【华为OD机考 统一考试机试C卷】两个字符串间的最短路径问题（C++ Java JavaScript Python）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多 ，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

题目描述

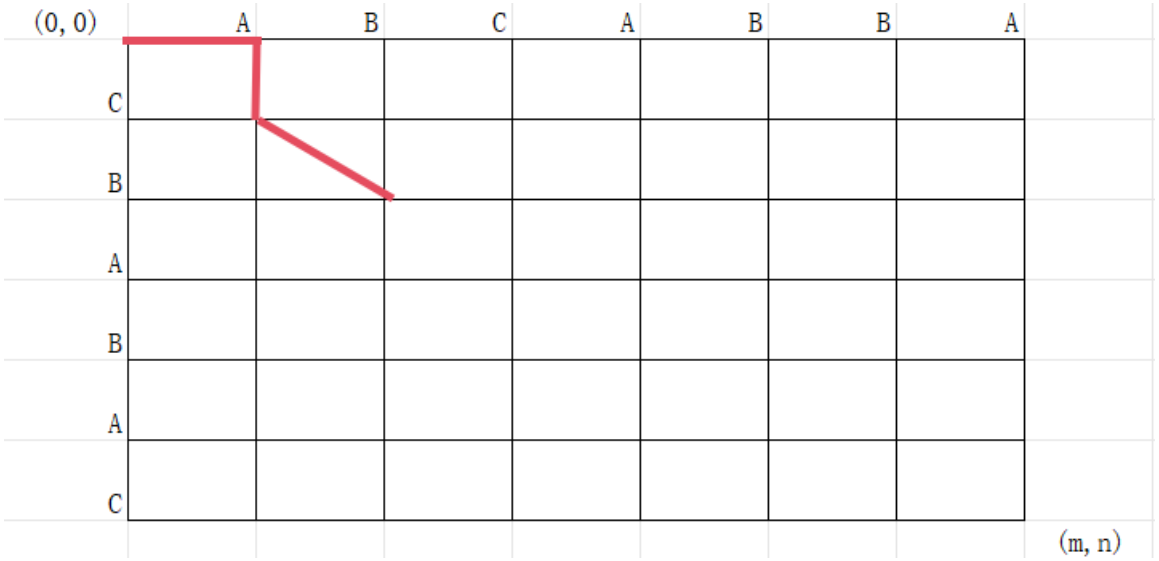
给定两个字符串，分别为字符串A与字符串B。

例如A字符串为ABCABBA，B字符串为CBABAC可以得到m*n的二维数组，定义原点为(0,0)，终点为(m,n)，水平与垂直的每一条边距离为1，

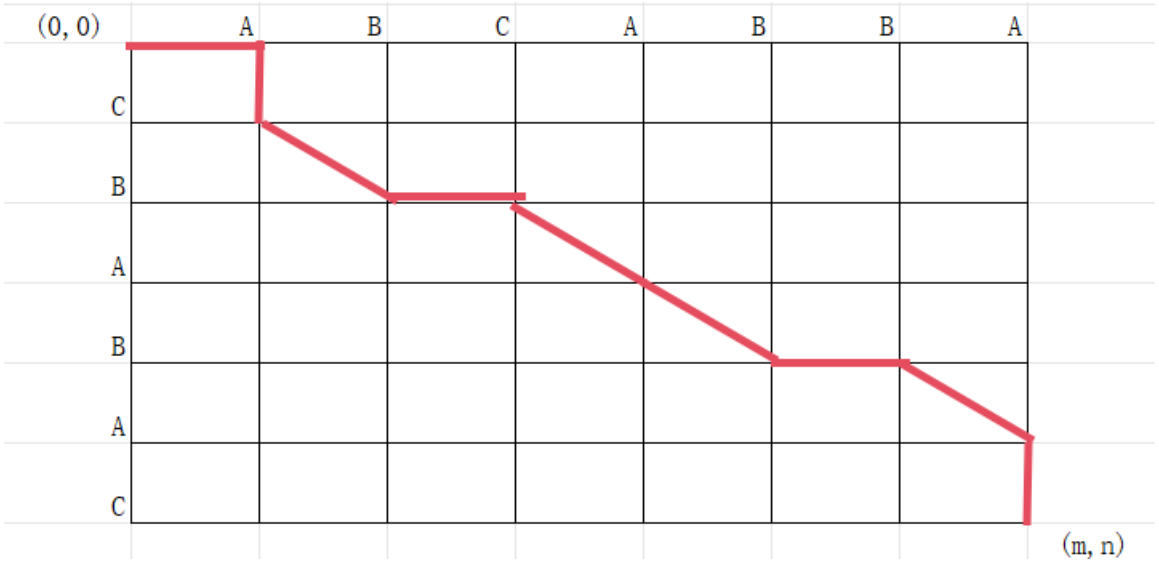
从原点(0,0)到(0,A)为水平边，距离为1，从(0,A)到(A,C)为垂直边，距离为1；

假设两个字符串同一位置的两个字符相同则可以作一个斜边，如(A,C)到(B,B)最短距离为斜边，距离同样为1。

作出所有的斜边，则有(0,0)到(B,B)的距离为 1个水平边+1个垂直边+1个斜边 =3。



根据定义可知，原点到终点的最短距离路径如下图红线标记，最短距离为9；
路径为(0,0)->(A,0)->(A,C)->(B,B)->(C,B)->(A,A)->(B,B)->(B,B)->(A,A)->(A,C)



输入描述

空格分割的两个字符串 A 与字符串 B

- 字符串不为"空串"
- 字符格式满足正则规则：[A-Z]
- 字符串长度 < 10000

输出描述

原点到终点的最短距离

用例

输入	ABC ABC
输出	3
说明	无

输入	ABCABBA CBABAC
输出	9
说明	无

解题思路

题意其实很简单，就是把AB两个字符串的字符映射到坐标轴上面。A的字符串为x轴，B的字符串为y轴。

起始和终点为（0,0）和（m,n）。然后求起点到终点的最短路径。这个路径的求法有限制。

如上面的图所示，起点为（0,0），然后下一个坐标轴，可以往（0，A）或者（C,0）走，这里我们选择往（0，A）走。然后再往（A,C）走。然后我们可以往（C,B）或者（B,0）或（B,B）（这个坐标是因为坐标的x和y都为B,所以可以斜着走）走。知道走到终点。

这个问题可以通过动态规划来解决。动态规划的基本思想是将一个复杂的问题分解为多个子问题，然后通过解决子问题来解决原问题。在这个问题中，我们需要找到从原点到终点的最短距离，这个距离可以通过计算到每个点的最短距离来得到。

首先，我们需要创建一个动态规划数组，用于存储到每个点的最短距离。然后，我们初始化数组的第一行和第一列，即从原点到每个点的距离。

接下来，我们遍历字符串B，对于每个字符，我们遍历字符串A，对于每个字符，我们检查当前字符是否与字符串B的当前字符匹配。如果匹配，那么我们更新动态规划数组的当前位置为左上角的值加1；如果不匹配，那么我们更新动态规划数组的当前位置为左边和上边的最小值加1。

最后，我们输出从原点到终点的最短距离，即动态规划数组的最后一个元素。

以下是详细的推导过程：

1. 初始化：
   - `dp[0]` 到 `dp[n]` 初始化为 `0` 到 `n`，因为从字符串A的开头到每个位置的最短距离就是对应的索引值（即水平移动的距离）。
2. 遍历字符串B（`i` 从 `1` 到 `m`）：
   - 在每次新的行开始时，更新 `dp[0]` 为 `i`，因为从字符串B的开头到当前位置的最短距离就是 `i`（即垂直移动的距离）。
   - 保存 `prev` 为左上角的值，即上一行的 `dp[0]`。
3. 遍历字符串A（`j` 从 `1` 到 `n`）：
   - 保存 `temp` 为当前 `dp[j]` 的值，因为在更新 `dp[j]` 时需要用到。
   - 如果 `A[j-1] == B[i-1]`（字符匹配），则 `dp[j]` 更新为 `prev + 1`，因为可以直接从左上角移动到当前位置。
   - 如果 `A[j-1] != B[i-1]`（字符不匹配），则 `dp[j]` 更新为 `min(dp[j], dp[j-1]) + 1`，即从左边或上边移动到当前位置的最小值加1。
   - 更新 `prev` 为当前 `dp[j]` 的原始值（即 `temp`）。
4. 最终，`dp[n]` 保存了从字符串A到字符串B的最短距离。

这个过程中，我们只保留了当前行和上一行的信息，从而将空间复杂度从O(mn)降低到了O(n)。

C++

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <sstream>

using namespace std;
int main() {
    string input;
    // 读取一行输入
    getline(cin, input);

```

```
    // 使用stringstream分割输入
    stringstream ss(input);
    string A, B;
    ss >> A >> B;

    // 获取字符串B和A的长度
    int m = B.length(), n = A.length();

    // 创建动态规划数组
    vector<int> dp(n + 1);

    // 初始化dp数组的第一行
    for (int j = 0; j <= n; ++j) {
        dp[j] = j;
    }

    // 遍历字符串B
    for (int i = 1; i <= m; ++i) {
        int prev = dp[0];
        dp[0] = i;
        // 遍历字符串A
        for (int j = 1; j <= n; ++j) {
            int temp = dp[j];
            // 如果字符匹配
            if (A[j - 1] == B[i - 1]) {
                dp[j] = prev + 1;
            } else {
                // 如果字符不匹配
                dp[j] = min(dp[j], dp[j - 1]) + 1;
            }
            prev = temp;
        }
    }

    // 输出最短距离
    cout << dp[n] << endl;
    return 0;
}
```

Java

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        String[] parts = input.split(" ");

        // 将分割后的两部分分别赋值给A和B
        String A = parts[0];
        String B = parts[1];
        // 获取字符串B的长度
        int m = B.length();
        // 获取字符串A的长度
        int n = A.length();
        // 创建一个动态规划数组，用于存储到每个点的最短距离
        int[] dp = new int[n + 1];

        // 初始化dp数组的第一行，即从(0,0)到(0,j)的距离
        for (int j = 0; j <= n; j++) {
            dp[j] = j;
        }

        // 遍历字符串B
        for (int i = 1; i <= m; i++) {
            // 保存左上角的值
            int prev = dp[0];
            // 更新dp数组的第一列，即从(0,0)到(i,0)的距离
            dp[0] = i;
            // 遍历字符串A
            for (int j = 1; j <= n; j++) {
                // 保存dp[j]的原始值，用于后面的更新
                int temp = dp[j];
                // 如果当前字符匹配，则更新dp[j]为左上角的值加1
                if (A.charAt(j - 1) == B.charAt(i - 1)) {
                    dp[j] = prev + 1;
                } else {
                    // 如果当前字符不匹配，则更新dp[j]为左边和上边的最小值加1
                    dp[j] = Math.min(dp[j], dp[j - 1]) + 1;
                }
```

```
                // 更新prev为当前dp[j]的原始值
                prev = temp;
            }
        }

        // 输出从(0,0)到(m,n)的最短距离
        System.out.println(dp[n]);
    }
}
```

javaScript

```javascript
const readline = require('readline');

// 创建readline接口
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

// 读取输入
rl.on('line', (input) => {
    const parts = input.split(' ');
    const A = parts[0];
    const B = parts[1];

    const m = B.length;
    const n = A.length;

    let dp = new Array(n + 1).fill(0).map((_, j) => j);

    // 遍历字符串B
    for (let i = 1; i <= m; i++) {
        let prev = dp[0];
        dp[0] = i;

        // 遍历字符串A
        for (let j = 1; j <= n; j++) {
            let temp = dp[j];

```

```
            // 如果字符匹配
            if (A[j - 1] === B[i - 1]) {
                dp[j] = prev + 1;
            } else {
                // 如果字符不匹配
                dp[j] = Math.min(dp[j], dp[j - 1]) + 1;
            }
            prev = temp;
        }
    }

    // 输出最短距离
    console.log(dp[n]);

    rl.close();
});
```

Python

```
# 读取输入
input_str = input()
A, B = input_str.split(" ")

m = len(B)
n = len(A)

# 创建动态规划数组
dp = [j for j in range(n + 1)]

# 遍历字符串B
for i in range(1, m + 1):
    prev = dp[0]
    dp[0] = i

    # 遍历字符串A
    for j in range(1, n + 1):
        temp = dp[j]

        # 如果字符匹配
        if A[j - 1] == B[i - 1]:
```

```
            dp[j] = prev + 1
        else:
            # 如果字符不匹配
            dp[j] = min(dp[j], dp[j - 1]) + 1
        prev = temp

# 输出最短距离
print(dp[n])
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师