


题目描述

小明负责维护项目下的代码，需要查找出重复代码，用以支撑后续的代码优化，请你帮助小明找出重复的代码。重复代码查找方法：以字符串形式给定两行代码（字符串长度 $1 < \text{length} \leq 100$ ，由英文字母、数字和空格组成），找出两行代码中的 **最长公共子串** 。

注：如果不存在公共子串，返回空字符串

输入描述

输入的参数text1, text2分别表示两行代码

输出描述

输出任一最长公共子串

用例

输入	hello123world hello123abc4
输出	hello123
说明	无

输入	private_void_method public_void_method
输出	_void_method
说明	无

输入	hiworld hiweb
输出	hiw
说明	无

题目解析

本题可以使用 **动态规划** 求解，下面解释下本题动态规划解题思路。

动态规划，一般是将大问题分解为规模更小的相同的子问题，如果子问题依旧很难求解，则继续分解，直到规模小到子问题可以被轻松求解。

比如上面求两个字符串的最长公共子串，如果两个字符串很长，那么我们将很难一下子发现最长公共子串，那么我们可以只看两个字符串的部分范围，比如字符串str1，只看0~i范围，字符串str2，只看0~j范围，然后求解str1的0~i范围和str2的0~j范围的最长公共子串，最简单就是str1的0~0范围，即空串，和任意str2范围的最长公共子串都是空串，接着我们可以慢慢扩大str1的范围。

可能上面说的还比较抽象，我们可以用一个二维数组dp来描述上面逻辑。

dp[i][j] 表示的是, str1的0~i范围和 str2的0~j范围 的公共子串的长度

初始时，我们可以得出如下二维矩阵

[illegible]

题目解析

本题可以使用 [动态规划](#)^Q 求解，下面解释下本题动态规划解题思路。

动态规划，一般是将大问题分解为规模更小的相同的子问题，如果子问题依旧很难求解，则继续分解，直到规模小到子问题可以被轻松求解。

比如上面求两个字符串的最长公共子串，如果两个字符串很长，那么我们将很难一下子发现最长公共子串，那么我们可以只看两个字符串的部分范围，比如字符串str1，只看0~i范围，字符串str2，只看0~j范围，然后求解str1的0~i范围和str2的0~j范围的最长公共子串，最简单就是str1的0~0范围，即空串，和任意str2范围的最长公共子串都是空串，接着我们可以慢慢扩大str1的范围。

可能上面说的还比较抽象，我们可以用一个二维数组dp来描述上面逻辑。

dp[i][j] 表示的是，str1的0~i范围和 str2的0~j范围 的公共子串的长度

初始时，我们可以得出如下二维矩阵

str1 \ str2													
	""	h	e	l	l	o	1	2	3	a	b	c	4
""	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0												
e	0												
l	0												
l	0												
o	0												
1	0												
2	0												
3	0												
w	0												
o	0												
r	0												
l	0												
d	0												

CSDN @伏城之外

即str1取0~0，即空串时，和任意str2范围的公共子串的长度都是0。

同理，str2取0~0，即空串时，和任意str1范围的公共子串的长度都是0。

接下来，str1取0~1范围（即h），和str2取0~1范围（即h），的公共子串其实就是h，因此长度为1。如下图所示

str1 \ str2													
	""	h	e	l	l	o	1	2	3	a	b	c	4
""	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	1											
e	0												
l	0												
l	0												
o	0												
1	0												
2	0												
3	0												
w	0												
o	0												
r	0												
l	0												
d	0												

CSDN @伏城之外

但是上面公共子串长度求解，也可以看成是，str1扩大范围新增的h，和str2扩大范围新增的h相同，因此出现了公共子串，那么就相当于在str1未扩大范围前（即空串时），和str2未扩大范围前（即空串时），的公共子串长度的基础上+1。

即： $dp[1][1] = dp[0][0] + 1$

进一步分析，其实可得出 [状态转移方程](#)^Q：

$if(str1[i] == str2[j]) \quad dp[i][j] = dp[i-1][j-1] + 1$

 伏城之外 已关注

4

5

0

专栏目录

已订阅

str1 \ str2	""	h	e	l	l	o	1	2	3	a	b	c	4
""	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	1											
e	0												
l	0												
l	0												
o	0												
1	0												
2	0												
3	0												
w	0												
o	0												
r	0												
l	0												
d	0												

但是上面公共子串长度求解，也可以看成是，str1扩大范围新增的h，和str2扩大范围新增的h相同，因此出现了公共子串，那么就相当于在str1未扩大范围前（即空串时），和str2未扩大范围前（即空串时），的公共子串长度的基础上+1。

即：dp[1][1] = dp[0][0] + 1

进一步分析，其实可得出 **状态转移方程**：

if(str1[i] == str2[j]) dp[i][j] = dp[i-1][j-1] + 1

再接下来，str1还是取0~1范围（即h），和str2取0~2范围（即he），

str1 \ str2	""	h	e	l	l	o	1	2	3	a	b	c	4
""	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	1	0										
e	0												
l	0												
l	0												
o	0												
1	0												
2	0												
3	0												
w	0												
o	0												
r	0												
l	0												
d	0												

此时str1还是相当于新增h，而str2相当于新增e，新增部分不相同，所以此处没有增长前面基础上得到的公共子串的长度。

那么此时相当于 if(str1[i] != str2[j]) dp[i][j] = dp[i-1][j-1] + 0，

但是这种思路是错误的，因为当str1,str2新增范围字符不同时，意味着公共子串的中断，我们应该将此时的dp[i][j]置为0，这样才能防止dp[i+1][j+1]对应的字符相同时，继承前面的公共子串长度。

请看下面错误例子

		a	b	f	c
		0	0	0	0
a		0	1		
b		0		2	
e		0			2

str1 \ str2	""	h	e	l	l	o	1	2	3	a	b	c	4
""	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	1	0										
e	0												
l	0												
l	0												
o	0												
1	0												
2	0												
3	0												
w	0												
o	0												
r	0												
l	0												
d	0												

此时str1还是相当于新增h，而str2相当于新增e，新增部分不相同，所以此处没有增长前面基础上得到的公共子串的长度。

那么此时相当于 $\text{if}(\text{str1}[i] \neq \text{str2}[j]) \text{ dp}[i][j] = \text{dp}[i-1][j-1] + 0$,

但是这种思路是错误的，因为当str1,str2新增范围字符不同时，意味着公共子串的中断，我们应该将此时的 $\text{dp}[i][j]$ 置为0，这样才能防止 $\text{dp}[i+1][j+1]$ 对应的字符相同时，继承前面的公共子串长度。

请看下面错误例子

		a	b	f	c
	0	0	0	0	0
a	0	1			
b	0		2		
e	0			2	
c	0				3

正确例子应该是

		a	b	f	c
	0	0	0	0	0
a	0	1			
b	0		2		
e	0			0	
c	0				1

因此正确的动态规划状态转移方程是：

- $\text{dp}[i][0] = 0$
- $\text{dp}[0][j] = 0$
- $\text{if}(\text{str1}[i] == \text{str2}[j]) \text{ dp}[i][j] = \text{dp}[i-1][j-1] + 1 \text{ else } \text{dp}[i][j] = 0$

但是此题并不是让我们求解最长的公共子串的长度，而是求出任一最长公共子串。因此我们可以在遍历求解上面二维矩阵每一个元素时，记录下公共子串最大的长度值，比如下面例子中，求解到黄色元素 $\text{dp}[2][2]$ 时，得到了目前最大的公共子串长度2。

		a	b	f	c
	0	0	0	0	0
a	0	1	0	0	0
b	0	0	2		
e	0				
c	0				

但是此题并不是让我们求解最长的公共子串的长度，而是求出任一最长公共子串。因此我们可以在遍历求解上面二维矩阵每一个元素时，记录下公共子串最大的长度值，比如下面例子中，求解到黄色元素 $dp[2][2]$ 时，得到了目前最大的公共子串长度2，

		a	b	f	c
	0	0	0	0	0
a	0	1	0	0	0
b	0	0	2		
e	0				
c	0				

因此我们可以从任一维度来获取这个公共子串，比如从列维度，相当于 $str2.slice(j - maxLen, j)$ ，其中 j 对应 $dp[i][j]$ 中的 j 。 $maxLen=dp[i][j]$ 。

		a	b	f	c
	0	0	0	0	0
a	0	1	0	0	0
b	0	0	2		
e	0				
c	0				

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12 });
13
14 if (lines.length === 2) {
15   const str1 = lines[0];
16   const str2 = lines[1];
17   console.log(getResult(str1, str2));
18   lines.length = 0;
19 }
20
21 function getResult(str1, str2) {
22   const n = str1.length;
23   const m = str2.length;
24
25   const dp = new Array(n + 1).fill(0).map(() => new Array(m + 1).fill(0));
26
27   let max = 0;
28   let ans = "";
29
30   for (let i = 1; i <= n; i++) {
31     for (let j = 1; j <= m; j++) {
32       if (str1[i - 1] === str2[j - 1]) {
33         dp[i][j] = dp[i - 1][j - 1] + 1;
34
35         if (dp[i][j] > max) {
36           max = dp[i][j];
37           ans = str1.slice(i - max, i);
38         }
39       } else {
40         dp[i][j] = 0;
41       }
42     }
43   }
44
45   return ans;
46 }
```

Java算法源码

```
1  import java.util.Scanner;
2
3  public class Main {
4    public static void main(String[] args) {
5      Scanner sc = new Scanner(System.in);
6
7      String str1 = sc.next();
8      String str2 = sc.next();
9
10     System.out.println(getResult(str1, str2));
11 }
```

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const str1 = lines[0];
15     const str2 = lines[1];
16     console.log(getResult(str1, str2));
17     lines.length = 0;
18   }
19 });
20
21 function getResult(str1, str2) {
22   const n = str1.length;
23   const m = str2.length;
24
25   const dp = new Array(n + 1).fill(0).map(() => new Array(m + 1).fill(0));
26
27   let max = 0;
28   let ans = "";
29
30   for (let i = 1; i <= n; i++) {
31     for (let j = 1; j <= m; j++) {
32       if (str1[i - 1] === str2[j - 1]) {
33         dp[i][j] = dp[i - 1][j - 1] + 1;
34
35         if (dp[i][j] > max) {
36           max = dp[i][j];
37           ans = str1.slice(i - max, i);
38         }
39       } else {
40         dp[i][j] = 0;
41       }
42     }
43   }
44
45   return ans;
46 }
```

Java算法源码

```
1  import java.util.Scanner;
2
3  public class Main {
4    public static void main(String[] args) {
5      Scanner sc = new Scanner(System.in);
6
7      String str1 = sc.next();
8      String str2 = sc.next();
9
10     System.out.println(getResult(str1, str2));
11   }
12
13   public static String getResult(String str1, String str2) {
14     int n = str1.length();
15     int m = str2.length();
16
17     int[][] dp = new int[n + 1][m + 1];
18
19     int max = 0;
20     String ans = "";
21
22     for (int i = 1; i <= n; i++) {
23       for (int j = 1; j <= m; j++) {
24         if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
25           dp[i][j] = dp[i - 1][j - 1] + 1;
26
27           if (dp[i][j] > max) {
28             max = dp[i][j];
29             ans = str1.substring(i - max, i);
30           }
31         } else {
32           dp[i][j] = 0;
33         }
34       }
35     }
36
37     return ans;
38   }
39 }
```

Java算法源码

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         String str1 = sc.next();
8         String str2 = sc.next();
9
10        System.out.println(getResult(str1, str2));
11    }
12
13    public static String getResult(String str1, String str2) {
14        int n = str1.length();
15        int m = str2.length();
16
17        int[][] dp = new int[n + 1][m + 1];
18
19        int max = 0;
20        String ans = "";
21
22        for (int i = 1; i <= n; i++) {
23            for (int j = 1; j <= m; j++) {
24                if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
25                    dp[i][j] = dp[i - 1][j - 1] + 1;
26
27                    if (dp[i][j] > max) {
28                        max = dp[i][j];
29                        ans = str1.substring(i - max, i);
30                    }
31                } else {
32                    dp[i][j] = 0;
33                }
34            }
35        }
36
37        return ans;
38    }
39 }
```

[复制](#)

Python算法源码

```
1 # 输入获取
2 str1 = input()
3 str2 = input()
4
5
6 # 算法入口
7 def getResult(str1, str2):
8     n = len(str1)
9     m = len(str2)
10
11     dp = [[0 for i in range(m + 1)] for j in range(n + 1)]
12
13     maxV = 0
14     ans = ""
15
16     for i in range(1, n + 1):
17         for j in range(1, m + 1):
18             if str1[i - 1] == str2[j - 1]:
19                 dp[i][j] = dp[i - 1][j - 1] + 1
20
21                 if dp[i][j] > maxV:
22                     maxV = dp[i][j]
23                     ans = str1[(i - maxV):i]
24             else:
25                 dp[i][j] = 0
26
27     return ans
28
29
30 # 算法调用
31 print(getResult(str1, str2))
```