通配符匹配

给定一个字符串 (s) 和一个字符模式 (p) ，实现一个支持 '?' 和 '*' 的通配符匹配。

'?' 可以匹配任何单个字符。

'*' 可以匹配任意字符串（包括空字符串）。

两个字符串**完全匹配**才算匹配成功。

说明:

- s 可能为空，且只包含从 a-z 的小写字母。
- p 可能为空，且只包含从 a-z 的小写字母，以及字符 ? 和 *。

示例 1:

输入:

```
s = "aa"

p = "a"
```

输出: false

解释: "a" 无法匹配 "aa" 整个字符串。

示例 2:

输入:

```
s = "aa"

p = "*"
```

输出: true

解释: '*' 可以匹配任意字符串。

示例 3:

输入:

```
s = "cb"

p = "?a"
```

输出: false

解释: '?' 可以匹配 'c', 但第二个 'a' 无法匹配 'b'。

示例 4:

输入:

```
s = "adceb"

p = "*a*b"
```

输出: `true`

解释: 第一个 `'*'` 可以匹配空字符串, 第二个 `'*'` 可以匹配字符串 `"dce"`.

示例 5:

输入:

```
s = "acdcb"

p = "a*c?b"
```

输出: `false`

我们用 $dp[i][j]$ 表示字符串 s 的前 i 个字符和模式 p 的前 j 个字符是否能匹配。在进行状态转移时，我们可以考虑模式 p 的第 j 个字符 p_j，与之对应的是字符串 s 中的第 i 个字符 s_i：

- 如果 p_j 是小写字母，那么 s_i 必须也为相同的小写字母，状态转移方程为：

$$dp[i][j] = (s_i \text{ 与 } p_j \text{ 相同}) \wedge dp[i-1][j-1]$$

  其中 $\wedge$ 表示逻辑与运算。也就是说，$dp[i][j]$ 为真，当且仅当 $dp[i-1][j-1]$ 为真，并且 s_i 与 p_j 相同。
- 如果 p_j 是问号，那么对 s_i 没有任何要求，状态转移方程为：

$$dp[i][j] = dp[i-1][j-1]$$

- 如果 p_j 是星号，那么同样对 s_i 没有任何要求，但是星号可以匹配零或任意多个小写字母，因此状态转移方程分为两种情况，即使用或不使用这个星号：

$$dp[i][j] = dp[i][j-1] \vee dp[i-1][j]$$

其中 $\vee$ 表示逻辑或运算。如果我们不使用这个星号，那么就会从 $dp[i][j-1]$ 转移而来；如果我们使用这个星号，那么就会从 $dp[i-1][j]$ 转移而来。

最终的状态转移方程如下：

$$dp[i][j] = \begin{cases} (s_i \text{ 与 } p_j \text{ 相同}) \wedge dp[i-1][j-1], & p_j \text{ 是小写字母} \\ dp[i-1][j-1], & p_j \text{ 是问号} \\ dp[i][j-1] \vee dp[i-1][j], & p_j \text{ 是星号} \end{cases}$$

```
class Solution {
public:
    //动态规划
    bool isMatch(string s, string p) {
        int m = s.size();
        int n = p.size();
        vector<vector<int>> dp(m + 1, vector<int>(n + 1));
        dp[0][0] = true;
        for (int i = 1; i <= n; ++i) {
            if (p[i - 1] == '*') {
                dp[0][i] = true;
            }
            else {
                break;
            }
        }
        for (int i = 1; i <= m; ++i) {
            for (int j = 1; j <= n; ++j) {
                if (p[j - 1] == '*') {
                    dp[i][j] = dp[i][j - 1] | dp[i - 1][j];
                }
                else if (p[j - 1] == '?' || s[i - 1] == p[j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1];
                }
            }
        }
        return dp[m][n];
    }
};
```