**算法分析与设计实验报告**

**第 1 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 邹林壮 | 学号 | 202208040412 | | 班级 | 计算机科学与技术（拔尖班）2201 |
| 时间 | 2024.10.19 | 地点 | 院楼432 | | | |
| 实验名称 | 题 1-2 字典序问题 | | | | | |
| 实验目的 | 通过上机实验，要求掌握字典序问题的问题描述、算法设计思想、程序设计。 | | | | | |
| 实验原理 | 利用分治法查找以某个字符开头，长度为k的字符串个数以及长度为k的字符串个数，并且使用备忘录方法进行空间和时间优化，并计算出程序运行所需要的时间。 | | | | | |
| 实验步骤 | 1. 首先确定输入输出，输入为n，n个升序字符串s，输出为n个编码cal,约束是字符串长度小于等于6 2. 对于任意长度的字符串，我们均可以求出以某个字符开头的字符串数，因此可以求出以某个字符开头的字符串总数，因此设以第i个字符开头的长度为k的升序字符串个数为f(i,k)=f(j,k-1)(i+1<=j<=26),字符串长度为k的字符串有g(k)=f(i,k)(1<=i<=26) 3. 针对某一个具体的字符，比如bcd，如何确定其编码？设其长度为k 首先求解长度小于k的全部字符串，即sum g(j)(1<=j<=k),然后依次对每个字符进行遍历，求以比该字符小的字符开头的字符串个数，得到这些字符串总和sum,最终该字符串的编号为sum+sum g(j)+1,sum+sum g(j)得到了该字符串前面的字符串个数，那么该字符串的编码就需要加1 | | | | | |
| 关键代码 | // 计算 f(i, k)  int f(int i, int k) {  if (k == 1) return 1;  if (memoF[i][k] != -1) return memoF[i][k]; // 如果已计算，直接返回  int l = 0;  for (int j = i + 1; j <= 26; j++) {  l += f(j, k - 1);  }  return memoF[i][k] = l; // 存储计算结果  }  // 计算 g(k)  int g(int k) {  if (memoG[k] != -1) return memoG[k]; // 如果已计算，直接返回  int l = 0;  for (int i = 1; i <= 26; i++) {  l += f(i, k);  }  return memoG[k] = l; // 存储计算结果  }  int cal(string s) {  int n = s.size();  int all = 0; // 初始化 all  for (int i = 1; i < n; i++) {  all += g(i);  }  for (int i = 0, temp = 0; i < n; i++) {  int l = s[i] - 'a' + 1;  for (int j = temp + 1; j < l; j++) {  all += f(j, n - i);  }  temp = l;  }  return all + 1; // +1 for the empty string  } | | | | | |
| 测试结果 | 1)确定正确性：测试了几个案例，主要是衔接部分的内容，如果正确，可以初步判定其正确  IMG_256  经过手动验证，这几个样例正确  并且在网上查到了别人写的代码，写了对拍程序进行判定  IMG_256  2)时间复杂度为O(n \* 26)  f(i, k)函数计算从字母表的第i个字母开始，形成长度为k的序列的数量。对于每个i和k，函数遍历从i+1到26的所有字母，因此对于固定的k，时间复杂度是 O(26)。  由于k的范围是从1到6，对于每个固定的i，f(i, k)的总时间复杂度是 O(26 \* 6)。  g(k)函数计算形成长度为k的序列的总数量，通过遍历所有i从1到26。  对于每个k，g(k)的时间复杂度是O(26)，因为需要对每个i调用f(i, k)。  g(k)被调用了7次（每个k从1到6一次）。  对于输入字符串s的每个字符，cal函数调用g(i)和f(j, n - i)。g(i)的调用次数是n（字符串长度），每次调用的时间复杂度是 O(26)。f(j, n - i)的调用次数取决于s[i]的值，最坏情况下是 O(26)。  空间复杂度O(27\*7),主要由备忘录数组的大小决定，其栈空间最深为O(7)  3)构造了三个规模的数据集：10，10000，1000000  IMG_256 | | | | | |
| 实验心得 | 1. 递归解决问题中存在着重叠子问题，重叠子问题会重复计算子问题，导致时间和空间成本增加，因此可以采用备忘录方法，通过记录每次递归求解的值，在下一次遇到该子问题时可以直接使用前面已经求解的值，从而减小成本。 2. 学习使用验证程序正确性的方法，设计了对拍验证结果，也就是将已有的可能正确的代码与自己的代码，在随机数生成的样本中进行比较，如果全部匹配，可以初步说明自己的答案是正确的，也可以通过几个关键点或者极端情况下的条件，从而判定程序的正确性。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

**问题代码：**

**#include <bits/stdc++.h>**

**using namespace std;**

**// 备忘录数组**

**vector<vector<int>> memoF(27, vector<int>(7, -1)); // f(i, k)**

**vector<int> memoG(7, -1); // g(k)**

**// 计算 f(i, k)**

**int f(int i, int k) {**

**if (k == 1) return 1;**

**if (memoF[i][k] != -1) return memoF[i][k]; // 如果已计算，直接返回**

**int l = 0;**

**for (int j = i + 1; j <= 26; j++) {**

**l += f(j, k - 1);**

**}**

**return memoF[i][k] = l; // 存储计算结果**

**}**

**// 计算 g(k)**

**int g(int k) {**

**if (memoG[k] != -1) return memoG[k]; // 如果已计算，直接返回**

**int l = 0;**

**for (int i = 1; i <= 26; i++) {**

**l += f(i, k);**

**}**

**return memoG[k] = l; // 存储计算结果**

**}**

**int cal(string s) {**

**int n = s.size();**

**int all = 0; // 初始化 all**

**for (int i = 1; i < n; i++) {**

**all += g(i);**

**}**

**for (int i = 0, temp = 0; i < n; i++) {**

**int l = s[i] - 'a' + 1;**

**for (int j = temp + 1; j < l; j++) {**

**all += f(j, n - i);**

**}**

**temp = l;**

**}**

**return all + 1; // +1 for the empty string**

**}**

**int main() {**

**int n=0;**

**cin>>n;**

**while(n--) {**

**string s;**

**cin >> s;**

**cout<<cal(s)<<endl;**

**}**

**}**

**测试时间代码：**

**#include <bits/stdc++.h>**

**#include <chrono>**

**using namespace std;**

**using namespace chrono;**

**// 备忘录数组**

**vector<vector<int>> memoF(27, vector<int>(7, -1)); // f(i, k)**

**vector<int> memoG(7, -1); // g(k)**

**// 计算 f(i, k)**

**int f(int i, int k) {**

**if (k == 1) return 1;**

**if (memoF[i][k] != -1) return memoF[i][k]; // 如果已计算，直接返回**

**int l = 0;**

**for (int j = i + 1; j <= 26; j++) {**

**l += f(j, k - 1);**

**}**

**return memoF[i][k] = l; // 存储计算结果**

**}**

**// 计算 g(k)**

**int g(int k) {**

**if (memoG[k] != -1) return memoG[k]; // 如果已计算，直接返回**

**int l = 0;**

**for (int i = 1; i <= 26; i++) {**

**l += f(i, k);**

**}**

**return memoG[k] = l; // 存储计算结果**

**}**

**int cal(const string &s) {**

**int n = s.size();**

**int all = 0; // 初始化 all**

**for (int i = 1; i < n; i++) {**

**all += g(i);**

**}**

**for (int i = 0, temp = 0; i < n; i++) {**

**int l = s[i] - 'a' + 1;**

**for (int j = temp + 1; j < l; j++) {**

**all += f(j, n - i);**

**}**

**temp = l;**

**}**

**return all + 1; // +1 for the empty string**

**}**

**// 生成随机升序字符串**

**string generateAscendingString(int length) {**

**string s;**

**for (int i = 0; i < length; i++) {**

**char ch = 'a' + rand() % 26;**

**if (s.empty() || ch > s.back()) {**

**s.push\_back(ch);**

**} else {**

**ch = s.back() + 1;**

**if (ch <= 'z') s.push\_back(ch);**

**}**

**}**

**return s;**

**}**

**// 生成测试数据并写入文件**

**void generateTestData(const string &filename, int n) {**

**ofstream fout(filename);**

**fout << n << endl;**

**for (int i = 0; i < n; i++) {**

**int len = rand() % 6 + 1; // 随机生成字符串长度 1-6**

**string s = generateAscendingString(len);**

**fout << s << endl;**

**}**

**fout.close();**

**}**

**// 从文件读取数据**

**vector<string> readTestData(const string &filename) {**

**ifstream fin(filename);**

**int n;**

**fin >> n;**

**vector<string> data(n);**

**for (int i = 0; i < n; i++) {**

**fin >> data[i];**

**}**

**fin.close();**

**return data;**

**}**

**int main() {**

**srand(time(0)); // 初始化随机种子**

**// 数据规模**

**vector<int> sizes = {10, 1000, 1000000}; // 三种不同规模的数据**

**for (int dataSize : sizes) {**

**string filename = "ascending\_strings\_" + to\_string(dataSize) + ".txt";**

**// 生成测试数据并写入文件**

**generateTestData(filename, dataSize);**

**// 从文件读取数据**

**vector<string> data = readTestData(filename);**

**// 计时**

**auto start = high\_resolution\_clock::now();**

**for (const string &s : data) {**

**int result = cal(s);**

**}**

**auto end = high\_resolution\_clock::now();**

**// 记录运行时间**

**auto duration = duration\_cast<milliseconds>(end - start);**

**cout << "处理 " << dataSize << " 个升序字符串的运行时间为: "**

**<< duration.count() << " 毫秒" << endl;**

**cout << "----------------------------------------" << endl;**

**}**

**return 0;**

**}**

**对拍代码：**

**#include <iostream>**

**#include <vector>**

**#include <string>**

**#include <cstdlib>**

**#include <ctime>**

**using namespace std;**

**// 第一个算法实现：组合数方法**

**int C(int n, int m) {**

**int sum1 = 1, sum2 = 1;**

**for (int i = 2; i < m + 1; ++i)**

**sum2 \*= i;**

**for (int i = n; i > n - m; --i)**

**sum1 \*= i;**

**return sum1 / sum2;**

**}**

**int cal1(string code) {**

**int len = code.length(), sum = 0;**

**int code\_num[len];**

**// 1. 将字符串进行编码**

**for (int i = 0; i < len; ++i)**

**code\_num[i] = code[i] - 96;**

**// 2. 统计位数小于len的个数**

**for (int i = 1; i < len; ++i)**

**sum += C(26, i);**

**// 3. 统计位数等于len的字符串到所求之间的个数**

**int start = 1; // 第 x 位的起点，初始化为1**

**for (int i = 0; i < len; ++i) {**

**for (int j = start; j < code\_num[i]; ++j) {**

**sum += C(26 - j, len - i - 1);**

**// 26 - j 表示剩下还有多少字母可以选择**

**// len - i - 1 表示剩下多少位置可以放字母**

**}**

**start = code\_num[i] + 1; // 更新下一位的起点，因为升序，所以至少 + 1**

**}**

**return sum+1;**

**}**

**// 第二个算法实现：递归和备忘录方法**

**vector<vector<int>> memoF(27, vector<int>(7, -1)); // f(i, k)**

**vector<int> memoG(7, -1); // g(k)**

**int f(int i, int k) {**

**if (k == 1) return 1;**

**if (memoF[i][k] != -1) return memoF[i][k];**

**int l = 0;**

**for (int j = i + 1; j <= 26; j++) {**

**l += f(j, k - 1);**

**}**

**return memoF[i][k] = l;**

**}**

**int g(int k) {**

**if (memoG[k] != -1) return memoG[k];**

**int l = 0;**

**for (int i = 1; i <= 26; i++) {**

**l += f(i, k);**

**}**

**return memoG[k] = l;**

**}**

**int cal2(string s) {**

**int n = s.size();**

**int all = 0; // 初始化 all**

**for (int i = 1; i < n; i++) {**

**all += g(i);**

**}**

**for (int i = 0, temp = 0; i < n; i++) {**

**int l = s[i] - 'a' + 1;**

**for (int j = temp + 1; j < l; j++) {**

**all += f(j, n - i);**

**}**

**temp = l;**

**}**

**return all + 1;**

**}**

**// 随机生成升序字符串**

**string generateRandomAscendingString(int maxLength) {**

**int len = rand() % maxLength + 1; // 随机生成长度 1 到 maxLength**

**string s;**

**for (int i = 0; i < len; ++i) {**

**char ch = 'a' + rand() % 26;**

**if (s.empty() || ch > s.back()) {**

**s.push\_back(ch);**

**} else {**

**ch = s.back() + 1;**

**if (ch <= 'z') s.push\_back(ch);**

**}**

**}**

**return s;**

**}**

**int main() {**

**srand(time(0)); // 随机数种子**

**// 测试用例数量**

**int testCases = 100;**

**// 最大字符串长度**

**int maxLength = 6;**

**for (int i = 0; i < testCases; ++i) {**

**string s = generateRandomAscendingString(maxLength);**

**// 分别用两种算法计算结果**

**int result1 = cal1(s);**

**int result2 = cal2(s);**

**// 输出测试字符串和两个算法的结果**

**cout << "测试字符串: " << s << endl;**

**cout << "算法1结果: " << result1 << ", 算法2结果: " << result2 << endl;**

**// 比较两个结果是否一致**

**if (result1 != result2) {**

**cout << "结果不一致！" << endl;**

**break;**

**} else {**

**cout << "结果一致！" << endl;**

**}**

**cout << "-----------------------------------" << endl;**

**}**

**return 0;**

**}**