给定一个长空字符串 s，其最长子串指的是满足字符都是互不相同的子串的最长长度。

另外，给你一个字符，你需要从最多可以改变 k 个字符的情况下，找出这个最长子串的长度。

换句话说，你可以将字符串中最多 k 个字符从它的原位置替换到其他字母，以形成一个满足字符互不重复的子串，并且这个子串是所有可能的改变后能得到的最长子串之一。

输出这个最长子串的长度。

输入描述

输入为两行，第一行为单个整数 K，第二行为字符串 S。

输出描述

输出最长的字符串。

用例

输入	5 12abc-abCABc-4aB@
输出	12abc-abc-ABC-4aB-@
说明	字串为12abc、abCABc、4aB@，第一个子串胜出， 因为这个子串最少不同一字母为abc、ABc、4aB、@， abC中大写字母较多，需转为abc， ABc中大写字母较多，需改为ABC， 4aB中大写字母偏多为1个，不需编辑。 @中没有字母，海阔天空12abc-abc-ABC-4aB-@

输入	12 12abc-abCABc-4aB@
输出	12abc-abCABc4aB@
说明	字串为12abc、abCABc、4aB@，第一个子串胜出， 因为这个子串最少不同字一母为abcCABc4aB@， 这个子串中大小写字母都为4个，不需编辑， 海阔天空12abc-abCABc4aB@

题目解析

此题应该是考察字符串和编程能力的题，只需要熟悉要掌握字符串操作？即可。

本题比较容易理解，第一个子串是否最长大小写转换？？？是特别的明显事件，应该是不用测试。

下面代码中，判断大小写转换时比较一个石头，考虑时可以找题目行使用。

Java算法源码

```
import java.util.Scanner;
import java.util.StringJoiner;

public class Main {
  static int[] fact;

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

    int k = sc.nextInt();
    String s = sc.next();

    System.out.println(getResult(k, s));
  }

  public static String getResult(int k, String s) {
    String[] arr = s.split("-");

    // 第一个子串不应为空
    StringJoiner sj = new StringJoiner("-");

    // 这个数组用来保存每个子串的长度，最多不超过26，
    int[] lens = new int[26];

    sj.add(arr[0]);

    // 将第一个子串的字母统计一遍
    StringBuilder sb = new StringBuilder();
    for (int i = 1; i < arr.length; i++) sb.append(arr[i]);
    String rawStr = sb.toString();

    for (int i = 0; i < rawStr.length(); i += k) {
      String subStr = rawStr.substring(i, Math.min(i + k, rawStr.length()));
      // 计算出小写字母总数，只需要转为小写字母，大写字母忽略，只需统计转为小写字母
      sj.add(convert(subStr));
    }

    return sj.toString();
  }

  public static String convert(String str) {
    int lowerCount = 0;
    int upperCount = 0;

    for (int i = 0; i < str.length(); i++) {
      char c = str.charAt(i);
      if (c >= 'a' && c <= 'z') lowerCount++;
      else if (c >= 'A' && c <= 'Z') upperCount++;
    }

    if (lowerCount > upperCount) return str.toLowerCase();
    else if (lowerCount < upperCount) return str.toUpperCase();
    else return str;
  }
}
```

JS算法源码

```
// 注意node.js 的输入输出，需要引入模块，如下
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 === 2) {
    let k = parseInt(lines[0]);
    let s = lines[1];
    console.log(getResult(k, s));
    lines.length = 0;
  }
});

function getResult(k, s) {
  const arr = s.split("-");

  const ans = [];
  // 第一个子串不应为空
  ans.push(arr[0]);

  // 将后面的字母统计一遍，统计小写字母
  const maxStr = arr.slice(1).join("");
  for (let i = 0; i < maxStr.length; i += k) {
    const subStr = maxStr.slice(i, i + k);
    // 计算出小写字母总数，只需要转为小写字母，大写字母忽略，只需统计转为小写字母
    ans.push(convert(subStr));
  }

  return ans.join("-");
}

function convert(s) {
  let lowerCount = 0;
  let upperCount = 0;

  for (let i = 0; i < s.length; i++) {
    if (s[i] >= 'a' && s[i] <= 'z') lowerCount++;
    else if (s[i] >= 'A' && s[i] <= 'Z') upperCount++;
  }

  if (lowerCount > upperCount) return s.toLowerCase();
  else if (lowerCount < upperCount) return s.toUpperCase();
  else return s;
}
```

Python算法源码

```
# 输入输出
k = int(input())
s = input()

def convert(s):
    lowerCount = 0
    upperCount = 0

    for c in s:
        if 'a' <= c <= 'z':
            lowerCount += 1
        elif 'A' <= c <= 'Z':
            upperCount += 1

    if lowerCount > upperCount:
        return s.lower()
    elif lowerCount < upperCount:
        return s.upper()
    else:
        return s

# 主函数
def getResult():
    arr = s.split("-")

    ans = []
    # 第一个子串不应为空
    ans.append(arr[0])

    # 将后面的字母统计一遍，统计小写字母
    maxStr = "".join(arr[1:])
    for i in range(0, len(maxStr), k):
        subStr = maxStr[i:i + k]
        # 计算出小写字母总数，只需要转为小写字母，大写字母忽略，只需统计转为小写字母
        ans.append(convert(subStr))

    return "-".join(ans)


# 输出结果
print(getResult())
```