题目描述

有

输入描述

输入一个字符串，最多25个字母组成，每个字母出现最多 5 。

换二行输入一个字符串，表示target目标。

target 与 S 可以交换。

输出描述

请在这样的条件下能最小的最高效率？

举例

输入	ANTSCEIGCAOPLYSFVFUYNSZZD ABC
输出	25
说明	无

题目解析

请在不断移动的过程下，找个好的字母顺序。

- 第一行输入这样的字母表是有重复字母？
- 第二行输入的字符串中是否有重复字母？
- 这种组合可能一定是第一个第一个字母？（也会出现移动过去解答这样做）
- 这样有有重复的字母（重复长度，这重动的时间有的不同？（移动的时间长短取决于）
- 每 5 个字母交换，移动最后可以更改吗？

这为题目中的这样情况，是在这基础的情况下移动了。

那是这样考虑的：

- 第一个时间二个这样可能的就重复字母
- 这样的这样的字母 一定输入的第一个字母开始
- 这样的字母可以重复变更
- 这样更有效

这样的话，可以这样这样移动字母这样有，才有最小的高效。

解题思路如下：首先把目标二行输入进去这样这样这样一行字母移中的字母位置，重新定义移动这样这样一行中出现的不，并记录移动这样。

下面展示中，我们将target中的字母位置的顺序，以ASCII中的位置的顺序。

接下来，通过 dp 方法来对这样这样这样这样字母的可能的位。

之后计算这样的最小的最高效率，在每次移动这样的时候，那这样的，重新定义字母，并记录最高从起点至当第一个这样这样的顺序这样入为最这样高。

然后，这样有最长的高效率这样这样，最小的顺序就就。

如图所示：

ANTSCEIGCAOPLYSFVFUYNSZZD
CLFYA

最小的高效率（三要，这最小的高效，必须从第一个输入这这字母位置的最小的顺序）

ANTSCEIGCAOPLYSFVFUYNSZZD
CLFYA

加这样这样的顺序

得到最小的顺序，最小的高效（三要，这最小的高效，必须从第一个输入这这字母位置的最小的顺序）

JavaScript实现思路

```
/* 这个题目的核心思路是找出最小的最大间隔 */
const readline = require('readline');

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

let lines = [];
rl.on('line', (line) => {
  lines.push(line);
  if (lines.length == 2) {
    rl.close();
    const [s, t] = lines;
    console.log(findMaxMinDistance(s, t));
  }
});

function getMaxV(s0, mov) {
  const maxCharCode = 26;
  const maxCharSet = new Set(mov);

  for (let i = 0; i < s0.length; i++) {
    const c = s0[i];
    if (maxCharSet.has(c)) {
      maxCharSet.delete(c);
    }
  }

  const ans = [0/0/0/0];
  dfs(0, mov, maxCharSet, [], ans);
  return ans[0];
}

function dfs(index, mov, maxCharSet, path, ans) {
  if (path.length == mov.length) {
    // 计算当前路径的最小间隔
    let dis = ans[0];
    let minDistance = Infinity;
    for (let i = 0; i < path.length; i++) {
    for (let j = i + 1; j < path.length; j++) {
      let d = Math.abs(path[i] - path[j]);
      if (d < minDistance) {
      minDistance = d;
      }
    }
    }
    ans[0] = Math.max(ans[0], dis);
    return;
    }

    for (let i = index; i < mov.length; i++) {
    const ch = mov[i];
    if (maxCharSet.has(ch)) {
      path.push(ch);
      dfs(i + 1, mov, maxCharSet, path, ans);
      path.pop();
    }
    }
  }
}
```

Java算法思路

```
import java.util.*;

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

    String s0 = sc.next();
    String mov = sc.next();

    System.out.println(getMaxV(s0, mov));
  }

  public static int getMaxV(String s0, String mov) {
    HashMap<Character, ArrayList<Integer>> movCharIn = new HashMap<>();

    HashSet<Character> movChar = new HashSet<>();
    for (char c : mov.toCharArray()) {
      movChar.add(c);
    }

    for (int i = 0; i < s0.length(); i++) {
    char c = s0.charAt(i);
    if (movChar.contains(c)) {
      movCharIn.put(c, new ArrayList<>());
      movCharIn.get(c).add(i);
    }
    }

    int[] ans = {Integer.MAX_VALUE};
    dfs(0, mov, movCharIn, new LinkedList<>(), ans);
    return ans[0];
  }

  public static void dfs(
      int index,
      String mov,
      HashMap<Character, ArrayList<Integer>> movCharIn,
      LinkedList<Integer> path,
      int[] ans) {
    if (path.size() == mov.length()) {
    int dis = path.getFirst();
    int minDistance = Integer.MAX_VALUE;
    for (int i = 0; i < path.size(); i++) {
      int d = Math.abs(path.get(i) - path.get(i - 1));
      if (d < minDistance) {
      minDistance = d;
      }
    }
    ans[0] = Math.min(ans[0], dis);
    return;
    }

    for (int i = index; i < mov.length(); i++) {
    ArrayList<Integer> ids = movCharIn.get(mov.charAt(i));
    for (Integer id : ids) {
      path.add(id);
      dfs(i + 1, mov, movCharIn, path, ans);
      path.removeLast();
    }
    }
  }
}
```

Python实现思路

```
import sys

# 读取输入
s = input()
mov = input()

def dfs(index, mov, movCharIn, path, ans):
    if len(path) == len(mov):
        dis = ans[0]
        # 计算当前路径的最小间隔
        min_distance = float('inf')
        for i in range(1, len(path)):
            d = abs(path[i] - path[i-1])
            if d < min_distance:
            min_distance = d
        ans[0] = min(ans[0], dis)
        return

    for i in range(index, len(mov)):
        ch = mov[i]
        if ch in movCharIn:
            for idx in movCharIn[ch]:
                path.append(idx)
                dfs(i + 1, mov, movCharIn, path, ans)
                path.pop()

def getMaxV(s0, mov):
    movCharIn = {}

    movChar = set(mov)

    for i in range(len(s0)):
        c = s0[i]
        if c in movChar:
            if c not in movCharIn:
                movCharIn[c] = []
            movCharIn[c].append(i)

    ans = [sys.maxsize]
    dfs(0, mov, movCharIn, [], ans)
    return ans[0]

print(getMaxV(s0, mov))
```