21、字符串比较，考点 or 实现——双指针

题目描述

给定字符串A、B和正整数V，A的长度与B的长度相等， 请计算A中满足如下条件的最大连续子串的长度：

1. 该连续子串在A和B中的位置和长度均相同。
2. 该连续子串|A[i] – B[i]|之和小于等于V。其中|A[i] – B[i]|表示两个字母**ASCII码之差的绝对值**。

输入描述

输入为三行：

- 第一行为字符串A，仅包含小写字符，1 <= A.length <=1000。
- 第二行为字符串B，仅包含小写字符，1 <= B.length <=1000。
- 第三行为正整数V，0<= V <= 10000。

输出描述

字符串最大连续子串的长度，要求该子串|A[i] – B[i]|之和小于等于V。

用例

输入	xxcdefg cdefghi 5
输出	2
说明	字符串A为xxcdefg，字符串B为cdefghi，V=5。 它的最大连续子串可以是cd->ef,de->fg,ef->gh,fg->hi， 所以最大连续子串是2。

在线OJ

题目详情 - 字符串比较 - HydroOJ

题目解析

本题其实可以转化为求解：和不超过v的最长连续子序列问题。

字符串a	x	x	c	d	e	f	g
字符串b	c	d	e	f	g	h	i
各位字符ascii码差绝对值	21	20	2	2	2	2	2

原始数组就是上面的ascii码差值绝对值数组diff：[21, 20, 2, 2, 2, 2, 2]

本题数量级不大，diff数组的长度最大1000，因此我们只要求出区间和<=v的所有区间，取其中最长的即可。这里任意区间的区间和求解可以通过前缀和完成，具体可以看下面博客：

算法设计 - 前缀和 & 差分数列_伏城之外的博客-CSDN博客

或者我们可以利用滑动窗口来求解：和不超过v的最长连续子序列问题。

我们可以定义两个指针L,R，分别代表滑窗的左右边界，初始化时，L，R都为0，

然后再定义一个滑窗内部和sum，初始为diff[r]

接下来，判断sum和v的大小：

- 如果 sum < v，则说明滑窗内部和过小，我们应该将滑窗的右边界R++，来扩大滑窗，滑窗内部和sum += diff[R]，需要注意的是，这里我们需要注意R越界问题，需要先判断R++是否越界，如果未越界，才能sum += diff[R]
- 如果 sum == v，则说明滑窗内部和刚刚好，我们应该记录此时滑窗对应的连续子序列长度：R - L + 1 作为一个可能解，接下来就是滑窗左右边界的移动问题：

> 1. 按照以往滑窗运动经验，此时应该L++，R++，但是这里我们只应该做R++，而不应该做L++，原因是后续的diff[i]可能都是0，即不会增加sum，只会增加连续子序列的长度，因此如果这种情况做了L++的话，我们会得不到最优解。
> 2. 同样地，这里做R++，也需要注意R越界问题，只有R++后不越界，我们才能sum += diff[R]

- 如果 sum > v，我们应该让滑窗左边界L++，来减少sum，即sum -= diff[L]，但是在做滑窗左边界L++之前，我们应该确认一下，上一个状态的滑窗，即范围是[L, R-1]的滑窗是否是满足sum <= v的滑窗，如果是，则我们需要记录上一个滑窗的长度R - L

> 1. 需要注意的是，当前滑窗做滑窗左边界L++后，L是有可能超过R的，因此我们需要保证L超过R后，R的位置要更新到等于L的地方，此时又相当于给滑窗sum += diff[R]
> 2. 同样地，需要注意R更新位置的越界问题，即只有R=L后不越界，才能sum += diff[R]

前缀和解法

Java算法源码

```java
import java.util.Scanner;

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

    String a = sc.nextLine();
    String b = sc.nextLine();
    int v = Integer.parseInt(sc.nextLine());

    System.out.println(getResult(a, b, v));
  }

  public static int getResult(String a, String b, int v) {
    int n = a.length();

    // a,b字符串的各位字符的ascii绝对值差距数组
    int[] preSum = new int[n + 1];
    for (int i = 1; i <= n; i++) {
      preSum[i] = preSum[i - 1] + Math.abs(a.charAt(i - 1) - b.charAt(i - 1));
    }

    // 记录题解
    int ans = 0;

```

```
25
26      for (int l = 0; l <= n - 1; l++) {
27        for (int r = l + 1; r <= n; r++) {
28          // 区间 [l+1, r]的和 = preSum[r] - preSum[l]
29          if (preSum[r] - preSum[l] <= v) {
30            ans = Math.max(ans, r - l);
31          }
32        }
33      }
34
35      return ans;
36    }
37  }
```

JS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 === 3) {
14     let a = lines[0];
15     let b = lines[1];
16     let v = parseInt(lines[2]);
17
18     console.log(getResult(a, b, v));
19
20     lines.length = 0;
21   }
22 });
23
24 function getResult(a, b, v) {
25   const n = a.length;
26
27   // a,b字符串的各位字符的ascii绝对值差距数组
28   const preSum = new Array(n + 1).fill(0);
```

```
  const preSum = new Array(n + 1).fill(0);
  for (let i = 1; i <= n; i++) {
    preSum[i] =
      preSum[i - 1] + Math.abs(a[i - 1].charCodeAt() - b[i - 1].charCodeAt());
  }

  // 记录题解
  let ans = 0;

  for (let l = 0; l <= n - 1; l++) {
    for (let r = l + 1; r <= n; r++) {
      // 区间 [l+1, r]的和 = preSum[r] - preSum[l]
      if (preSum[r] - preSum[l] <= v) {
        ans = Math.max(ans, r - l);
      }
    }
  }

  return ans;
}
```

Python算法源码

```
# 输入获取
a = input()
b = input()
v = int(input())


# 算法入口
def getResult():
    n = len(a)

    # a,b字符串的各位字符的ascii绝对值差距数组
    preSum = [0] * (n+1)
    for i in range(1, n+1):
        preSum[i] = preSum[i-1] + abs(ord(a[i-1]) - ord(b[i-1]))

    # 记录题解
    ans = 0

    for l in range(n):
        for r in range(l+1, n+1):
            # 区间 [l+1, r]的和 = preSum[r] - preSum[l]
            if preSum[r] - preSum[l] <= v:
                ans = max(ans, r-l)

    return ans


# 调用算法
print(getResult())
```

滑动窗口解法

Java算法源码

```java
import java.util.Scanner;

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

    String a = sc.nextLine();
    String b = sc.nextLine();
    int v = Integer.parseInt(sc.nextLine());

    System.out.println(getResult(a, b, v));
  }

  public static int getResult(String a, String b, int v) {
    int n = a.length();

    // a,b字符串的各位字符的ascii绝对值差距数组
    int[] diff = new int[n];
    for (int i = 0; i < n; i++) {
      diff[i] = Math.abs(a.charAt(i) - b.charAt(i));
    }

    // 记录题解
    int ans = 0;

    // 滑窗左右边界
    int l = 0;
    int r = 0;

    // 初始滑窗的内部和
    int sum = diff[r];

    while (r < n) {
      if (sum > v) {
        // 如果滑窗内部和超过了v，则我们需要先记录上一个滑窗[l, r-1]的长度
        if (sum - diff[r] <= v) ans = Math.max(ans, r - l);
        // 然后由于当前滑窗内部和已经超过了v，因此需要减少滑窗内部和，只能让滑窗左边界+1，内部和减去失去的diff[l]
        sum -= diff[l++];
        if (r < l) {
          // 注意左边界右移不能超过右边界，如果超过了，则右边界也需要+1，即变为左边界位置，此时内部和需要加入新右边界值diff[r]
          r = l;
          if (r < n) sum += diff[r];
        }
      } else {
        // 如果滑窗内部和没有超过v
        if (sum == v) {
          // 如果滑窗内部和==v,那么当前滑窗就是一个符合要求的，需要记录此时滑窗[l,r]的长度
          ans = Math.max(ans, r - l + 1);
        }
        // 接下来只做滑窗右边界+1，注意右边界不能越界，滑窗需要纳入新右边界只
        // 这里没有做左边界+1 动作，是因为后续的diff有可能都为0，
        // 比如diff = [0, 5, 0, 0, 0], v=5, 当L=0，R=1时，符合当前条件，如果此处做了l++,r++,那么将得不到最大长度
        if (++r < n) sum += diff[r];
      }
    }

    // 注意收尾处理，即最后必然是r越界，结束循环，因此最后一轮滑窗范围是[l, r-1]
    return Math.max(ans, r - l);
  }
}
```

JS算法源码

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

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

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 3) {
    let a = lines[0];
    let b = lines[1];
    let v = parseInt(lines[2]);

    console.log(getResult(a, b, v));

    lines.length = 0;
  }
});

function getResult(a, b, v) {
  const n = a.length;

  // a,b字符串的各位字符的ascii绝对值差距数组
  const diff = new Array(n);
```

```
  const diff = new Array(n);
  for (let i = 0; i < n; i++) {
    diff[i] = Math.abs(a[i].charCodeAt() - b[i].charCodeAt());
  }

  // 记录题解
  let ans = 0;

  // 滑窗左右边界
  let l = 0;
  let r = 0;

  // 初始滑窗的内部和
  let sum = diff[r];

  while (r < n) {
    if (sum > v) {
      // 如果滑窗内部和超过了v，则我们需要先记录上一个滑窗[l, r-1]的长度
      if (sum - diff[r] <= v) ans = Math.max(ans, r - l);
      // 然后由于当前滑窗内部和已经超过了v，因此需要减少滑窗内部和，只能让滑窗左边界+1，内部和减去失去的diff[l]
      sum -= diff[l++];
      if (r < l) {
        // 注意左边界右移不能超过右边界，如果超过了，则右边界也需要+1，即变为左边界位置，此时内部和需要加入新右边界值diff[r]
        r = l;
        if (r < n) sum += diff[r];
      }
    } else {
      // 如果滑窗内部和没有超过v
      if (sum == v) {
        // 如果滑窗内部和==v,那么当前滑窗就是一个符合要求的，需要记录此时滑窗[l,r]的长度
        ans = Math.max(ans, r - l + 1);
```

```
        ans = Math.max(ans, r - l + 1);
      }
      // 接下来只做滑窗右边界+1，注意右边界不能越界，滑窗需要纳入新右边界只
      // 这里没有做左边界+1 动作，是因为后续的diff有可能都为0，
      // 比如diff = [0, 5, 0, 0, 0], v=5, 当L=0，R=1时，符合当前条件，如果此处做了l++,r++,那么将得不到最大长度
      if (++r < n) sum += diff[r];
    }
  }

  // 注意收尾处理，即最后必然是r越界，结束循环，因此最后一轮滑窗范围是[l, r-1]
  return Math.max(ans, r - l);
}
```

Python算法源码

```python
# 输入获取
a = input()
b = input()
v = int(input())


# 算法入口
def getResult():
    n = len(a)

    # a,b字符串的各位字符的ascii绝对值差距数组
    diff = [0] * n
    for i in range(n):
        diff[i] = abs(ord(a[i]) - ord(b[i]))

    # 记录题解
    ans = 0

    # 滑窗左右边界
    l = 0
    r = 0

    # 初始滑窗的内部和
    total = diff[r]

    while r < n:
        if total > v:
            # 如果滑窗内部和超过了v，则我们需要先记录上一个滑窗[l, r-1]的长度
```

```
            # 如果滑窗内部和超过了v，则我们需要先记录上一个滑窗[l, r-1]的长度
            if total - diff[r] <= v:
                ans = max(ans, r - l)
            # 然后由于当前滑窗内部和已经超过了v，因此需要减少滑窗内部和，只能让滑窗左边界+1，内部和减去失去的diff[l]
            total -= diff[l]
            l += 1
            if r < l:
                # 注意左边界右移不能超过右边界，如果超过了，则右边界也需要+1，即变为左边界位置，此时内部和需要加入新右边界值dif
                r = l
                if r < n:
                    total += diff[r]
        else:
            # 如果滑窗内部和没有超过v
            if total == v:
                # 如果滑窗内部和==v,那么当前滑窗就是一个符合要求的，需要记录此时滑窗[l,r]的长度
                ans = max(ans, r - l + 1)
            # 接下来只做滑窗右边界+1，注意右边界不能越界，滑窗需要纳入新右边界值
            # 这里没有做左边界+1 动作，是因为后续的diff有可能都为0，
            # 比如diff = [0, 5, 0, 0, 0], v=5, 当L=0，R=1时，符合当前条件，如果此处做了l++,r++,那么将得不到最大长度
            r += 1
            if r < n:
                total += diff[r]

    # 注意收尾处理，即最后必然是r越界，结束循环，因此最后一轮滑窗范围是[l, r-1]
    return max(ans, r-l)


# 调用算法
print(getResult())
```