**《数据结构》课程实践报告**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 23软工 | 姓名 | 王天予 | 学号 | 2362401031 |
| 实验布置日期 | | 2024.11.5 | | 提交  日期 |  | | 成绩 |  |

课程实践实验4：counting numbers

## 一、问题描述及要求

### 1. 问题描述和要求

本实验的目标是基于给定的正整数 n (1 ≤ n ≤ 2001)，通过将不超过 n 一半的整数 m 依次放在 n 的左边，构成新整数 mn，并继续向左添加不超过 m 一半的整数 k，依次类推，生成所有可能的整数。我们需要计算并输出这些整数的总数，同时输出生成的前1000个数。实验中主要使用优先级队列来存储和处理生成的整数，并保证整数按特定顺序输出。

### 2. 功能描述

* **计算整数总数**：递归实现统计所有可能的整数个数。
* **优先级队列实现**：将生成的整数存入优先级队列并按长度和字典顺序排序，确保输出符合要求。

示例：

* 输入：8
* 输出：Total = 10，First 10 numbers: 8, 18, 28, 38, 48, 128, 138, 148, 248, 1248

## 二、概要设计

### 1. 系统功能列表

**优先级队列类**：

* priority\_queue<node>：使用优先级队列管理生成的整数，按长度升序和字典顺序排列。

**辅助函数**：

* myitoa(int x): 将整数转换为字符串。
* fastCount(int u): 递归计算所有可能的整数个数，并使用记忆化递归优化计算效率。
* print(int u, string subfix): 递归生成整数并存入优先级队列，输出前1000个数。

## 三、详细设计

### 1. 数据结构设计

#### 优先级队列

优先级队列用于存储和排序生成的整数，排序规则为：

1. 首先按整数的长度降序排列。
2. 如果长度相同，则按字典顺序降序排列。

### 2. 函数设计

1. **fastCount(int u)**
   * **功能**：递归计算所有可能生成的整数个数，并使用记忆化递归进行优化。
   * **输入**：整数 u
   * **输出**：生成的整数总数
2. **print(int u, string subfix)**
   * **功能**：递归生成所有可能的整数，将其存入优先级队列，输出前1000个数。
   * **输入**：整数 u 和后缀字符串 subfix
   * **输出**：前1000个生成的整数
3. **myitoa(int x)**
   * **功能**：将整数转换为字符串格式。
   * **输入**：整数 x
   * **输出**：对应的字符串

## 四、实验结果

### 测试数据及结果

1. **测试用例 1**
   * 输入：8
   * 输出：  
     Total=10  
     First 10 numbers:  
     8 18 28 38 48 128 138 148 248 1248
   * 结果：通过
2. **测试用例 2**
   * 输入：4
   * 输出：
   * Total=4
   * First 4 numbers:  
     4 14 24 124
   * 结果：通过
3. **测试用例 3**
   * 输入：15
   * 输出：
   * Total=26
   * First 26 numbers:  
     15 115 215 315 415 515 615 715 1215 1315 1415 1515 1615 1715 2415 2515 2615 2715 3615 3715 12415 12515 12615 12715 13615 13715
   * 结果：通过
4. **测试用例 4**
   * 输入：2001
   * 输出：
   * Total=264830889564
   * First 1000 numbers:  
     (略)
   * 结果：通过

### 运行效率分析

1. **时间复杂度**：每个整数最多需遍历其所有可能的前缀组合，时间复杂度为 O(n^2)，对于题目所要求的 n 范围内，这样的复杂度是可接受的。  
   若无需输出每个数字，通过记忆化搜索，问题的总体时间复杂度可进一步降低至, 从而使n=2001的数据可以在1s内完成计算。
2. **空间复杂度**：优先级队列的最大空间为 O(n)，程序的内存消耗相对较低。

## 五、实验分析与探讨

### 1. 测试结果分析

1）**功能验证**：  
测试结果表明，程序能够正确生成所有可能的整数，并使用优先级队列完成排序和输出，符合题目要求。

2）**优先级队列的有效性**：  
优先级队列按指定规则排序整数，使得输出顺序满足题目要求。通过使用优先级队列，可以高效地管理和输出大规模整数集合。

### 2. 实现中的问题与解决方法

1）**递归计算效率**  
描述：初始实现存在重复计算的问题，导致性能较差。  
解决方法：通过记忆化递归优化计算过程，显著提高了效率。

2）**优先级队列的排序规则**  
描述：起初未能正确排序生成的整数，导致输出不符合要求。  
解决方法：调整排序规则，使其按长度和字典顺序排列，保证输出顺序正确。

## 六、总结与反思

本实验通过使用优先级队列管理和排序生成的整数，展示了如何高效地解决复杂的组合生成问题。实验过程中，深刻体会到递归优化和数据结构选择的重要性。在实现中，优先级队列的排序规则对程序的正确性起到关键作用。实验结果验证了程序的正确性和稳定性，但仍有进一步优化空间，如改进递归函数的效率和降低空间复杂度。

**附录：源代码**

1. **实验环境**：

编译环境：TDM-GCC 4.9.2 64bit release

命令行参数：-static-libgcc -std=c++11

2、**目录结构**

（1）Solution.cpp

#include <iostream>

#include <queue>

using namespace std;

const int maxn = 2007;

struct node{

int len;

string val;

node(){}

node(string \_val){

val = \_val;

len = val.length();

}

bool operator < (const node &a) const{

return (len > a.len || (len == a.len && val > a.val));

}

};

priority\_queue<node> pq;

long long f[maxn];

string myitoa(int x){

string ans;

while (x != 0){

ans = (char)(48+x%10) + ans;

x /= 10;

}

return ans;

}

long long fastCount(int u){

if (f[u] != -1) return f[u];

long long ans = 0;

for (int i=1;i<=u/2;i++)

ans += fastCount(i);

f[u] = ans+1;

return f[u];

}

void print(int u, string subfix){

if (pq.size() > 1000) return;

string curSubfix = myitoa(u) + subfix;

pq.push(node(curSubfix));

for (int i=1;i<=u/2;i++)

print(i, curSubfix);

return ;

}

int main(){

int n;cin>>n;

for (int i=0;i<=n;i++) f[i] = -1;

long long ans = fastCount(n);

cout<<"Total="<<ans<<endl;

print(n,"");

cout<<"First "<<min(n,1000)<<" numbers:"<<endl;

while (!pq.empty()){

cout<<pq.top().val<<' ';

pq.pop();

}

return 0;

}