动态规划：

```cpp
class Solution {
public:
    bool isInterleave(string s1, string s2, string s3) {
        int m = s1.size();
        int n = s2.size();
        int l = s3.size();

        // 如果 s1 和 s2 的长度和不等于 s3 的长度，直接返回 false
        if (m + n != l) {
            return false;
        }

        // 动态规划表，dp[i][j] 表示 s1 的前 i 个字符和 s2 的前 j
        // 个字符是否可以组成 s3 的前 i+j 个字符
        vector<vector<bool>> dp(m + 1, vector<bool>(n + 1, false));

        // 初始化 dp 表
        dp[0][0] = true;

        // 初始化 s1 的前缀能否匹配 s3 的前缀
        for (int i = 1; i <= m; i++) {
            dp[i][0] = dp[i - 1][0] && (s1[i - 1] == s3[i - 1]);
        }

        // 初始化 s2 的前缀能否匹配 s3 的前缀
        for (int j = 1; j <= n; j++) {
            dp[0][j] = dp[0][j - 1] && (s2[j - 1] == s3[j - 1]);
        }

        // 填充 dp 表
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                dp[i][j] = (dp[i - 1][j] && s1[i - 1] == s3[i + j - 1]) ||
                           (dp[i][j - 1] && s2[j - 1] == s3[i + j - 1]);
            }
        }

        // 返回 dp 表的最后一个值，表示 s1 和 s2 是否能交错组成 s3
        return dp[m][n];
```

```
40         }
41     };
```

BFS：1926. 迷宫中离入口最近的出口

```cpp
class Solution {
    int m, n;
    int dx[4] = {1, -1, 0, 0};
    int dy[4] = {0, 0, 1, -1};
    bool vis[101][101];

public:
    int nearestExit(vector<vector<char>>& maze, vector<int>& e) {
        m = maze.size();
        n = maze[0].size();
        int row = e[0];
        int col = e[1];

        queue<pair<int, int>> q;
        q.push({row, col});
        int step = 0;
        vis[row][col] = true;
        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];
                    int y = b + dy[j];
                    if (x >= 0 && x < m && y >= 0 && y < n &&
                        maze[x][y] == '.' && vis[x][y] == false) {
                        if (x == 0 || x == m - 1 || y == 0 || y == n - 1) {
                            return step;
                        }
                        q.push({x,y});
                        vis[x][y] = true;
                    }
                }
            }
        }
        return -1;
    }
```

```
40 };
```