两个数组dp问题 :1143. 最长公共子序列

解题思路:

我们使用两个示例字符串：

- text1 = " ABCD"
- text2 = " ACBD"

1. **初始化 DP 表格**：
我们首先初始化一个大小为 (m+1) x (n+1) 的动态规划 (DP) 表格，其中 m 和 n 分别是两个字符串的长度。表格用于存储两个字符串的不同前缀之间的最长公共子序列的长度。
   - 表格的第一行和第一列代表空字符串。
   - 每个单元格 dp[i][j] 表示 text1 的前 i 个字符和 text2 的前 j 个字符之间的最长公共子序列长度。
2. **初始状态**：
所有 dp[0][j] 和 dp[i][0] 都初始化为 0，因为与空字符串的 LCS 长度是 0。表格最开始的样子是：

		A	C	B	D
	0	0	0	0	0
A	0				
B	0				
C	0				
D	0				

3. **遍历并填充表格**：
接下来，我们使用嵌套循环来遍历每个字符。对于每个 i 和 j，我们根据状态转移方程更新 dp[i][j]：
   - 如果 text1[i] == text2[j]，那么 dp[i][j] = dp[i-1][j-1] + 1。这表示当两个字符相同时，可以将它们纳入公共子序列。
   - 否则，dp[i][j] = max(dp[i-1][j], dp[i][j-1])。这表示我们不将当前字符纳入子序列，取前一个状态的最大值。
4. **状态转移示例**：
假设 text1 = " ABCD" 和 text2 = " ACBD"，按步骤更新表格：
   - 第一步，比较 text1[1] = 'A' 和 text2[1] = 'A'，它们相等，所以 dp[1][1] = dp[0][0] + 1 = 1。

		A	C	B	D
	0	0	0	0	0
A	0	1			
B	0				
C	0				
D	0				

   - 第二步，比较 text1[1] = 'A' 和 text2[2] = 'C'，它们不相等，所以 dp[1][2] = max(dp[0][2], dp[1][1]) = 1。
   - 继续按照此规则填充表格：

最终的表格会是这样：

		A	C	B	D
	0	0	0	0	0
A	0	1	1	1	1
B	0	1	1	2	2
C	0	1	2	2	2
D	0	1	2	2	3

代码:

```cpp
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
        int m = text1.size();
        int n = text2.size();
        text1 = " " + text1;
        text2 = " " + text2;

        vector<vector<int>> dp(m + 1,vector<int>(n + 1));
        for(int i = 1; i <=m ;i++)
        {
            for(int j = 1;j <=n ;j++)
            {
                if(text1[i] == text2[j]) dp[i][j] = dp[i-1][j-1] + 1;
                else dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
            }
        }
        return dp[m][n];
    }
};
```

BFS： 200. 岛屿数量

解题思路：

代码：

```cpp
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++) {
                if (grid[i][j] == '1' && !vis[i][j]) {
                    ret++;
                    bfs(grid, i, j);
                }
            }
        }
        return ret;
    }
    void bfs(vector<vector<char>>& grid, int i, int j) {
        queue<pair<int, int>> q;
        q.push({i, j});
        vis[i][j] = true;
        while (q.size()) {
            auto [a, b] = q.front();
            q.pop();
            for (int k = 0; k < 4; k++) {
                int x = a + dx[k];
                int 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;
                }
            }
        }
    }
```

```
40 };
```