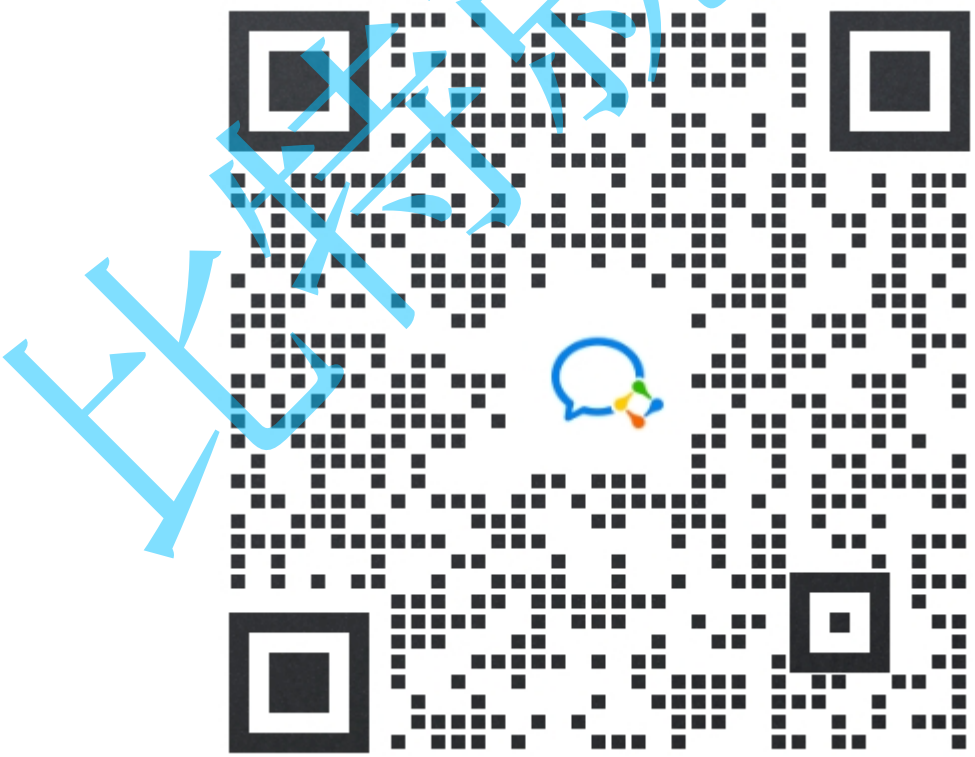
优选算法精品课(No.082~No.100)

版权说明

对比特算法感兴趣，可以联系这个微信。



板书链接

https://gitee.com/wu-xiaozhe/algorithm

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：**

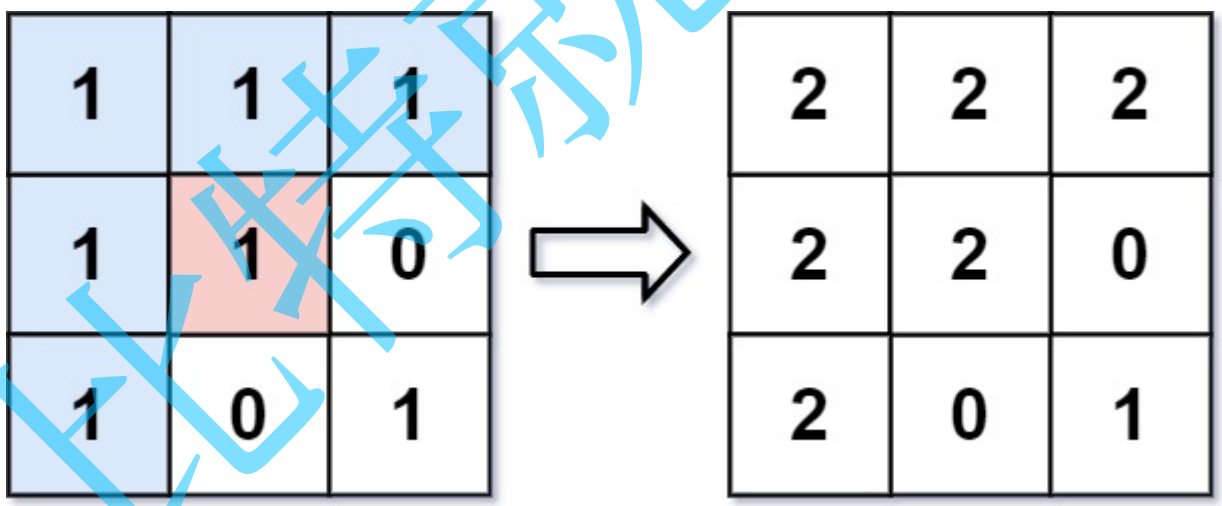
有一幅以 m x n 的二维整数数组表示的图画 image ，其中 image[i][j] 表示该图画的像素值大小。

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：

```
class Solution
{
    typedef pair<int, int> PII;

    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};

public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc,
int color)
    {
        int prev = image[sr][sc];
        if(prev == color) return image;
        int m = image.size(), n = image[0].size();

        queue<PII> q;
        q.push({sr, sc});

        while(!q.empty())
        {
            auto [a, b] = q.front();
            image[a][b] = color;
            q.pop();
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && image[x][y] == prev)
                {
                    q.push({x, y});
                }
            }
        }
```

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

C++ 运行结果：

Java 算法代码：

```java
class Solution
{
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};

    public int[][] floodFill(int[][] image, int sr, int sc, int color)
    {
        int prev = image[sr][sc]; // 统计刚开始的颜色
        if(prev == color) return image; // 处理边界情况

        int m = image.length, n = image[0].length;

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{sr, sc});

        while(!q.isEmpty())
        {
            int[] t = q.poll();
            int a = t[0], b = t[1];
            image[a][b] = color;
            // 上下左右四个方向
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && image[x][y] == prev)
                {
                    q.add(new int[]{x, y});
                }
            }
        }
```

```
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：

```
class Solution
{
    int dx[4] = {1, -1, 0, 0};
    int dy[4] = {0, 0, 1, -1};
    bool vis[301][301];
    int m, n;

public:
    int numIslands(vector<vector<char>>& grid)
    {
        m = grid.size(), n = grid[0].size();
        int ret = 0;
        for(int i = 0; i < m; i++)
        {
            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：

Java 算法代码：

```java
class Solution
{
    int[] dx = {0, 0, -1, 1};
    int[] dy = {1, -1, 0, 0};
    boolean[][] vis = new boolean[301][301];
    int m, n;

    public int numIslands(char[][] grid)
    {
        m = grid.length; n = grid[0].length;

        int ret = 0;
        for(int i = 0; i < m; i++)
        {
            for(int j = 0; j < n; j++)
            {
                if(grid[i][j] == '1' && !vis[i][j])
                {
                    ret++;
                    bfs(grid, i, j);
                }
            }
        }
        return ret;
    }

    public void bfs(char[][] grid, int i, int j)
    {
        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{i, j});
        vis[i][j] = true;

        while(!q.isEmpty())
        {
            int[] t = q.poll();
            int a = t[0], b = t[1];
            for(int k = 0; k < 4; k++)
            {
                int x = a + dx[k], y = b + dy[k];
                if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == '1' &&
!vis[x][y])
                {
                    q.add(new int[]{x, y});
                    vis[x][y] = true;
                }
            }
```

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

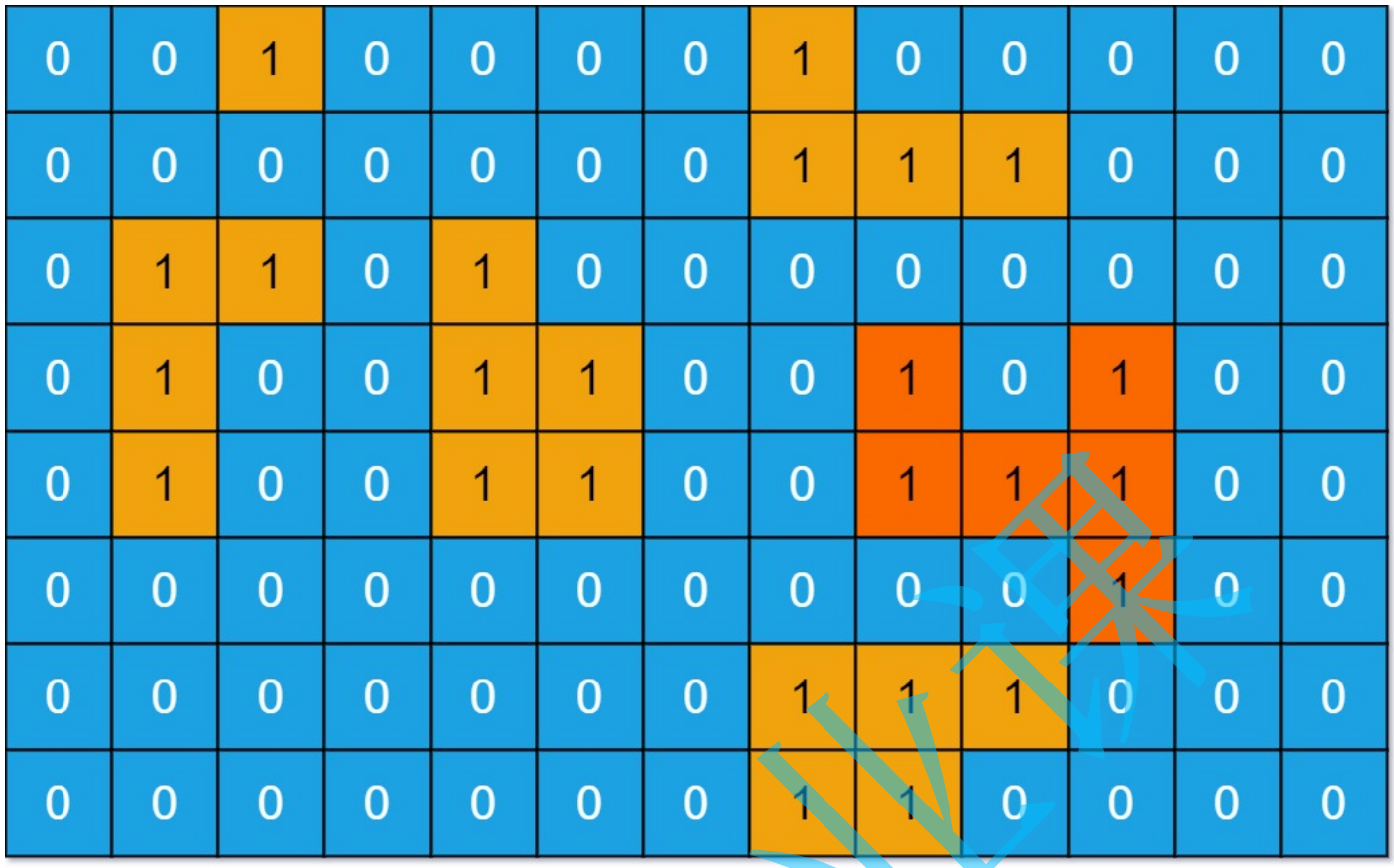
Java 运行结果：

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 x 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：



输入：

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

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：

```cpp
class Solution
{
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};
    bool vis[51][51];
    int m, n;

public:
    int maxAreaOfIsland(vector<vector<int>>& grid)
    {
        m = grid.size(), n = grid[0].size();

        int ret = 0;
        for(int i = 0; i < m; i++)
        {
            for(int j = 0; j < n; j++)
            {
                if(grid[i][j] == 1 && !vis[i][j])
                {
                    ret = max(ret, bfs(grid, i, j));
                }
            }
        }
        return ret;
    }

    int bfs(vector<vector<int>>& grid, int i, int j)
    {
        int count = 0;
        queue<pair<int, int>> q;
        q.push({i, j});
        vis[i][j] = true;
```

```
        count++;

        while(q.size())
        {
            auto [a, b] = q.front();
            q.pop();
            for(int k = 0; k < 4; k++)
            {
                int x = a + dx[k], y = b + dy[k];
                if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == 1 &&
!vis[x][y])
                {
                    q.push({x, y});
                    vis[x][y] = true;
                    count++;
                }
            }
        }
        return count;
    }
};
```

C++ 运行结果：

Java 算法代码：

```
class Solution
{
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};
    boolean[][] vis = new boolean[51][51];
    int m, n;

    public int maxAreaOfIsland(int[][] grid)
    {
        m = grid.length; n = grid[0].length;

        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：

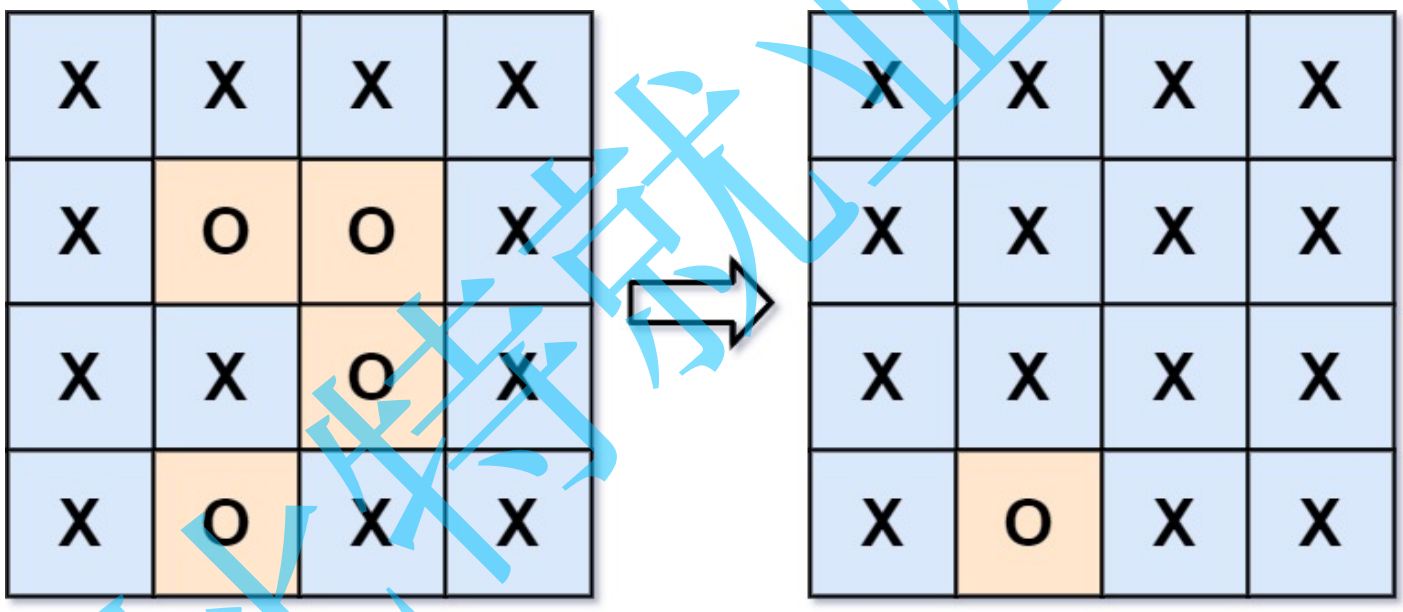
81. 被围绕的区域（medium）

1. **题目链接：130. 被围绕的区域**

2. **题目描述：**

给你一个 m x 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：

```
class Solution
{
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};
    int m, n;

public:
    void solve(vector<vector<char>>& board)
    {
        m = board.size(), n = board[0].size();

        // 1. 先处理边界上的 'O' 联通块，全部修改成 '.'
        for(int j = 0; j < n; j++)
        {
            if(board[0][j] == 'O') bfs(board, 0, j);
            if(board[m - 1][j] == 'O') bfs(board, m - 1, j);
        }

        for(int i = 0; i < m; i++)
        {
            if(board[i][0] == 'O') bfs(board, i, 0);
            if(board[i][n - 1] == 'O') bfs(board, i, n - 1);
        }

        // 2. 还原
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(board[i][j] == 'O') board[i][j] = 'X';
                else if(board[i][j] == '.') board[i][j] = 'O';
```

```
    }

    void bfs(vector<vector<char>>& board, int i, int j)
    {
        queue<pair<int, int>> q;
        q.push({i, j});
        board[i][j] = '.';

        while(q.size())
        {
            auto [a, b] = q.front();
            q.pop();
            for(int k = 0; k < 4; k++)
            {
                int x = a + dx[k], y = b + dy[k];
                if(x >= 0 && x < m && y >= 0 && y < n && board[x][y] == 'O')
                {
                    q.push({x, y});
                    board[x][y] = '.';
                }
            }
        }
    }
};
```

C++ 代码结果：

Java 算法代码：

```
class Solution
{
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};
    int m, n;

    public void solve(char[][] board)
    {
        m = board.length; n = board[0].length;
```

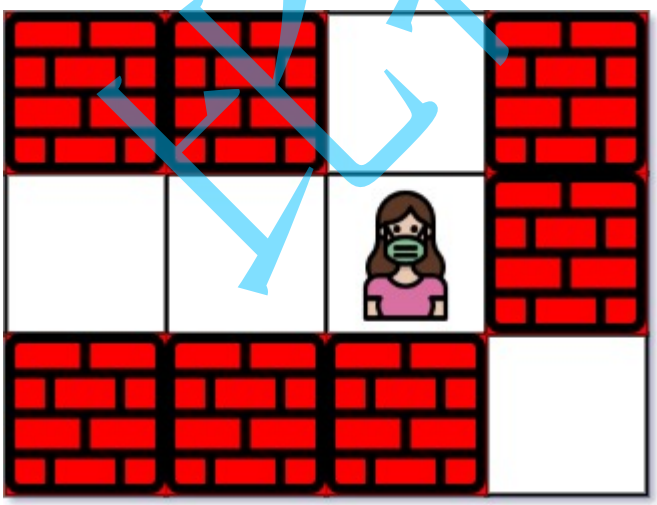
```java
        // 1. 先处理边界的 'O' 联通块，全部修改成 '.'
        for(int j = 0; j < n; j++)
        {
            if(board[0][j] == 'O') bfs(board, 0, j);
            if(board[m - 1][j] == 'O') bfs(board, m - 1, j);
        }

        for(int i = 0; i < m; i++)
        {
            if(board[i][0] == 'O') bfs(board, i, 0);
            if(board[i][n - 1] == 'O') bfs(board, i, n - 1);
        }

        // 2. 还原
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(board[i][j] == 'O') board[i][j] = 'X';
                else if(board[i][j] == '.') board[i][j] = 'O';
    }

    public void bfs(char[][] board, int i, int j)
    {
        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{i, j});
        board[i][j] = '.';

        while(!q.isEmpty())
        {
            int[] t = q.poll();
            int a = t[0], b = t[1];
            for(int k = 0; k < 4; k++)
            {
                int x = a + dx[k], y = b + dy[k];
                if(x >= 0 && x < m && y >= 0 && y < n && board[x][y] == 'O')
                {
                    board[x][y] = '.';
                    q.add(new int[]{x, y});
                }
            }
        }
    }
}
```

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 x 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) 。

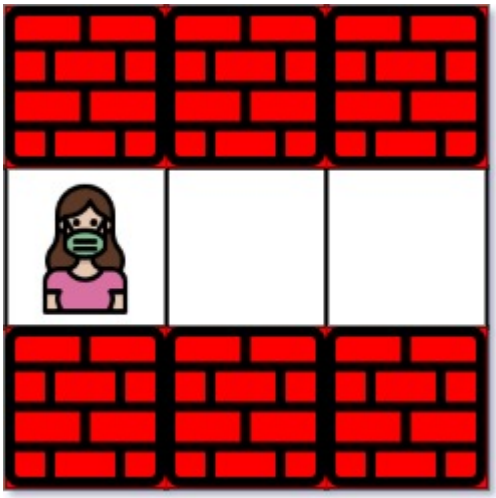
一开始，你在入口格子 (1,2) 处。

- 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。

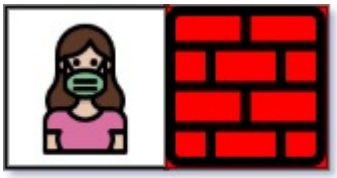
(1,0) 不算出口，因为它是入口格子。

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：

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

public:
    int nearestExit(vector<vector<char>>& maze, vector<int>& e)
    {
        int m = maze.size(), n = maze[0].size();

        bool vis[m][n];
        memset(vis, 0, sizeof vis);

        queue<pair<int, int>> q;
```

```
        q.push({e[0], e[1]});
        vis[e[0]][e[1]] = true;

        int step = 0;
        while(q.size())
        {
            step++;
            int sz = q.size();
            for(int i = 0; i < sz; i++)
            {
                auto [a, b] = q.front();
                q.pop();
                for(int j = 0; j < 4; j++)
                {
                    int x = a + dx[j], y = b + dy[j];
                    if(x >= 0 && x < m && y >= 0 && y < n && maze[x][y] == '.' && !vis[x][y])
                    {
                        // 判断是否已经到达出口
                        if(x == 0 || x == m - 1 || y == 0 || y == n - 1) return step;
                        q.push({x, y});
                        vis[x][y] = true;
                    }
                }
            }
        }

        return -1;
    }
};
```

C++ 运行结果：

Java 算法代码：

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

```java
    int[] dy = {1, -1, 0, 0};

    public int nearestExit(char[][] maze, int[] e)
    {
        int m = maze.length, n = maze[0].length;

        boolean[][] vis = new boolean[m][n];

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{e[0], e[1]});
        vis[e[0]][e[1]] = true;
        int step = 0;

        while(!q.isEmpty())
        {
            step++;
            int sz = q.size();
            for(int i = 0; i < sz; i++)
            {
                int[] t = q.poll();
                int a = t[0], b = t[1];
                for(int j = 0; j < 4; j++)
                {
                    int x = a + dx[j], y = b + dy[j];
                    if(x >= 0 && x < m && y >= 0 && y < n && maze[x][y] == '.' && !vis[x][y])
                    {
                        // 判断是否已经到出口
                        if(x == 0 || x == m - 1 || y == 0 || y == n - 1) return step;
                        vis[x][y] = true;
                        q.add(new int[]{x, y});
                    }
                }
            }
        }
        return -1;
    }
}
```

Java 运行结果：

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：

```
class Solution
{
public:
    int minMutation(string startGene, string endGene, vector<string>& bank)
    {
        unordered_set<string> vis; // 用来标记已经搜索过的状态
        unordered_set<string> hash(bank.begin(), bank.end()); // 存储基因库里面的字符串
        string change = "ACGT";

        if(startGene == endGene) return 0;
        if(!hash.count(endGene)) return -1;

        queue<string> q;
        q.push(startGene);
        vis.insert(startGene);

        int ret = 0;
        while(q.size())
        {
            ret++;
            int sz = q.size();
            while(sz--)
            {
                string t = q.front();
                q.pop();
                for(int i = 0; i < 8; i++)
                {
                    string tmp = t; // 细节问题
                    for(int j = 0; j < 4; j++)
                    {
                        tmp[i] = change[j];
                        if(hash.count(tmp) && !vis.count(tmp))
                        {
```

```
                            if(tmp == endGene) return ret;
                            q.push(tmp);
                            vis.insert(tmp);
                        }
                    }
                }
            }
        }
        return -1;
    }
};
```

C++ 代码结果：

Java 算法代码：

```java
class Solution
{
    public int minMutation(String startGene, String endGene, String[] bank)
    {
        Set<String> vis = new HashSet<>(); // 用来标记已经搜索过的状态
        Set<String> hash = new HashSet<>(); // 用来统计基因库里面的字符串
        for(String s : bank) hash.add(s);
        char[] change = {'A', 'C', 'G', 'T'};

        if(startGene.equals(endGene)) return 0;
        if(!hash.contains(endGene)) return -1;

        Queue<String> q = new LinkedList<>();
        q.add(startGene);
        vis.add(startGene);
        int step = 0;

        while(!q.isEmpty())
        {
            step++;
            int sz = q.size();
            while(sz-- != 0)
            {
```

```
                String t = q.poll();
                for(int i = 0; i < 8; i++)
                {
                    char[] tmp = t.toCharArray();
                    for(int j = 0; j < 4; j++)
                    {
                        tmp[i] = change[j];
                        String next = new String(tmp);
                        if(hash.contains(next) && !vis.contains(next))
                        {
                            if(next.equals(endGene)) return step;
                            q.add(next);
                            vis.add(next);
                        }
                    }
                }
            }
        }
        return -1;
    }
}
```

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 <= i <= 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：

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

```

```
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：

Java 算法代码：

```java
class Solution
{
    public int ladderLength(String beginWord, String endWord, List<String> wordList)
    {
        Set<String> hash = new HashSet<>();
        for(String s : wordList) hash.add(s);
        Set<String> vis = new HashSet<>(); // 标记已经搜索过的单词

        if(!hash.contains(endWord)) return 0;

        Queue<String> q = new LinkedList<>();
        q.add(beginWord);
        vis.add(beginWord);

        int ret = 1;
        while(!q.isEmpty())
        {
            ret++;
            int sz = q.size();
            while(sz-- != 0)
            {
                String t = q.poll();
                for(int i = 0; i < t.length(); i++)
                {
                    char[] tmp = t.toCharArray();
                    for(char ch = 'a'; ch <= 'z'; ch++)
                    {
                        tmp[i] = ch;
                        String next = new String(tmp);
                        if(hash.contains(next) && !vis.contains(next))
                        {
                            if(next.equals(endWord)) return ret;
                            q.add(next);
                            vis.add(next);
                        }
                    }
                }
            }
        }
        return 0;
    }
}
```


Java 运行结果：

Java

时间 72 ms 击败 63.19% 内存 45.7 MB 击败 31.59%

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 x 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

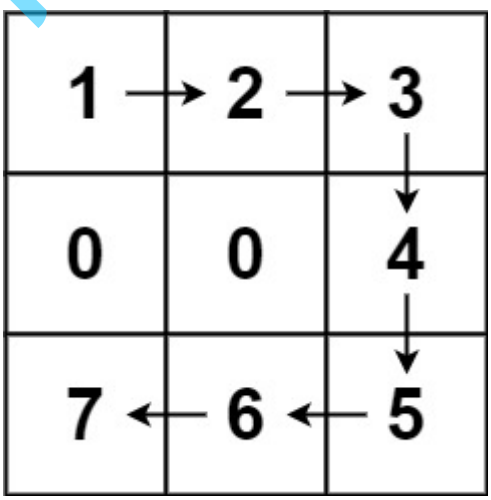
每一步，你都可以向上、下、左、右四个方向之一移动一个单位，如果你站的地方有一棵树，那么你可以决定是否要砍倒它。

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：

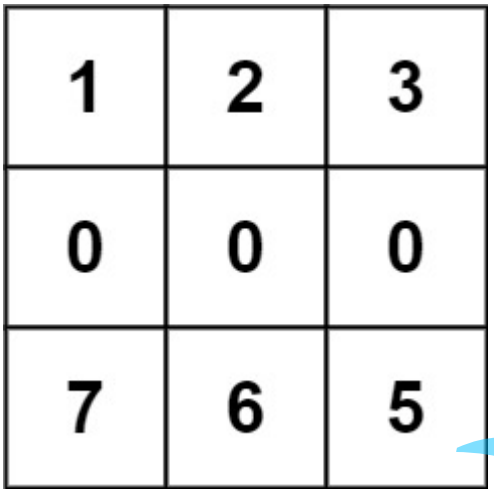


输入： forest = [[1,2,3],[0,0,4],[7,6,5]]

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：



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：

a. 先找出砍树的顺序；

b. 然后按照砍树的顺序，一个一个的用 `bfs` 求出最短路即可。

C++ 算法代码：

```cpp
class Solution
{
    int m, n;
public:
    int cutOffTree(vector<vector<int>>& f)
    {
        m = f.size(), n = f[0].size();
        // 1. 准备工作：找出砍树的顺序
        vector<pair<int, int>> trees;
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(f[i][j] > 1) trees.push_back({i, j});

        sort(trees.begin(), trees.end(), [&](const pair<int, int>& p1, const pair<int, int>& p2)
        {
            return f[p1.first][p1.second] < f[p2.first][p2.second];
        });

        // 2. 按照顺序砍树
        int bx = 0, by = 0;
        int ret = 0;
        for(auto& [a, b] : trees)
        {
            int step = bfs(f, bx, by, a, b);
            if(step == -1) return -1;
            ret += step;
            bx = a, by = b;
        }
        return ret;
    }

    bool vis[51][51];
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};

    int bfs(vector<vector<int>>& f, int bx, int by, int ex, int ey)
    {
        if(bx == ex && by == ey) return 0;

        queue<pair<int, int>> q;
        memset(vis, 0, sizeof vis); // 清空之前的数据
        q.push({bx, by});
        vis[bx][by] = true;

        int step = 0;
        while(q.size())
```

```
47          {
48              step++;
49              int sz = q.size();
50              while(sz--)
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：

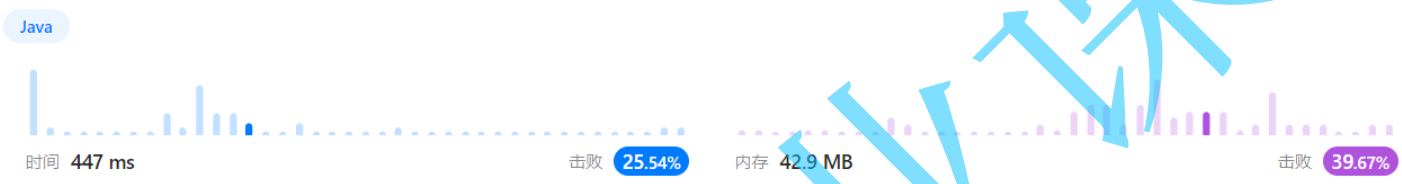
Java 算法代码：

```
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)
    != 0 && !vis[x][y])
56                      {
57                          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：

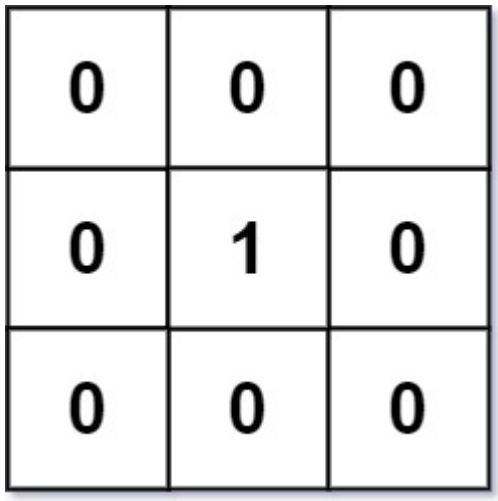


多源 BFS

86. 01 矩阵（medium）

1. **题目链接：542. 01 矩阵**
2. **题目描述：**

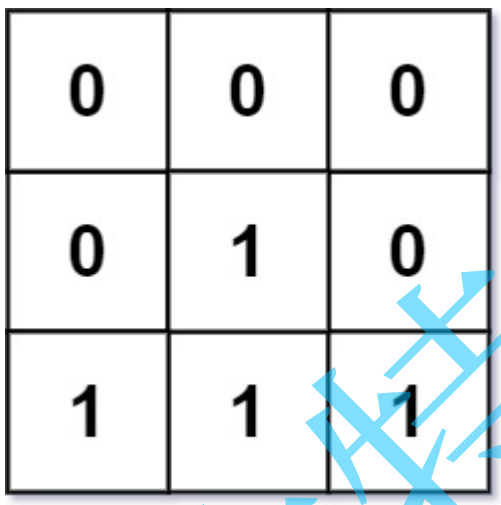
> 给定一个由 0 和 1 组成的矩阵 mat ，请输出一个大小相同的矩阵，其中每一个格子是 mat 中对应位置元素到最近的 0 的距离。
>
> 两个相邻元素间的距离为 1 。
>
> 示例 1：



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：



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，我们怎么才能保证遇到的 `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：

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

public:
    vector<vector<int>> updateMatrix(vector<vector<int>>& mat)
    {
        int m = mat.size(), n = mat[0].size();

        // dist[i][j] == -1 表示：没有搜索过
        // dist[i][j] != -1 表示：最短距离
        vector<vector<int>> dist(m, vector<int>(n, -1));
        queue<pair<int, int>> q;

        // 1. 把所有的源点加入到队列中
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(mat[i][j] == 0)
                {
                    q.push({i, j});
```

```
                    dist[i][j] = 0;
                }

        // 2. 一层一层的往外扩
        while(q.size())
        {
            auto [a, b] = q.front(); q.pop();
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
                {
                    dist[x][y] = dist[a][b] + 1;
                    q.push({x, y});
                }
            }
        }
        return dist;
    }
};
```

C++ 运行结果：

Java 算法代码：

```
class Solution
{
    int[] dx = {0, 0, -1, 1};
    int[] dy = {1, -1, 0, 0};

    public int[][] updateMatrix(int[][] mat)
    {
        int m = mat.length, n = mat[0].length;
        // dist[i][j] == -1：标记当前位置没有被搜索过
        // dist[i][j] != -1：存的是最短距离
        int[][] dist = new int[m][n];
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
```

```
                dist[i][j] = -1;
        Queue<int[]> q = new LinkedList<>();

        // 1. 把所有的源点加入到队列里面
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
            {
                if(mat[i][j] == 0)
                {
                    dist[i][j] = 0;
                    q.add(new int[]{i, j});
                }
            }

        // 2. 一层一层的往外扩
        while(!q.isEmpty())
        {
            int[] t = q.poll();
            int a = t[0], b = t[1];
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
                {
                    dist[x][y] = dist[a][b] + 1;
                    q.add(new int[]{x, y});
                }
            }
        }
        return dist;
    }
}
```

Java 运行结果：

87. 飞地的数量（medium）

1. **题目链接：1020. 飞地的数量**

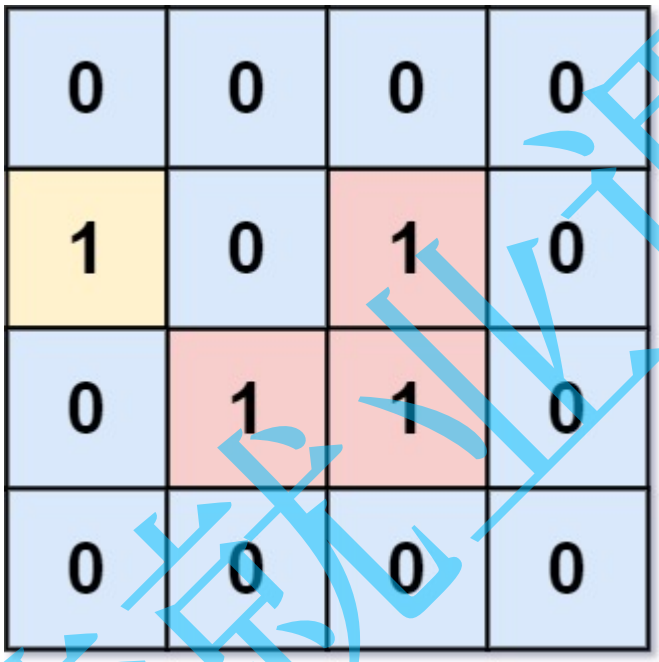
2. **题目描述：**

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。

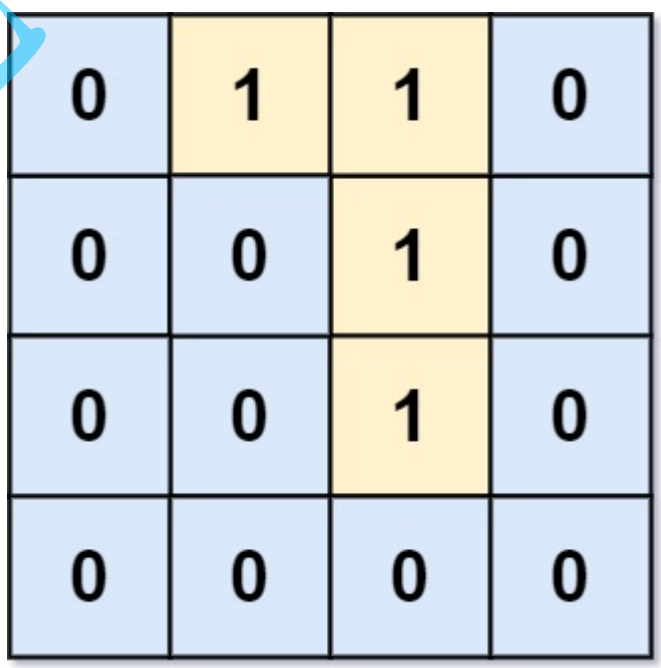
返回网格中 **无法** 在任意次数的移动中离开网格边界的陆地单元格的数量。

示例 1：



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：



> **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：

```cpp
class Solution
{
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};

public:
    int numEnclaves(vector<vector<int>>& grid)
    {
        int m = grid.size(), n = grid[0].size();
        vector<vector<bool>> vis(m, vector<bool>(n));
        queue<pair<int, int>> q;

        // 1. 把边上的 1 加入到队列中
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(i == 0 || i == m - 1 || j == 0 || j == n - 1)
                {
                    if(grid[i][j] == 1)
                    {
                        q.push({i, j});
```

```
                    vis[i][j] = true;
                }
            }
        // 2. 多源 bfs
        while(q.size())
        {
            auto [a, b] = q.front();
            q.pop();
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == 1 &&
!vis[x][y])
                {
                    vis[x][y] = true;
                    q.push({x, y});
                }
            }
        }
        // 3. 统计结果
        int ret = 0;
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(grid[i][j] == 1 && !vis[i][j])
                    ret++;
        return ret;
    }
};
```

C++ 代码结果：

Java 算法代码：

```
class Solution
{
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};

```

```java
    public int numEnclaves(int[][] grid)
    {
        int m = grid.length, n = grid[0].length;
        boolean[][] vis = new boolean[m][n];
        Queue<int[]> q = new LinkedList<>();

        // 1. 把边上的 1 全部加入到队列中
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(i == 0 || i == m - 1 || j == 0 || j == n - 1)
                {
                    if(grid[i][j] == 1)
                    {
                        q.add(new int[]{i, j});
                        vis[i][j] = true;
                    }
                }
        // 2. 多源 bfs
        while(!q.isEmpty())
        {
            int[] t = q.poll();
            int a = t[0], b = t[1];
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == 1 &&
!vis[x][y])
                {
                    q.add(new int[]{x, y});
                    vis[x][y] = true;
                }
            }
        }
        // 3. 提取结果
        int ret = 0;
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(grid[i][j] == 1 && !vis[i][j])
                    ret++;
        return ret;
    }
}
```

Java 运行结果：

Java

时间 8 ms　击败 61.70%　内存 53.5 MB　击败 30.28%

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 x 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。

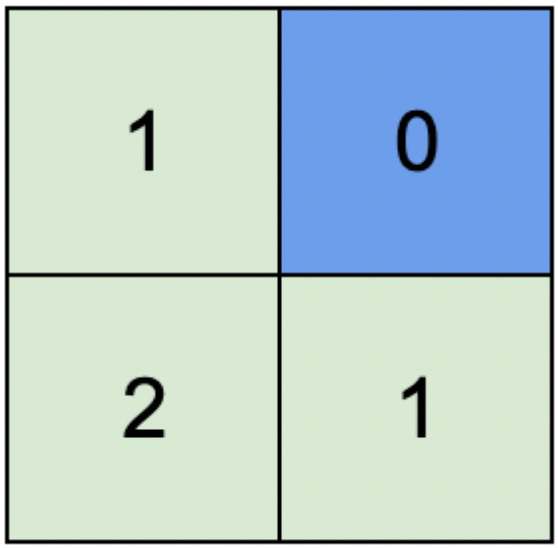
你需要按照如下规则给每个单元格安排高度：

- 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** 。

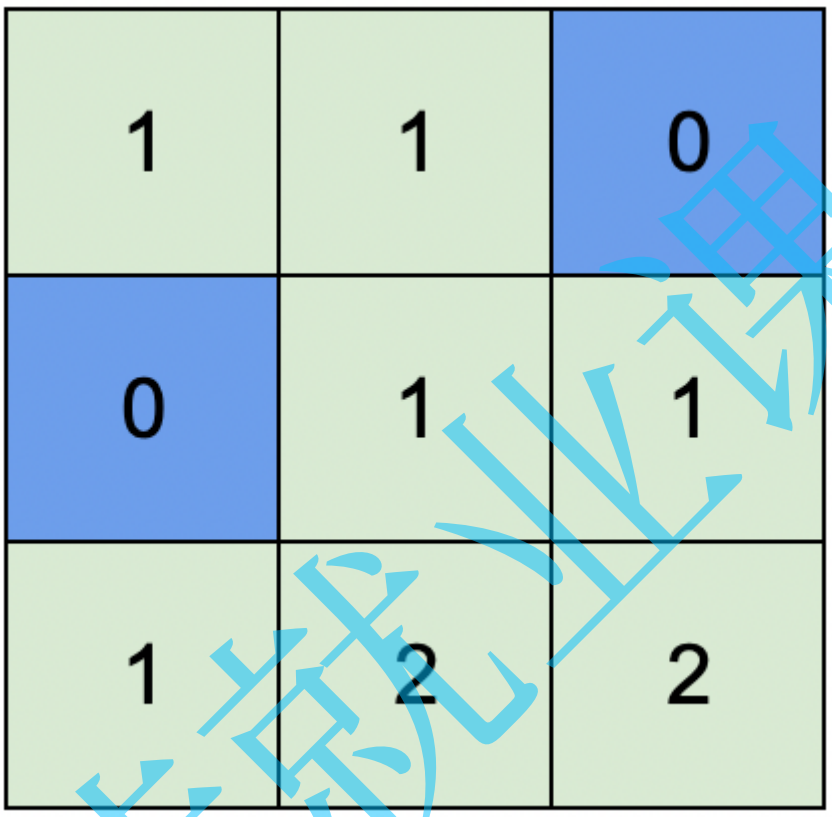
请你返回一个大小为 `m x n` 的整数矩阵 `height` ，其中 `height[i][j]` 是格子 `(i, j)` 的高度。如果有多种解法，请返回 **任意一个** 。

示例 1：



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：



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：

```cpp
class Solution
{
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};

public:
    vector<vector<int>> highestPeak(vector<vector<int>>& isWater)
    {
        int m = isWater.size(), n = isWater[0].size();
        vector<vector<int>> dist(m, vector<int>(n, -1));
        queue<pair<int, int>> q;

        // 1. 把所有的源点加入到队列里面
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(isWater[i][j])
                {
                    dist[i][j] = 0;
                    q.push({i, j});
                }
        // 2. 多源 bfs
        while(q.size())
        {
            auto [a, b] = q.front(); q.pop();
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
                {
                    dist[x][y] = dist[a][b] + 1;
                    q.push({x, y});
                }
            }
        }
        return dist;
    }
};
```

C++ 代码结果：

Java 算法代码：

```java
class Solution
{
    int[] dx = {0, 0, -1, 1};
    int[] dy = {1, -1, 0, 0};

    public int[][] highestPeak(int[][] isWater)
    {
        int m = isWater.length, n = isWater[0].length;
        int[][] dist = new int[m][n];
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                dist[i][j] = -1;
        Queue<int[]> q = new LinkedList<>();

        // 1. 所有的源点加入到队列里面
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(isWater[i][j] == 1)
                {
                    q.add(new int[]{i, j});
                    dist[i][j] = 0;
                }
        // 2. 多源 bfs
        while(!q.isEmpty())
        {
            int[] t = q.poll();
            int a = t[0], b = t[1];
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
                {
                    dist[x][y] = dist[a][b] + 1;
                    q.add(new int[]{x, y});
                }
            }
        }
```

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

Java 运行结果：

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 x 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）：`(x0, y0)` 和 `(x1, y1)` 这两个单元格之间的距离是 `|x0 - x1| + |y0 - y1|`。

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：

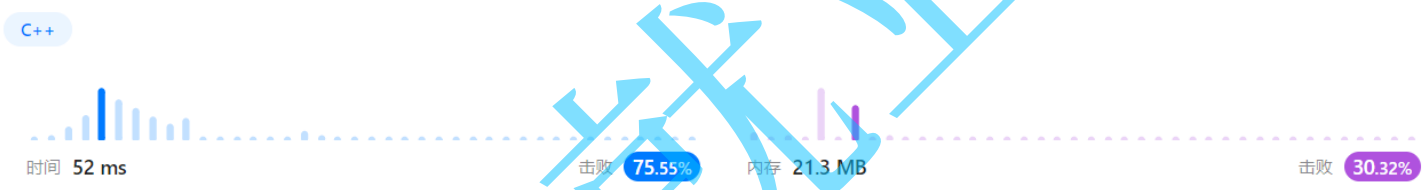
01矩阵的变型题，直接用多源 bfs 解决即可。

C++ 算法代码：

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

public:
    int maxDistance(vector<vector<int>>& grid)
    {
        int m = grid.size(), n = grid[0].size();
        vector<vector<int>> dist(m, vector<int>(n, -1));
        queue<pair<int, int>> q;

        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(grid[i][j])
                {
                    dist[i][j] = 0;
                    q.push({i, j});
                }

        int ret = -1;
        while(q.size())
```

```
        {
            auto [a, b] = q.front(); q.pop();
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < m && y >= 0 && y < n && dist[x][y] == -1)
                {
                    dist[x][y] = dist[a][b] + 1;
                    q.push({x, y});
                    ret = max(ret, dist[x][y]);
                }
            }
        }
        return ret;
    }
};
```

C++ 代码结果：



Java 算法代码：

```
class Solution
{
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};

    public int maxDistance(int[][] grid)
    {
        int m = grid.length, n = grid[0].length;
        int[][] dist = new int[m][n];
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                dist[i][j] = -1;
        Queue<int[]> q = new LinkedList<>();

        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                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：

a. 将所有入度为 0 的点加入到队列中；

b. 当队列不空的时候，一直循环：

i. 取出队头元素；

ii. 将于队头元素相连的顶点的入度 - 1；

iii. 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：

```cpp
class Solution
{
public:
    bool canFinish(int n, vector<vector<int>>& p)
    {
        unordered_map<int, vector<int>> edges; // 邻接表
        vector<int> in(n); // 存储每一个结点的入度

        // 1. 建图
        for(auto& e : p)
        {
            int a = e[0], b = e[1];
            edges[b].push_back(a);
            in[a]++;
        }

        // 2. 拓扑排序 - bfs
        queue<int> q;
        // 把所有入度为 0 的点加入到队列中
        for(int i = 0; i < n; i++)
        {
            if(in[i] == 0) q.push(i);
        }

        // 层序遍历
        while(!q.empty())
        {
            int t = q.front();
            q.pop();
            // 修改相连的边
            for(int e : edges[t])
            {
                in[e]--;
                if(in[e] == 0) q.push(e);
            }
        }

        // 3. 判断是否有环
        for(int i : in)
            if(i)
                return false;

        return true;
```

```
    }
};
```

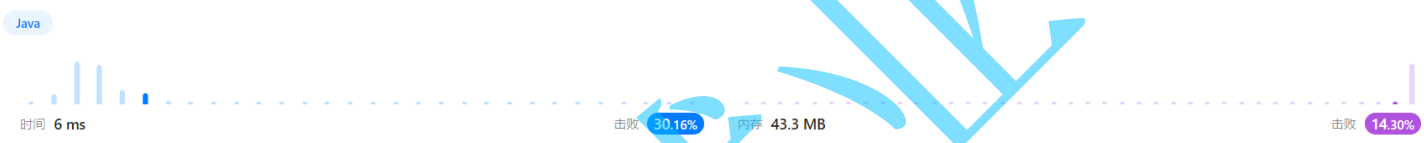
C++ 代码结果：

Java 算法代码：

```java
class Solution
{
    public boolean canFinish(int n, int[][] p)
    {
        // 1. 准备工作
        int[] in = new int[n]; // 统计每一个顶点的入度
        Map<Integer, List<Integer>> edges = new HashMap<>(); // 邻接表存图

        // 2. 建图
        for(int i = 0; i < p.length; i++)
        {
            int a = p[i][0], b = p[i][1]; // b -> a
            if(!edges.containsKey(b))
            {
                edges.put(b, new ArrayList<>());
            }
            edges.get(b).add(a);
            in[a]++;
        }

        // 3. 拓扑排序
        Queue<Integer> q = new LinkedList<>();
        // (1) 先把入度为 0 的点，加入到队列中
        for(int i = 0; i < n; i++)
        {
            if(in[i] == 0) q.add(i);
        }

        // (2) bfs
        while(!q.isEmpty())
        {
            int t = q.poll();

```

```
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：

```
class Solution
{
public:
    vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites)
    {
        // 1. 准备工作
        vector<vector<int>> edges(numCourses); // 邻接表存储图
        vector<int> in(numCourses); // 存储每一个点的入度

        // 2. 建图
        for(auto& p : prerequisites)
        {
            int a = p[0], b = p[1]; // b -> a
            edges[b].push_back(a);
            in[a]++;
        }

```

```
        // 3. 拓扑排序
        vector<int> ret; // 统计排序后的结果
        queue<int> q;
        for(int i = 0; i < numCourses; i++)
        {
            if(in[i] == 0) q.push(i);
        }
        while(q.size())
        {
            int t = q.front(); q.pop();
            ret.push_back(t);
            for(int a : edges[t])
            {
                in[a]--;
                if(in[a] == 0) q.push(a);
            }
        }
        if(ret.size() == numCourses) return ret;
        else return {};
    }
};
```

C++ 代码结果：

Java 算法代码：

```
class Solution
{
    public int[] findOrder(int n, int[][] prerequisites)
    {
        // 1. 准备工作
        int[] in = new int[n]; // 统计每个顶点的入度
        List<List<Integer>> edges = new ArrayList<>();
        for(int i = 0; i < n; i++)
        {
            edges.add(new ArrayList<>());
        }

        // 2. 建图
        for(int i = 0; i < prerequisites.length; i++)
```

```java
        {
            int a = prerequisites[i][0], b = prerequisites[i][1]; // b -> a
            edges.get(b).add(a);
            in[a]++;
        }

        // 3. 拓扑排序
        Queue<Integer> q = new LinkedList<>();
        int[] ret = new int[n];
        int index = 0;
        for(int i = 0; i < n; i++)
        {
            if(in[i] == 0) q.add(i);
        }

        while(!q.isEmpty())
        {
            int t = q.poll();
            ret[index++] = t;
            for(int a : edges.get(t))
            {
                in[a]--;
                if(in[a] == 0) q.add(a);
            }
        }

        if(index == n) return ret;
        return new int[0];
    }
}
```

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）：

a. 两层 for 循环枚举出所有的两个字符串的组合；

b. 然后利用指针，根据字典序规则找出信息。

C++ 算法代码：

```cpp
class Solution
{
    unordered_map<char, unordered_set<char>> edges; // 邻接表来存储图
    unordered_map<char, int> in; // 统计入度
    bool cheak; // 处理边界情况

public:
    string alienOrder(vector<string>& words)
    {
        // 1. 建图 + 初始化入度哈希表
        for(auto& s : words)
        {
            for(auto ch : s)
            {
                in[ch] = 0;
            }
        }
        int n = words.size();
        for(int i = 0; i < n; i++)
        {
            for(int j = i + 1; j < n; j++)
            {
                add(words[i], words[j]);
                if(cheak) return "";
            }
        }

        // 2. 拓扑排序
        queue<char> q;
        for(auto& [a, b] : in)
        {
            if(b == 0) q.push(a);
        }
        string ret;
        while(q.size())
        {
            char t = q.front(); q.pop();
            ret += t;
            for(char ch : edges[t])
            {
                if(--in[ch] == 0) q.push(ch);
            }
        }
```

```
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)
```

```
    {
        // 1. 初始化入度哈希表 + 建图
        for(String s : words)
        {
            for(int i = 0; i < s.length(); i++)
            {
                char ch = s.charAt(i);
                in.put(ch, 0);
            }
        }
        int n = words.length;
        for(int i = 0; i < n; i++)
        {
            for(int j = i + 1; j < n; j++)
            {
                add(words[i], words[j]);
                if(check == true) return "";
            }
        }

        // 2. 拓扑排序
        Queue<Character> q = new LinkedList<>();
        for(char ch : in.keySet())
        {
            if(in.get(ch) == 0) q.add(ch);
        }
        StringBuffer ret = new StringBuffer();
        while(!q.isEmpty())
        {
            char t = q.poll();
            ret.append(t);

            if(!edges.containsKey(t)) continue;

            for(char ch : edges.get(t))
            {
                in.put(ch, in.get(ch) - 1);
                if(in.get(ch) == 0) q.add(ch);
            }
        }

        // 3. 判断
        for(char ch : in.keySet())
        {
            if(in.get(ch) != 0) return "";
        }
        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：