

## 动态规划：44. 通配符匹配

```
1  class Solution {
2  public:
3      bool isMatch(string s, string p) {
4          int m = s.size();
5          int n = p.size();
6          s = " " + s;
7          p = " " + p;
8          vector<vector<bool>> dp(m + 1, vector<bool>(n + 1));
9          dp[0][0] = true;
10
11         for (int j = 1; j <= n; j++) {
12             if (p[j] == '*') {
13                 dp[0][j] = dp[0][j - 1]; // * 可以匹配空字符串
14             }
15         }
16
17         for (int i = 1; i <= m; i++) {
18             for (int j = 1; j <= n; j++) {
19                 if (p[j] == '?')
20                     dp[i][j] = dp[i - 1][j - 1];
21                 else if (p[j] == '*') {
22                     dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
23                 } else {
24                     if (s[i] == p[j] && dp[i - 1][j - 1]) {
25                         dp[i][j] = true;
26                     }
27                 }
28             }
29         }
30         return dp[m][n];
31     }
32 };
```

**bfs:** 130. 被围绕的区域

```

1  class Solution {
2      int dx[4] = {1, -1, 0, 0};
3      int dy[4] = {0, 0, 1, -1};
4      int m = 0;
5      int n = 0;
6
7  public:
8      void bfs(vector<vector<char>>& board, int i, int j) {
9          queue<pair<int, int>> q;
10         q.push({i, j});
11         board[i][j] = '.'; // 标记已访问
12         while (!q.empty()) {
13             auto [a, b] = q.front();
14             q.pop();
15
16             for (int k = 0; k < 4; k++) {
17                 int x = a + dx[k]; // 更新应基于 a, b
18                 int y = b + dy[k]; // 更新应基于 a, b
19                 if (x >= 0 && x < m && y >= 0 && y < n && board[x][y] == '0') {
20                     q.push({x, y});
21                     board[x][y] = '.'; // 标记已访问
22                 }
23             }
24         }
25     }
26
27     void solve(vector<vector<char>>& board) {
28         if (board.empty()) return; // 边界检查
29
30         m = board.size(); // 行数
31         n = board[0].size(); // 列数
32
33         // 处理第一列和最后一列
34         for (int i = 0; i < m; i++) {
35             if (board[i][0] == '0') {
36                 bfs(board, i, 0);
37             }
38             if (board[i][n - 1] == '0') {
39                 bfs(board, i, n - 1);

```

```
40     }
41 }
42
43 // 处理第一行和最后一行
44 for (int j = 0; j < n; j++) {
45     if (board[0][j] == 'O') {
46         bfs(board, 0, j);
47     }
48     if (board[m - 1][j] == 'O') {
49         bfs(board, m - 1, j);
50     }
51 }
52
53 // 遍历整个 board，更新结果
54 for (int i = 0; i < m; i++) {
55     for (int j = 0; j < n; j++) {
56         if (board[i][j] == '.') {
57             board[i][j] = 'O'; // 从边界联通的 O
58         } else if (board[i][j] == 'O') {
59             board[i][j] = 'X'; // 被包围的 O
60         }
61     }
62 }
63 }
64 };
```