         {
37             char t = q.front(); q.pop();
38             ret += t;
39             for(char ch : edges[t])
40             {
41                 if(--in[ch] == 0) q.push(ch);
42             }
43         }
```

```

44
45     // 3. 判断
46     for(auto& [a, b] : in)
47         if(b != 0) return "";
48
49     return ret;
50 }
51
52 void add(string& s1, string& s2)
53 {
54     int n = min(s1.size(), s2.size());
55     int i = 0;
56     for( ; i < n; i++)
57     {
58         if(s1[i] != s2[i])
59         {
60             char a = s1[i], b = s2[i]; // a -> b
61             if(!edges.count(a) || !edges[a].count(b))
62             {
63                 edges[a].insert(b);
64                 in[b]++;
65             }
66             break;
67         }
68     }
69     if(i == s2.size() && i < s1.size()) cheak = true;
70 }
71 };

```

## C++ 代码结果:



## Java 算法代码:

```

1 class Solution
2 {
3     Map<Character, Set<Character>> edges = new HashMap<>(); // 邻接表
4     Map<Character, Integer> in = new HashMap<>(); // 统计每个节点的入度
5     boolean check;
6
7     public String alienOrder(String[] words)

```

```
8      {
9          // 1. 初始化入度哈希表 + 建图
10         for(String s : words)
11         {
12             for(int i = 0; i < s.length(); i++)
13             {
14                 char ch = s.charAt(i);
15                 in.put(ch, 0);
16             }
17         }
18         int n = words.length;
19         for(int i = 0; i < n; i++)
20         {
21             for(int j = i + 1; j < n; j++)
22             {
23                 add(words[i], words[j]);
24                 if(check == true) return "";
25             }
26         }
27
28         // 2. 拓扑排序
29         Queue<Character> q = new LinkedList<>();
30         for(char ch : in.keySet())
31         {
32             if(in.get(ch) == 0) q.add(ch);
33         }
34         StringBuffer ret = new StringBuffer();
35         while(!q.isEmpty())
36         {
37             char t = q.poll();
38             ret.append(t);
39
40             if(!edges.containsKey(t)) continue;
41
42             for(char ch : edges.get(t))
43             {
44                 in.put(ch, in.get(ch) - 1);
45                 if(in.get(ch) == 0) q.add(ch);
46             }
47         }
48
49         // 3. 判断
50         for(char ch : in.keySet())
51         {
52             if(in.get(ch) != 0) return "";
53         }
54         return ret.toString();
```

```

55     }
56
57     public void add(String s1, String s2)
58     {
59         int n = Math.min(s1.length(), s2.length());
60         int i = 0;
61         for( ; i < n; i++)
62         {
63             char c1 = s1.charAt(i), c2 = s2.charAt(i);
64             if(c1 != c2)
65             {
66                 // c1 -> c2
67                 if(!edges.containsKey(c1))
68                 {
69                     edges.put(c1, new HashSet<>());
70                 }
71                 if(!edges.get(c1).contains(c2))
72                 {
73                     edges.get(c1).add(c2);
74                     in.put(c2, in.get(c2) + 1);
75                 }
76                 break;
77             }
78         }
79         if(i == s2.length() && i < s1.length()) check = true;
80     }
81 }

```

## Java 运行结果：

