# 算法设计与分析实验报告


实验题目：利用 FFT 算法改进大整数乘法的算法效率

姓名：马天成

学号：2020211376

日期：2022-10-18

# 目录

# 一、实验环境

## 1.1 设备规格

### 设备规格

Legion R7000P2020H

| | |
|---|---|
| 设备名称 | PC |
| 处理器 | AMD Ryzen 7 4800H with Radeon Graphics<br>2.90 GHz |
| 机带 RAM | 16.0 GB (15.9 GB 可用) |
| 设备 ID | 370678B7-1FA2-4C35-8BAB-F92D3429406B |
| 产品 ID | 00342-35932-44511-AAOEM |
| 系统类型 | 64 位操作系统, 基于 x64 的处理器 |
| 笔和触控 | 为 10 触摸点提供笔和触控支持 |

## 1.2 操作系统

### Windows 规格

| | |
|---|---|
| 版本 | Windows 10 家庭中文版 |
| 版本号 | 21H2 |
| 安装日期 | 2022/4/5 |
| 操作系统内部版本 | 19044.2006 |
| 序列号 | PF2660CA |
| 体验 | Windows Feature Experience Pack 120.2212.4180.0 |

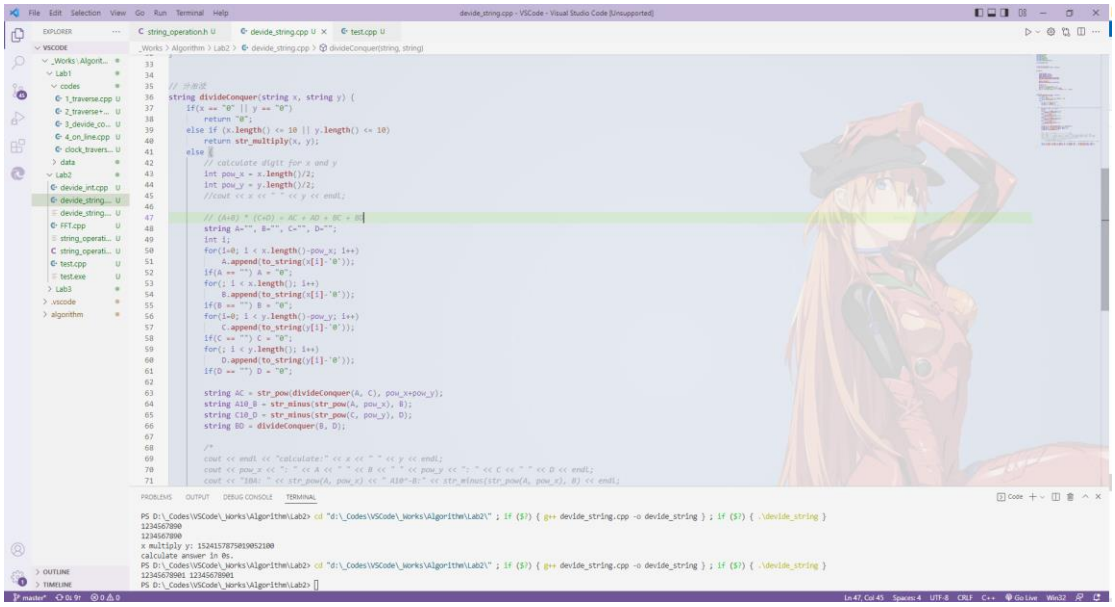
## 1.3 编程语言&编译器

C++11

```
version : MinGW-W64-builds-4.3.5
user    : nixman
date    : 05.12.2018-10:29:36 AM
```

## 1.4 开发工具

VSCode



# 二、实验内容

## 2.1 实验内容及要求

1. 算法的设计与实现
学习 FFT（快速傅里叶变换算法）的基本原理，基于快速傅里叶变换算法改进传统的大整数乘法分治算法，即将基于分治的大整数乘法的算法效率由 O(n 1.59)提高到 O(nlogn)。

2. 测试要求
设计测试数据集，编写测试程序，用于测试：
a) 正确性：所实现的三种算法的正确性；
b) 算法复杂性：针对两种算法，设计测试数据集，评价各个算法在算 法复杂性上的表现；（最差情况、平均情况）

3. 撰写评价报告
a) 结合实验结果，在理论上给予总结和评价两种算法在算法复杂性和效率上的表现。形成电子版实验报告。

4. 作业清单
①程序（算法程序和测试程序）；
②测试数据集和测试程序；

## 2.2 基于传统分治的大整数乘法算法

### 2.2.1 原理

分治法推导大整数乘法公式


$$XY = 2AC * 10^{xn0+yn0} + \left(A * 10^{xn0} - B\right)\left(D - C * 10^{yn0}\right) + 2BD$$

### 2.2.2 数据结构

大整数一定是由字符串来存的。这样才能实现大整数乘法。

这里限制所有的输入都是正数。

这里就采用递归的手法了。非递归属是有点折磨人。虽然说写肯定能写，但是时间一定是来不及了。

### 2.2.3 字符串计算

这里提供了五个函数:

```
// string add
string str_add(string a, string b);
// string minus ( a>b )
string str_minus(string a, string b);
// string exp
string str_pow(string a, int exp);
// character multiply string
string char_str_multiply(char ch, string a);
// string multiply
string str_multiply(string a, string b);
```

功能如注释所写。

**注意的地方：**

0. 所有地方注意 0 输入
1. 加法注意进位
2. 减法注意借位和去除前导 0
3. 乘法注意 pow

### 2.2.3 测试

- 最好最坏情况测试

假如我们采用这种方式

```
 */
 return str_minus( str_add( str_add(AC, AC), str_add(BD, BD) ) , divideConquer(A10_B, C10_D) );
 return str_minus( str_add( str_add(AC, AC), str_add(BD, BD) ) , str_multiply(A10_B, C10_D) );
```

那么这两个乘法也会采用 divide 的方法。

```
PS D:\_Codes\VSCode\_Works\Algorithm\Lab2> cd
1234567890
1234567890
x multiply y: 1524157875019052100
calculate answer in 0s.
PS D:\_Codes\VSCode\_Works\Algorithm\Lab2> cd
12345678901 12345678901
PS D:\_Codes\VSCode\_Works\Algorithm\Lab2>
```

那么十位数就会爆掉。

但假如用这个就不会有太大的问题。因为他是指数级的减小规模。

```
 */
 return str_minus( str_add( str_add(AC, AC), str_add(BD, BD) ) , str_multiply(A10_B, C10_D) );
 return str_minus( str_add( str_add(AC, AC), str_add(BD, BD) ) , divideConquer(A10_B, C10_D) );
```

- 平均情况

基本都可以在 0 秒内解决。（主要是过大的输入会直接爆栈）

```
PS D:\_Codes\VSCode\_Works\Algor
123456789 123456789
x multiply y: 15241578750190521
calculate answer in 0s.
```

## 2.3 FFT

### 2.3.1 原理

1. 设 N 点序列 x(n)，将 x(n)按奇偶分组：

$$X(2l)=\sum_{n=0}^{\frac{N}{2}-1}[x(n)+x(n+\frac{N}{2})]W_N^{2ln}$$

$$X(2l+1)=\sum_{n=0}^{\frac{N}{2}-1}[x(n)-x(n+\frac{N}{2})]W_N^{2ln}W_N^{n}$$

$$l=0,1,\ldots,\frac{N}{2}-1$$

2. 改写为：

$X_1(K)+W_N^k X_2(K), 0 \le k \le \frac{N}{2}-1$

3. 一个 N 点 DFT 分解为两个 N/2 点的 DFT:

$X_1(K)$和$X_2(K)$

4. 继续分解，迭代下去，其运算量约为

$\frac{N}{2}\log_2 N$

**组合数基四 FFT**

N≠$2^M$时，可采取补零使其成为 N=2，或者先分解为两个 p、q 的序列，其中 p*q=N，然后进行计算。

## 2.3.2 代码

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<time.h>

using namespace std;

const int MAXN = 1e7 + 10;
inline int read()
{
    char c = getchar(); int x = 0, f = 1;
    while (c < '0' || c > '9') {
        if (c == '-')f = -1;
        c = getchar(); }
    while (c >= '0' && c <= '9') {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x * f;
}
const double Pi = acos(-1.0);


struct complex
{
    double x, y;
    complex(double xx = 0, double yy = 0) { x = xx, y = yy; }
} a[MAXN], b[MAXN];
```

```
complex operator + (complex a, complex b) {
    return complex(a.x + b.x, a.y + b.y);
}
complex operator - (complex a, complex b) {
    return complex(a.x - b.x, a.y - b.y);
}
complex operator * (complex a, complex b) {
    return complex(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
}

int N, M;
int l, r[MAXN];
int limit = 1;

void fast_fast_tle(complex* A, int type)
{
    //Find out the sequence to iterate
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]);
    for (int mid = 1; mid < limit; mid <<= 1) { //Half the length of the interval to
be combined
        complex Wn(cos(Pi / mid), type * sin(Pi / mid)); //Unit root
        for (int R = mid << 1, j = 0; j < limit; j += R) { //R is the length of the
interval, and j is where it has been before
            complex w(1, 0); //power
            for (int k = 0; k < mid; k++, w = w * Wn) { //Enumerate the left half
                complex x = A[j + k], y = w * A[j + mid + k];
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
        }
    }
}
int main() {
    // read data
    cout << "Please input two numbers started with bit\n";
    int N = read(), M = read();
    for (int i = 0; i <= N; i++) a[i].x = read();
    for (int i = 0; i <= M; i++) b[i].x = read();

    // start timer
    clock_t start, end;
```

```
    start = clock();
    while (limit <= N + M) limit <<= 1, l++;
    for (int i = 0; i < limit; i++)
        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));

    // start fft
    fast_fast_tle(a, 1);
    fast_fast_tle(b, 1);
    for (int i = 0; i <= limit; i++)
        a[i] = a[i] * b[i];
    fast_fast_tle(a, -1);
    string temp;
    int* ntemp = new int[N + M + 2];
    for (int i = 0;i < N + M + 2;i++)
        ntemp[i] = 0;
    int t = 0;
    for (int i = N + M; i >=0 ; i--) {
        t= (int)(a[i].x / limit + 0.5);
        ntemp[i] = (ntemp[i+1]+t) / 10;
        ntemp[i+1] = (ntemp[i+1] + t) % 10;
    }
    bool flag = true;
    for (int i = 0;i < N + M + 2;i++) {
        if (flag)
            ntemp[i] != 0 ? flag = false : flag = true;
        if(!flag)
            temp += ntemp[i]+'0';
    }

    // output time and answer
    cout << temp<<endl;
    end = clock();
    cout << "time = " << double(end - start) / CLOCKS_PER_SEC << "s" << endl;
    return 0;
}
```

### 2.3.3 测试数据

```
PS D:\_Codes\VSCode\_Works\Algorithm\Lab2>
Please input two numbers started with bit
4 5
1 2 3 4 5 6 7 8 9 10 11
8382390795
time = 0.001s
```

```
PS D:\_Codes\VSCode\_Works\Algorithm\Lab2> cd "d:\_Code
Please input two numbers started with bit
20 20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
1524157902758725803932632220698064319464 1
time = 0s
PS D:\ Codes\VSCode\ Works\Algorithm\Lab2>
```

只能说 FFT 算法没有爆栈速度也快，非常美丽。

# 三、出现问题及解决

### 1. 编写字符串功能

在编写字符串功能的时候，至少花费了 6h。主要是经过各种各样的测试才能保证自己的算法是天衣无缝的。其中主要是 0 的处理以及倒转进位借位要处理，所以才会显得比较复杂。

### 2. 经典分治的实现

在实现的时候，我发现无论是边界的处理还是分治的位数处理都需要小心。比如 2 位数出现的 0，或者因为边界值过小导致的问题规模太大而导致时间复杂度巨高。

### 3. FFT 的处理

其中主要是罩着老师给的算法嗯看，看不懂也得写。所以还是非常折磨人的。主要就是写转换的时候需要很多心思，但有了老师给的算法，那就比较简单了。

# 四、总结

## 4.1 时间复杂度空间复杂度

- 经典的分治算法
  第一种写法：

```
return str_minus( str_add( str_add(AC, AC), str_add(BD, BD) ) , divideConquer(A10_B, C10_D) );
return str_minus( str_add( str_add(AC, AC), str_add(BD, BD) ) , str_multiply(A10_B, C10_D) );
```

这个显然复杂度很高，我一开始甚至认为是指数型的。
理想情况下：x、y 位数相同
XY=AC*10n+((A-B)(D-C)+AC+BD)*10n/2+BD

$$T(n) = \begin{cases} O(1) & n = 1 \\ 3T(n/2) + O(n) & n > 1 \end{cases}$$

看起来似乎更复杂，其实只需要做三次 n/2 为整数的乘法。T(n)=O（n^log3）
非理想，则 x，y 位数不同。

空间复杂度也很高，每一个栈都需要与输入规模成正比的线性空间。

- FFT

1. 设 N 点序列 x(n)，将 x(n)按奇偶分组：

$$X(2l) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x(n+\frac{N}{2})] W_N^{2ln}$$

$$X(2l+1) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) - x(n+\frac{N}{2})] W_N^{2ln} W_N^{n}$$

$$l = 0, 1, \ldots, \frac{N}{2} - 1$$

2. 改写为：

$$X_1(K) + W_N^k X_2(K), 0 \le k \le \frac{N}{2} - 1$$

3. 一个 N 点 DFT 分解为两个 N/2 点的 DFT：

$X_1(K)$和$X_2(K)$

4. 继续分解，迭代下去，其运算量约为

$$\frac{N}{2} \log_2 N$$

## 4.2 实验理解

首先是经典的分治算法。其实我一开始写了个 int 或者 long 型的，发现根本就是。。。天方夜谭。这能叫大整数？所以只好乖乖改成字符串。

在这之后，我又进行了字符串计算的编写。这个是比较耗时间的，因为我们这个东西确实是很多 bug 需要处理。从 0 的处理到位数的处理到分治的处理，都需要话一定时间来进行 debug。

接下来是 FFT。FFT 老师上课也描述过一遍了，印象很深，但压根没怎么听懂哈哈哈。因为确实接受起来比较难。

我们需要明白：FFT 算法实质上就是 DFT 算法的改良版，而 DFT 算法则是傅里叶变换的离散版。按傅里叶变换→DFT→FFT 的思路推导，即可理解 FFT。

在经历过这个算法的熏陶过后。我充分了解了算法的美丽。他居然能把我们认为的很普通的高复杂度问题简化成如此简单的一个问题。简直是太美妙了。这个分治的思想和点线巧妙的转换思路，我们也要灵活运用在以后的方方面面。