第一章算法实现题

学号：2209060322　　姓名：梁桐　　班级：计算机 2203

1-1

算法实现题 1-1　统计数字问题

★ 问题描述：一本书的页码从自然数 1 开始顺序编码直到自然数 n。书的页码按照通常的习惯编排，每个页码都不含多余的前导数字 0。例如，第 6 页用数字 6 表示，而不是 06 或 006 等。数字计数问题要求对给定书的总页码 n，计算出书的全部页码中分别用到多少次数字 0，1，2，…，9。

★ 算法设计：给定表示书的总页码的十进制整数 n（$1 \leqslant n \leqslant 10^9$），计算书的全部页码中分别用到多少次数字 0，1，2，…，9。

★ 数据输入：输入数据由文件名为 input. txt 的文本文件提供。每个文件只有 1 行，给出表示书的总页码的整数 n。

★ 结果输出：将计算结果输出到文件 output. txt。输出文件共有 10 行，在第 k 行输出页码中用到数字 k-1 的次数，$k = 1,2,\cdots,10$。

输入文件示例	输出文件示例
input. txt	output. txt
11	1 4 1 1 1 1 1 1 1 1

解法一逻辑：

读取整数 n：

从输入文件中读取一个整数 n，表示要统计的页数。

初始化计数器：

定义一个大小为 10 的数组 counts[10]，初始化为 0，用于统计数字 0 到 9 在 1 到 n 中的出现次数。

统计数字出现次数：

使用一个循环，从 1 遍历到 n，对于每一页 page：

使用 current 变量存储当前页面的数字。

通过取模操作 (current % 10) 获取当前数字的最后一位，并将其对应的计数器加 1。

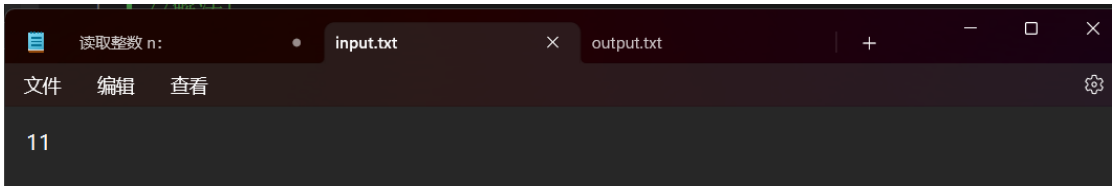
将 current 右移一位（整除 10），继续处理下一位数字，直到 current 变为 0。

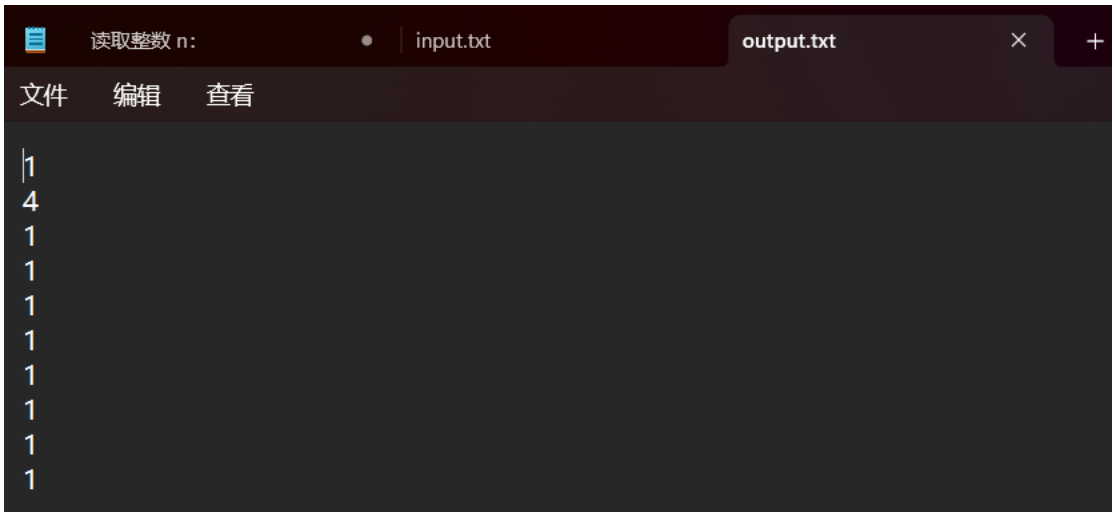
代码：

```
#include <iostream>
#include <fstream>
int main() {
    // 使用原始字符串字面量避免路径转义问题目
```

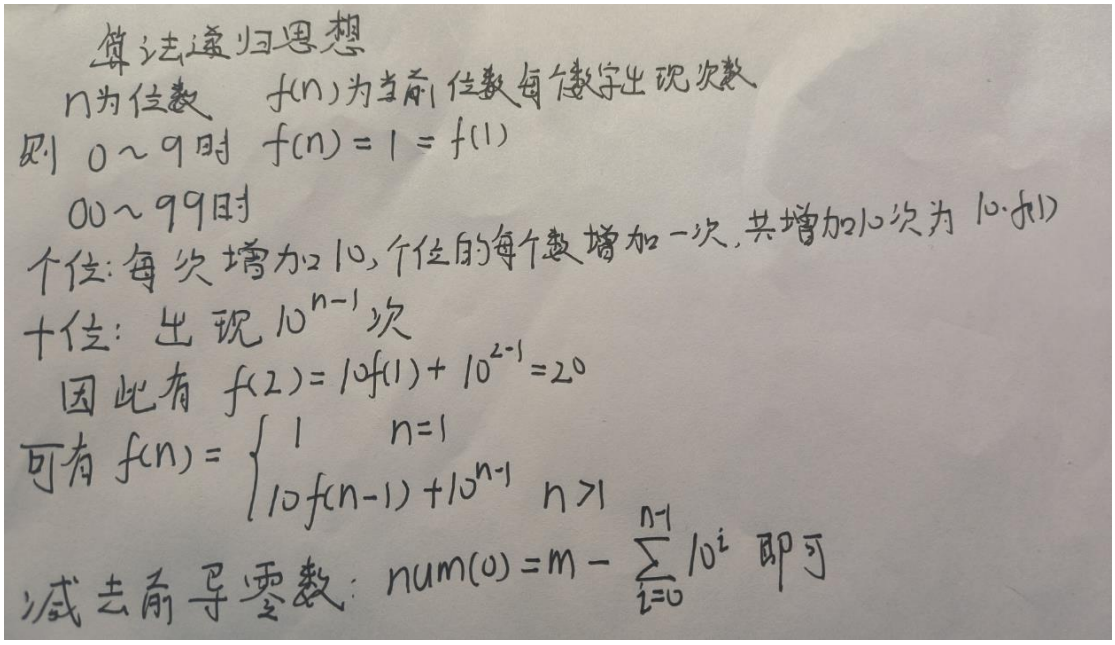
```
    std::ifstream infile(R"(D:\aaqqwenjian\课程\大三上\算法设计与分析\Code\SRFA1_1\input.txt)");
    if (!infile) {
        std::cerr << "无法打开 input.txt 文件" << std::endl;
        return 1;
    }

    int n;
    infile >> n;
    infile.close();

    int counts[10] = {0};

    for (int page = 1; page <= n; ++page) {
        int current = page;
        while (current > 0) {
            int digit = current % 10;
            counts[digit]++;
            current /= 10;
        }
    }

    std::ofstream outfile(R"(D:\aaqqwenjian\课程\大三上\算法设计与分析\Code\SRFA1_1\output.txt)");
    if (!outfile) {
        std::cerr << "无法打开 output.txt 文件" << std::endl;
        return 1;
    }

    // 打印并写入结果到文件
    for (int i = 0; i < 10; ++i) {
        outfile << counts[i] << std::endl; // 写入文件
        std::cout << counts[i] << std::endl; // 打印到控制台
    }

    outfile.close();
    return 0;
}
```

结果：





解法二：
算法递归思想：



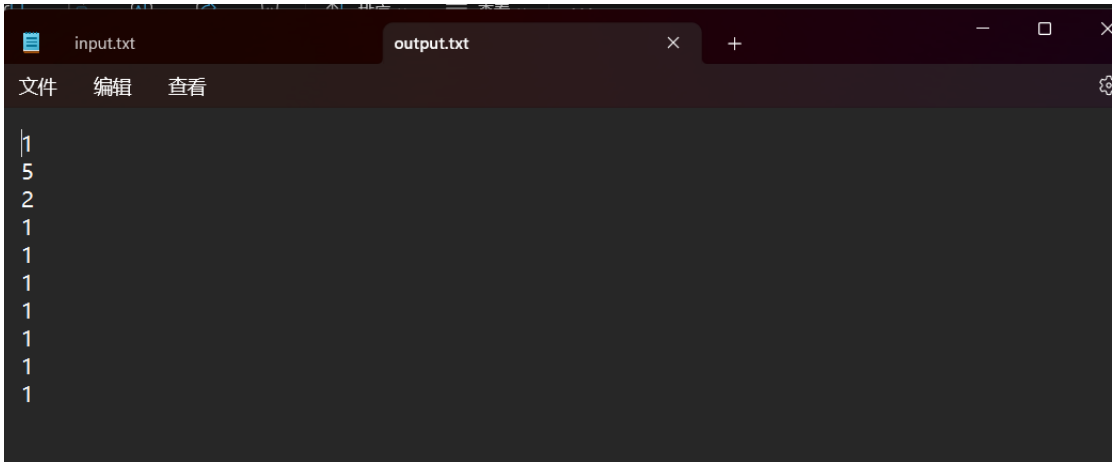
代码：

```
//解法 2
#include <iostream>
#include <cmath>
#include <fstream>
#include <string>
using namespace std;
int c[10];
int get_wei(int n)                  //  获取数字 n 的位数
{
    return n == 0 ? 1 : int(log10(n)) + 1;   //  特殊情况处理
}

int get_max(int n)                  //  获取数字 n 最高位的数字
{
    return n / int(pow(10, get_wei(n) - 1));
}

int get_yu(int n)                   //  获取数字相对于最高位的余数
{
    return n % int(pow(10, get_wei(n) - 1));
}
int get_zero(int m)                 //  获取 m 位数中零的个数
{
    if (m == 1)
    {
        return 1;
    }
    return get_zero(m - 1) + int(pow(10, m - 1));
}
void solve(int n)                   //  解决函数
{
    for (int i = 0; i < 10; i++)            //  计算除最高位之外内位中数字出现的次数
    {
        c[i] += get_max(n) * (get_wei(n) - 1) * int(pow(10, get_wei(n) - 2));
    }
    for (int i = 0; i < get_max(n); i++)    //  计算最高位除最大数字之外其余数出现的次
数
    {
        c[i] += int(pow(10, get_wei(n) - 1));
    }

    c[get_max(n)] += get_yu(n) + 1;       //  加上最高位最大数出现的次数
```

```
int t = get_yu(n);
    if (t == 0) {                                   // 余数为零情况
        c[0] += get_wei(n) - 1;
        return;
    }
    int len = log10(t) + 1;
    if (len != get_wei(n) - 1) {                    // 余数的位数小于最大位数-1 的情况
        c[0] += (get_wei(n) - len - 1) * (t + 1);
    }
    return solve(t);                                // 递归调用解决余数的出现次数
}

int main()
{
    const string inputFilePath = R"(D:\aaqqwenjian\课程\大三上\算法设计与分析\Code\SRFA1_1\input.txt)";
    const string outputFilePath = R"(D:\aaqqwenjian\课程\大三上\算法设计与分析\Code\SRFA1_1\output.txt)";

    // 打开输入文件
    ifstream infile(inputFilePath);
    if (!infile) {
        cerr << "无法打开输入文件: " << inputFilePath << endl;
        return 1;
    }

    // 打开输出文件
    ofstream outfile(outputFilePath);
    if (!outfile) {
        cerr << "无法打开输出文件: " << outputFilePath << endl;
        return 1;
    }

    int n;
    while (infile >> n) {   // 从输入文件读取页码
        for (int i = 0; i < 10; i++) {
            c[i] = 0;   // 清零计数器
        }
        solve(n);
        c[0] -= get_zero(get_wei(n));   // 前导零处理
```

```
// 将结果写入输出文件，每个数字的出现次数单独占一行
            for (int j = 0; j < 10; j++) {
                outfile << c[j] << endl;
            }
        }

        // 关闭文件
        infile.close();
```

结果：



1-2

题目要求:

算法实现题 1-2　字典序问题

★问题描述:在数据加密和数据压缩中常需要对特殊的字符串进行编码。给定的字母表A由 26 个小写英文字母组成,即 $A=\{a,b,\cdots,z\}$。该字母表产生的升序字符串是指字符串中字母从左到右出现的次序与字母在字母表中出现的次序相同,且每个字符最多出现 1 次。例如,a,b,ab,bc,xyz 等字符串都是升序字符串。现在对字母表A产生的所有长度不超过 6 的升序字符串按照字典序排列并编码如下。

1	2	…	26	27	28	…
a	b	…	z	ab	ac	…

对于任意长度不超过 6 的升序字符串,迅速计算出它在上述字典中的编码。

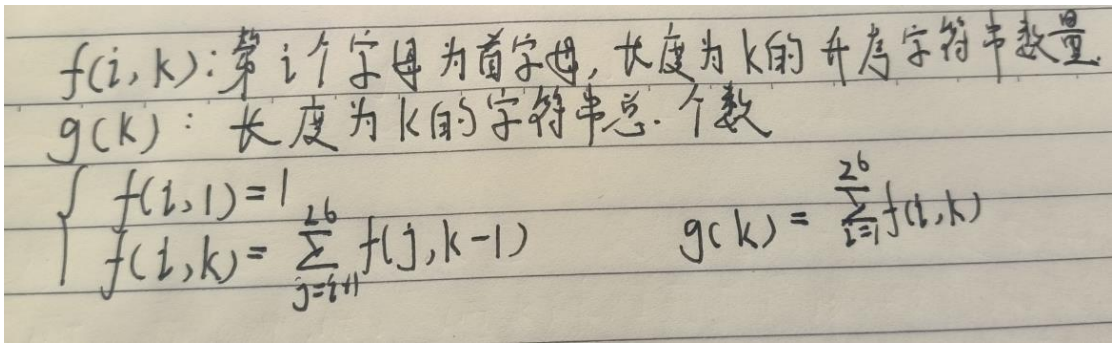
★算法设计:对于给定的长度不超过 6 的升序字符串,计算它在上述字典中的编码。

★数据输入:输入数据由文件名为 input.txt 的文本文件提供。文件的第 1 行是一个正整数k,表示接下来共有k行。在接下来的k行中,每行给出一个字符串。

★结果输出:将计算结果输出到文件 output.txt。文件共有k行,每行对应于一个字符串的编码。

输入文件示例	输出文件示例
input.txt	output.txt
2	1
a	2
b	

算法思想:



代码:

```
#include <iostream>
#include <string>
```

```
#include <fstream>

using namespace std;

// 计算以首字母 i 开头、长度为 k 的升序字符串的数量
int countAscendingStrings(int startLetter, int length) {
	if (length == 1) return 1;

	int total = 0;
	for (int nextLetter = startLetter + 1; nextLetter <= 26; nextLetter++) {
		total += countAscendingStrings(nextLetter, length - 1);
	}
	return total;
}

// 计算长度为 k 的所有升序字符串的总数
int totalAscendingStringsOfLength(int length) {
	int total = 0;
	for (int firstLetter = 1; firstLetter <= 26; firstLetter++) {
		total += countAscendingStrings(firstLetter, length);
	}
	return total;
}

// 判断字符串是否为升序
bool isAscending(const string& str) {
	for (size_t i = 0; i < str.length() - 1; i++) {
		if (str[i] >= str[i + 1]) {
			return false; // 不是升序
		}
	}
	return true; // 是升序
}

// 计算字符串在升序字符串中的位置
int calculatePosition(const string& str) {
	if (!isAscending(str)) {
		return 0; // 非升序字符串返回 0
	}

	int position = 0;
	int length = str.length()
```

```
    // 统计长度小于 str 的所有升序字符串数量
    for (int k = 1; k < length; k++) {
        position += totalAscendingStringsOfLength(k);
    }

    // 处理与 str 相同长度的字符串
    int firstLetter = str[0] - 'a' + 1; // 当前字符串的首字母对应的数字
    for (int i = 1; i < firstLetter; i++) {
        position += countAscendingStrings(i, length);
    }

    for (size_t j = 1; j < length; j++) {
        int currentLetter = str[j] - 'a' + 1;
        for (int i = str[j - 1] - 'a' + 2; i < currentLetter; i++) {
            position += countAscendingStrings(i, length - j);
        }
    }

    return position + 1; // 返回位置，从 1 开始计数
}

int main() {
    ifstream inputFile("input.txt");
    ofstream outputFile("output.txt");

    if (!inputFile.is_open() || !outputFile.is_open()) {
        cerr << "Error opening file!" << endl;
        return 1;
    }

    int testCaseCount;
    inputFile >> testCaseCount;
    while (testCaseCount--) {
        string str;
        inputFile >> str;
        int result = calculatePosition(str);
        if (result == 0) {
            outputFile << "ERROR!" << endl;
        }
        else {
            outputFile << result << endl;
        }
    }
```

```
    inputFile.close();
    outputFile.close();
    return 0;
}
```

代码分析：

核心思路

1. 递归计算数量

使用递归函数 countAscendingStrings 来计算以某个字母为开头、长度为 k 的所有升序字符串的数量。例如，如果首字母是 a，则后续字母可以是 b 到 z 中的任意字母。

2. 统计不同长度的字符串

使用函数 totalAscendingStringsOfLength 计算所有可能长度为 k 的升序字符串的总数。通过遍历可能的首字母从 a 到 z 并调用 countAscendingStrings 来实现。

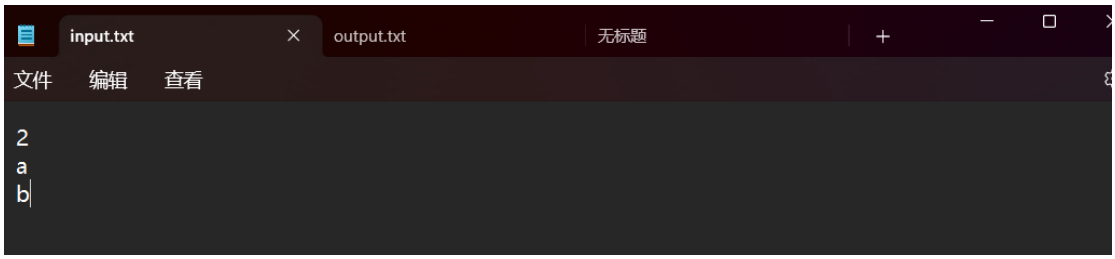
3. 升序判断

使用 isAscending 函数判断给定字符串是否符合升序的条件。如果字符串不符合升序条件，则返回 0。

4. 计算位置

calculatePosition 函数负责计算字符串在所有升序字符串中的位置。首先，统计所有长度小于给定字符串的升序字符串数量。然后，处理与给定字符串相同长度的升序字符串。

运行结果：



1-3

题目：

算法实现题 1-3　最多约数问题

★ 问题描述：正整数 x 的约数是能整除 x 的正整数。正整数 x 的约数个数记为 div(x)。例如，1，2，5，10 都是正整数 10 的约数，且 div(10) = 4。设 a 和 b 是 2 个正整数，$a \leqslant b$，找出 a 和 b 之间约数个数最多的数 x。

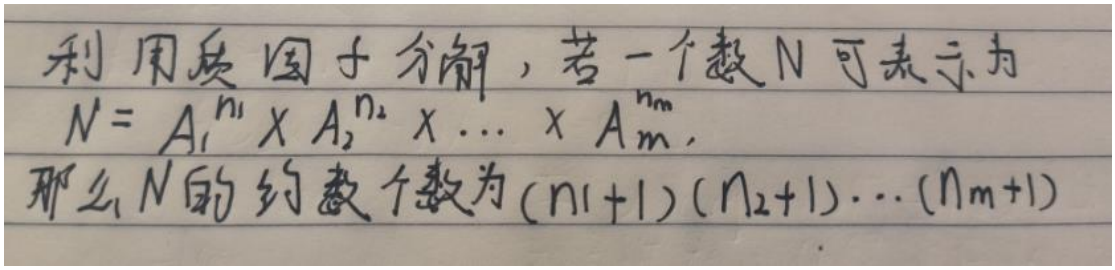
★ 算法设计：对于给定的 2 个正整数 $a \leqslant b$，计算 a 和 b 之间约数个数最多的数。

★ 数据输入：输入数据由文件名为 input. txt 的文本文件提供。文件的第 1 行有 2 个正整数 a 和 b。

★ 结果输出：若找到的 a 和 b 之间约数个数最多的数是 x，则将 div(x) 输出到文件 output. txt。

输入文件示例	输出文件示例
input. txt	output. txt
1 36	9

算法思想：



代码：

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

vector<int> p(10000000, 0);

int prime(int n) {
	int count = 1;
	for (int i = 2; i * i <= n; i++) {
		if (n % i == 0) {
			int factorCount = 0;
			while (n % i == 0) {
				n /= i;
```

```
                factorCount++;
            }
            count *= (factorCount + 1);
        }
    }
    if (n > 1) count *= 2; // n is prime
    return count;
}

int main() {
    int a, b;

    // 从文件中读取输入
    ifstream inputFile("input.txt");
    if (!inputFile) {
        cerr << "Cannot open file input.txt" << endl;
        return 1;
    }
    inputFile >> a >> b;
    inputFile.close();

    int maxCount = 0;
    int maxNum = a;

    for (int i = a; i <= b; i++) {
        int val = prime(i);
        if (val > maxCount) {
            maxCount = val;
            maxNum = i;
        }
    }
    // 将结果写入文件
    ofstream outputFile("output.txt");
    if (!outputFile) {
        cerr << "Cannot open file output.txt" << endl;
        return 1;
    }
    outputFile   << maxCount << endl;
    outputFile.close();

    return 0;
}
```

运行结果：


input.txt
output.txt
文件
编辑
查看
0 36


input.txt
output.txt
文件
编辑
查看
9