**《数据结构》实验报告1**

**学号：09021227 姓名 ：金桥**

**实验题号：1 实验日期：\_**2022.09.20**\_\_ 实验类型：** 必做

**1．问题描述：**描述实验内容和要求以及需要解决的问题。

**提示：**结合教师课堂讲授内容，仔细分析实验要求，对实验内容进行需求分析。

**实验内容：**

公式1：

公式2：

使用伪随机数rand，随机获得所有系数 并采用多项式结构存储。

对于给定的 值

* 采用迭代法，编写两个函数F1a和F2a，实现两个公式计算方法
* 采用递归法，编写两个函数F1b和F2b，实现两个公式计算方法

令 计算f(x)的值

1. 取 x=1.1
2. 当n取值为 {1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100} 时，分别利用 clock() 函数，获得4个算法在同一计算机上的运行时间，并给出对比表格，每行代表一个算法，每列代表n的一个取值
3. 将2中的对比表格，转为图形可视化的表示
4. 撰写实验报告

**需求分析：**

首先需要编写多项式类，实现对系数的存储，并编写F1a,F2a,F1b,F2b成员函数，实现不同的计算方法。

之后编写f3函数，此函数不是成员函数，通过f3调用F1a,F2a,F1b,F2b并比较其运算所消耗时间，并打印出表格，转为可视化图形。

**2．算法思想：**详细描述解决相应问题所需要的算法设计思想。

多项式存储使用顺序表进行存储，内部使用动态数组实现系数的存储与添加。

F1a,F2a采用迭代法，使用for循环进行计算。

F1b,F2b采用递归法，在函数返回时调用自身进行计算。

**3．功能函数：**描述所设计的功能函数。如果有多个函数，需要描述它们之间的关系。

Polynomial类：

AddTerm: 添加多项式项，通过给定系数与质数添加新项，如果新项次数不高于当前最高项次数则直接将系数相加，否则为系数数组重新分配空间。

F1a: 使用迭代法计算公式1，给定x，返回计算值

F2a: 使用迭代法计算公式2，给定x，返回计算值

F1b: 使用递归法计算公式1，给定x，返回计算值

F2b: 使用递归法计算公式2，给定x，返回计算值

f3: 使用不同的方法计算f3，并将用时打印在输出上。

**4．测试数据：**设计测试数据，或具体给出测试数据。

提示：要求测试数据能全面地测试所设计程序的功能。

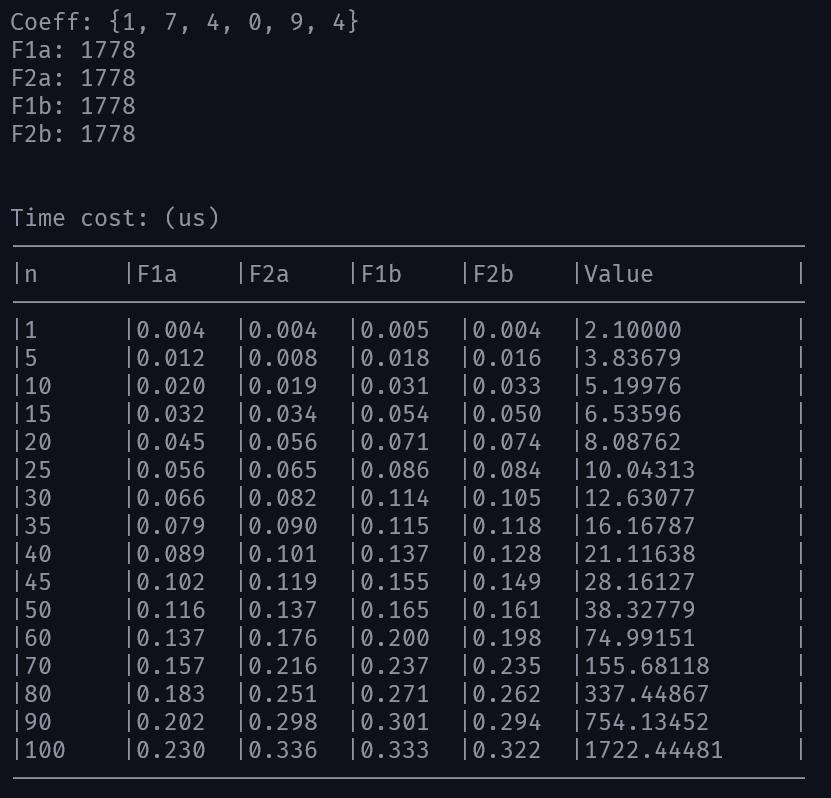
使用伪随机数rand，随机获得所有系数

获得测试参数：1，7，4，0，9，4

取x值为3

对于f3取x=1.1,n={1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100}

**5．测试情况：**给出程序的测试情况，分析运行结果，显示实验结果截图。

****

对于同样的系数与x，F1a,F2a,F1b,F2b计算结果均相同，为1778，结果正确

对于f3,在不同的n的情况下均计算了四个算法的用时并输出了f3的值，结果均正确。

统计四个函数总共的运行用时，得到以下图表：

可以看出使用迭代法时，公式1耗时较少，而在使用递归法时，公式2耗时较少，总体来说，迭代法比递归法用时要少。

**6．实验总结：**写出实验过程中遇到的问题，以及问题的解决过程。分析算法的时间复杂度和空间复杂度，总结实验心得体会。

遇到的问题：在运行程序时，由于处理速度过快导致计算出来的用时均为零

解决方法：运行1000000次计算，从而使得时间可以被计算出来，单位从秒变成微秒。

F1a: 时间复杂度O(n)，空间复杂度O(1)

F2a: 时间复杂度O(n)，空间复杂度O(1)

F1b: 时间复杂度O(n)，空间复杂度O(n)

F2b: 时间复杂度O(n)，空间复杂度O(n)

对于相同时间复杂度的四个算法而言，用时不同的原因我猜测可能为IO方面用时不同。但这四个算法总体上用时是同一个级别的，符合时间复杂度均为O(n)

**7. 源代码：**给出项目所有源程序清单。

建议：源程序中应有充分的注释，例如注释每个函数参数的含义、函数返回值的含义、函数的功能、主要语句段的功能，等等。

实验源程序清单

1. Polynomial.h

#pragma once

#include <iostream>

class Polynomial {

public:

Polynomial(int deg, double\* coef);

Polynomial(int deg);

~Polynomial();

// Add New Term

void AddTerm(double coef, int exp);

// Using iteration

double F1a(double x);

double F2a(double x);

// Using recursion

double F1b(double x, int n = 0, double pow\_x = 1);

double F2b(double x, int n = 0);

private:

// Highest exponent

int m\_degree;

// Coefficients

double\* m\_coef;

};

1. Polynomial.cpp

#include "Polynomial.h"

Polynomial::Polynomial(int deg, double\* coef) : m\_degree(deg) {

// Construct with given degree and coeff

m\_coef = new double[deg + 1];

for (int i = 0; i <= deg; ++i) {

m\_coef[i] = coef[i];

}

}

Polynomial::Polynomial(int deg) : m\_degree(deg) {

// Construct with given degree

m\_coef = new double[deg + 1];

for (int i = 0; i <= deg; ++i) {

m\_coef[i] = rand() % 10;

}

}

Polynomial::~Polynomial() {

// Release memory

delete[] m\_coef;

}

void Polynomial::AddTerm(double coef, int exp) {

if (exp > m\_degree) {

double\* tmp\_coef = new double[exp + 1];

for (int i = 0; i <= m\_degree; ++i)

tmp\_coef[i] = m\_coef[i];

tmp\_coef[exp - 1] += coef;

delete[] m\_coef;

m\_coef = tmp\_coef;

}

else {

m\_coef[exp - 1] += coef;

}

}

// Using iteration

double Polynomial::F1a(double x) {

double res = 0;

double power = 1;

for (int i = 0; i <= m\_degree; ++i) {

res += power \* m\_coef[i];

power \*= x;

}

return res;

}

double Polynomial::F2a(double x) {

double res = m\_coef[m\_degree];

for (int i = m\_degree - 1; i >= 0; --i) {

res \*= x;

res += m\_coef[i];

}

return res;

}

// Using recursion

double Polynomial::F1b(double x, int n, double pow\_x) {

if (n == m\_degree) {

return m\_coef[m\_degree] \* pow\_x;

}

return m\_coef[n] \* pow\_x + F1b(x, n + 1, pow\_x \* x);

}

double Polynomial::F2b(double x, int n) {

if (n == m\_degree) {

return m\_coef[m\_degree];

}

return m\_coef[n] + x \* F2b(x, n + 1);

}

(3) main.cpp

#include <iostream>

#include <random>

#include <ctime>

#include <iomanip>

#include "Polynomial.h"

int n\_num[] = { 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100 };

// Calculation f3 with F1a, F2a, F1b, F2b

// Print time cost

void f3(double x, int n) {

// Load coef

double\* coef\_array = new double[n + 1];

double ret;

coef\_array[0] = 1;

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

coef\_array[i] = double(1) / i;

}

// Init polynomial

Polynomial poly = Polynomial(n, coef\_array);

double startF1a, endF1a, timeF1a;

double startF2a, endF2a, timeF2a;

double startF1b, endF1b, timeF1b;

double startF2b, endF2b, timeF2b;

startF1a = clock();

for (int cnt = 0; cnt < 1000000; ++cnt)

poly.F1a(x);

endF1a = clock();

timeF1a = (endF1a - startF1a) / CLOCKS\_PER\_SEC;

startF2a = clock();

for (int cnt = 0; cnt < 1000000; ++cnt)

poly.F2a(x);

endF2a = clock();

timeF2a = (endF2a - startF2a) / CLOCKS\_PER\_SEC;

startF1b = clock();

for (int cnt = 0; cnt < 1000000; ++cnt)

poly.F1b(x);

endF1b = clock();

timeF1b = (endF1b - startF1b) / CLOCKS\_PER\_SEC;

startF2b = clock();

for (int cnt = 0; cnt < 1000000; ++cnt)

poly.F2b(x);

endF2b = clock();

timeF2b = (endF2b - startF2b) / CLOCKS\_PER\_SEC;

// Print time cost

std::cout << "|" << n << "\t|";

std::cout << std::fixed << std::setprecision(3) << timeF1a << "\t|";

std::cout << std::fixed << std::setprecision(3) << timeF2a << "\t|";

std::cout << std::fixed << std::setprecision(3) << timeF1b << "\t|";

std::cout << std::fixed << std::setprecision(3) << timeF2b << "\t|";

std::cout << std::fixed << std::setprecision(5) << poly.F1a(x) << "\t|" << std::endl;

// Release memory

delete[] coef\_array;

}

int main() {

std::cout << "Coeff: {";

double coef\_1[6];

for (int i = 0; i < 5; ++i) {

coef\_1[i] = rand() % 10;

std::cout << coef\_1[i] << ", ";

}

coef\_1[5] = rand() % 10;

std::cout << coef\_1[5] << "}" << std::endl;

Polynomial test(5);

std::cout << "F1a: " << test.F1a(3) << std::endl;

std::cout << "F2a: " << test.F2a(3) << std::endl;

std::cout << "F1b: " << test.F1b(3) << std::endl;

std::cout << "F2b: " << test.F2b(3) << std::endl;

std::cout << std::endl << std::endl;

std::cout << "Time cost: (us)" << std::endl;

std::cout << "---------------------------------------------------------" << std::endl;

std::cout << "|n\t|F1a\t|F2a\t|F1b\t|F2b\t|Value\t\t|" << std::endl;

std::cout << "---------------------------------------------------------" << std::endl;

for (int i = 0; i < 16; ++i)

f3(1.1, n\_num[i]);

std::cout << "---------------------------------------------------------" << std::endl;

}