

题目描述

小华负责公司 知识图谱 产品，现在要通过新词挖掘完善知识图谱。

新词挖掘：给出一个待挖掘问题内容字符串Content和一个词的字符串word，找到content中所有word的新词。

新词：使用词word的字符排列形成的字符串。

请帮小华实现新词挖掘，返回发现的新词的数量。

输入描述

第一行输入为待挖掘的文本内容content；

第二行输入为词word；

输出描述

在content中找到的所有word的新词的数量。

备注

- $0 \leq \text{content的长度} \leq 10000000$
- $1 \leq \text{word的长度} \leq 2000$

用例

输入	qweebaewqd qwe
输出	2
说明	起始索引等于0的子串是“qwe”，它是word的新词。 起始索引等于6的子串是“ewq”，它是word的新词。
输入	abab ab
输出	3
说明	起始索引等于0的子串是”ab“，它是word的新词。 起始索引等于1的子串是”ba“，它是word的新词。 起始索引等于2的子串是”ab“，它是word的新词。

题目解析

本题最简单的解法就是利用定长（word长度） 滑动窗口 来选择 content中的子串，只要选中的子串和word二者，在字典序排序后是一样的，则可以记为一个挖掘到的新词。代码如下：

```
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 content = lines[0];
15   const word = lines[1];
16
17   console.log(getResult(content, word));
```

## 题目解析

本题最简单的解法就是利用定长（word长度）[滑动窗口](#)来选 content 中的子串，只要选中的子串和 word 二者，在字典序排序后是一样的，则可以记为一个挖掘到的新词。代码如下：

```
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 content = lines[0];
15     const word = lines[1];
16
17     console.log(getResult(content, word));
18     lines.length = 0;
19   }
20 });
21
22 function getResult(content, word) {
23   if (content.length < word.length) {
24     return 0;
25   }
26
27   const sorted_word = [...word].sort().join("");
28
29   let ans = 0;
30   let maxI = content.length - word.length;
31   for (let i = 0; i <= maxI; i++) {
32     const sorted_substr = [...content.slice(i, i + word.length)]
33       .sort()
34       .join("");
35
36     if (sorted_substr === sorted_word) ans++;
37   }
38
39   return ans;
40 }
```

但是上面算法的时间复杂度在： $O(n*m)$ ，其中n是content长度，m是word长度，也就是说，最多的会有  $10000000 * 2000$  次循环，有人可能会疑问时间复杂度中m哪来的？那是content截取出来的子串[字典序排序](#)的时间复杂度m。

因此上面的算法极容易超时。

那么上面算法是否有优化的可能呢？

我的优化思路是，利用word和content截取子串的字母数量比较来代替字典序排序后比较。

因为字典序排序存在一个较大的性能浪费，比如

```
qweebaewqd
xyz
```

我们可以明显发现，content中不存在word新词，因为任意位置长度3的滑动窗口中的第一个字母都不属于word，因此仅从滑动窗口内第一个字母就可以判断出结果，而不是需要截取子串字典序排序后再比较得出结果。

但是，通过字母数量比较也有缺陷，那就是，如果只有滑动窗口内部各字母数量和word的字母数量严格一致，才能算挖掘到了新词。

我的实现思路是：

遍历滑动窗口内部的字母，如c，如果该字母c是word中的，则先看统计数量count[c]是不是已经为0了，如果已经为0了，则说明当前滑动窗口遍历到的字母c是超标部分，因此当前滑动窗口不可用。下一个滑动窗口应该保证字母c不超标，即下一个滑动窗口的左边界应该在当前滑动窗口中字母c第一次出现位置的右边。

当然，如果遍历完滑动窗口内部所有字母，也没有出现超标现象，则说明该滑动窗口就是要挖掘的新词。

上面算法逻辑，其实我不太确定是否有多大的性能提升，但是相较于字典序排序应该有较多提升。我把上面算法逻辑称为跳跃式，即滑动窗口的左边界不是单统一格一格往右滑的，而是可以多格滑动的。

为了以防万一，我这里还想了一个传统滑动窗口的解法，那就是一格一格移动，但是也是通过统计滑动窗口内部字母数量来判断是否是word新词，但是后一个滑动窗口可以通过差异化思想，快速基于前一个滑动窗口得出自身内部字母数量。这块逻辑可以参考：


华为OD机试 - 完美走位\_伏城之外的博客-CSDN博客\_完美走位华为

中求解最小覆盖子串的过程。

JavaScript算法源码

跳跃式

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
```

 伏城之外 [已关注](#)

👍 0

💬

★ 7

📁

📄 0

🔖

📖 专栏目录

🔔 已订阅

```

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 content = lines[0];
15     const word = lines[1];
16
17     console.log(getResult(content, word));
18     lines.length = 0;
19   }
20 });
21
22 function getResult(content, word) {
23   // 如果content长度小于word, 则直接返回0
24   if (content.length < word.length) {
25     return 0;
26   }
27
28   // count统计word中各字母数量
29   const count = {};
30   for (let c of word) {
31     count[c] ? count[c]++ : (count[c] = 1);
32   }
33
34   // ans保存题解
35   let ans = 0;
36
37   // 滑动窗口左指针的移动范围为 0~maxI
38   let maxI = content.length - word.length;
39   outer: for (let i = 0; i <= maxI; i++) {
40     const countb = JSON.parse(JSON.stringify(count));
41     const letters = {};
42     for (let j = i; j < i + word.length; j++) {
43       const c = content[j];
44       // 如果滑动窗口j位置字母不属于word, 则说明当前滑动窗口内的单词肯定不是word新词, 且j之前的所有滑动窗口都毁了, 只能从j之后开始
45       if (count[c] === undefined) {
46         i = j;
47         continue outer;
48       }
49
50       // 如果滑动窗口内部字母是word中的, 且在限定数量内
51       if (countb[c] > 0) {
52         // 出现了一个, 则规定数量--
53         countb[c]--;
54         // 记录content中出现的字母的位置
55         letters[c] ? letters[c].push(j) : (letters[c] = [j]);
56       } else {
57         // 如果滑动窗口j位置字母属于word, 但是此时由于j位置字母的加入, 导致滑动窗口内部对应字母数量超过了word, 则说明当前滑动窗口不满足条件
58         // 且我们应该让之后的滑动窗口内部该字母数量不能超, 因此下一个滑动窗口的起始位置应该是该字母在当前滑动窗口第一次出现位置
59         i = letters[c][0];
60         continue outer;
61       }
62     }
63     ans++;
64   }
65
66   return ans;
67 }

```

传统式

```

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 content = lines[0];
15     const word = lines[1];
16
17     console.log(getResult(content, word));
18     lines.length = 0;
19   }
20 });

```

传统式

```
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 content = lines[0];
15     const word = lines[1];
16
17     console.log(getResult(content, word));
18     lines.length = 0;
19   }
20 });
21
22 function getResult(content, word) {
23   // 如果content长度小于word, 则直接返回0
24   if (content.length < word.length) {
25     return 0;
26   }
27
28   // ans保存题解
29   let ans = 0;
30
31   // total记录word字母总数
32   let total = word.length;
33   // count统计word中各字母数量
34   const count = {};
35   for (let c of word) {
36     count[c] ? count[c]++ : (count[c] = 1);
37   }
38
39   // 初始滑动窗口内即字母遍历
40   for (let i = 0; i < word.length; i++) {
41     const c = content[i];
42     if (count[c] !== undefined && count[c]-- > 0) {
43       total--;
44     }
45   }
46   if (total === 0) ans++;
47
48   // 滑动窗口左指针的移动范围为 0~maxI
49   let maxI = content.length - word.length;
50   for (let i = 1; i <= maxI; i++) {
51     const remove_c = content[i - 1];
52     const add_c = content[i + word.length - 1];
53
54     if (count[remove_c] !== undefined && count[remove_c]++ >= 0) {
55       total++;
56     }
57
58     if (count[add_c] !== undefined && count[add_c]-- > 0) {
59       total--;
60     }
61
62     if (total === 0) ans++;
63   }
64
65   return ans;
66 }
```

Java算法源码

Java这里只实现一个传统式的，因为感觉跳跃式的性能提升局限性太大。

```
1  import java.util.HashMap;
2  import java.util.Scanner;
3
4  public class Main {
5    public static void main(String[] args) {
6      Scanner sc = new Scanner(System.in);
7
8      String content = sc.next();
9      String word = sc.next();
10
11      System.out.println(getResult(content, word));
12    }
13
14    public static int getResult(String content, String word) {
15      // 如果content长度小于word, 则直接返回0
16      if (content.length() < word.length()) {
17        return 0;
18      }
19    }
20  }
```

伏城之外 已关注

👍 0 🗨 7 🌟 0 📁 专栏目录 已订阅

## Java算法源码

Java这里只实现一个传统式的，因为感觉跳跃式的性能提升局限性太大。

```
1 import java.util.HashMap;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         String content = sc.next();
9         String word = sc.next();
10
11         System.out.println(getResult(content, word));
12     }
13
14     public static int getResult(String content, String word) {
15         // 如果content长度小于word，则直接返回0
16         if (content.length() < word.length()) {
17             return 0;
18         }
19
20         // ans保存题解
21         int ans = 0;
22
23         // total记录word字母总数
24         int total = word.length();
25
26         // count统计word中各字母数量
27         HashMap<Character, Integer> count = new HashMap<>();
28         for (int i = 0; i < word.length(); i++) {
29             Character c = word.charAt(i);
30             count.put(c, count.getOrDefault(c, 0) + 1);
31         }
32
33         // 初始滑动窗口内部字母遍历
34         for (int i = 0; i < word.length(); i++) {
35             Character c = content.charAt(i);
36             if (count.containsKey(c)) {
37                 if (count.get(c) > 0) {
38                     total--;
39                 }
40                 count.put(c, count.get(c) - 1);
41             }
42         }
43
44         if (total == 0) ans++;
45
46         // 滑动窗口左指针的移动范围为 0~maxI
47         int maxI = content.length() - word.length();
48         for (int i = 1; i <= maxI; i++) {
49             Character remove_c = content.charAt(i - 1);
50             Character add_c = content.charAt(i + word.length() - 1);
51
52             if (count.containsKey(remove_c)) {
53                 if (count.get(remove_c) >= 0) {
54                     total++;
55                 }
56                 count.put(remove_c, count.get(remove_c) + 1);
57             }
58
59             if (count.containsKey(add_c)) {
60                 if (count.get(add_c) > 0) {
61                     total--;
62                 }
63                 count.put(add_c, count.get(add_c) - 1);
64             }
65
66             if (total == 0) ans++;
67         }
68         return ans;
69     }
70 }
```

## Python算法源码

```
1 content = input()
2 word = input()
3
4
5 def getResult(content, word):
6     # 如果content长度小于word，则直接返回0
7     if len(content) < len(word):
8         return 0
9
10    # ans保存题解
11    ans = 0
12
13    # total记录word字母总数
```

伏城之外 已关注

👍 0 🗨 7



专栏目录

已订阅

## Python算法源码

```
1 content = input()
2 word = input()
3
4
5 def getResult(content, word):
6     # 如果content长度小于word, 则直接返回0
7     if len(content) < len(word):
8         return 0
9
10    # ans保存题解
11    ans = 0
12
13    # total记录word字母总数
14    total = len(word)
15
16    # count统计word中各字母数量
17    count = {}
18    for c in word:
19        if count.get(c) is None:
20            count[c] = 1
21        else:
22            count[c] += 1
23
24    # 初始滑动窗口内部字母遍历
25    for i in range(len(word)):
26        c = content[i]
27        if count.get(c) is not None:
28            if count[c] > 0:
29                total -= 1
30                count[c] -= 1
31
32    if total == 0:
33        ans += 1
34
35    # 滑动窗口左指针的移动范围为 0~maxI
36    maxI = len(content) - len(word) + 1
37    for i in range(1, maxI):
38        remove_c = content[i - 1]
39        add_c = content[i + len(word) - 1]
40
41        if count.get(remove_c) is not None:
42            if count[remove_c] >= 0:
43                total += 1
44                count[remove_c] += 1
45
46        if count.get(add_c) is not None:
47            if count[add_c] > 0:
48                total -= 1
49                count[add_c] -= 1
50
51        if total == 0:
52            ans += 1
53
54    return ans
55
56
57 print(getResult(content, word))
```