

实验报告

杨凌林 PB17000083

实验报告

实验设备与环境

实验设备

编译环境

实验内容和要求

实验一（矩阵链乘）

实验二（FFT）

实验方法和步骤

实验一（矩阵链乘）

几点注意

实验二（FFT）

实验结果和分析

实验一（矩阵链乘）

实验截图

实验结果分析

实验二（FFT）

实验截图

实验结果分析

实验总结

实验设备与环境

实验设备

系统版本	Windows 10 专业版
系统版本号	2004
计算机型号	Asus U4100U
处理器	Core i7-8550U

编译环境

IDE	CLion(2020.2.4)
环境	MinGW
编译器	gcc(9.2.0)
CMake	Bundled(3.17.3)

实验内容和要求

实验一（矩阵链乘）

从给定文件读入矩阵链的长度和矩阵大小，运用动态规划法求解最佳链乘方案。并将最佳链乘方案和最小乘法运算次数，以及运行时间写入文件。最后，需要画出运行时间 - 问题规模曲线做分析。

实验二（FFT）

从给定文件读入不同规模多项式的系数，使用FFT算法求解多项式的离散傅里叶变换，并将结果和运行时间写入文件。最后，需要对运行时间画图分析。

实验方法和步骤

实验一（矩阵链乘）

src 文件夹中文件和函数的组织形式如下：

```
1 main.c
2     int main()
3     void print_optimal(int **s, int i, int j, FILE *f)
4         // 将从第 i 到第 j 位矩阵的最佳链乘方案写入文件 f
5 matrix_chain.h
6     void matrix_chain_order(long long *p, long long **m, int **s, int n)
7         // 求解问题规模记录在 p 的矩阵连乘问题
8         // 最佳乘法次数记录在二维数组 m，最佳分割下标记录在二维数组 s
9         // n 是矩阵链长度
```

input.txt 文件包含 5 种规模的问题，故在 main 函数中，通过一个 for 循环对 5 种规模的问题进行处理。

在输出最佳链乘方案时，通过调用函数 print_optimal 实现。

通过三个文件指针，分别实现 input.txt 读入、result.txt 写入、time.txt 写入。

几点注意

- result.txt 文件仅含最小乘法次数和最佳链乘方案（括号表示），为输出类似 P214 图 15-5 的结果，在主函数 for 循环内，对 i = 0 的情形单独处理输出 m 和 s 表。
- 为防止数据溢出，在实验中对乘法次数使用 long long 数据类型。

实验二（FFT）

src 文件夹中文件和函数的组织形式如下：

```
1 main.c
2     int main()
3 FFT.h
4     struct comp // 定义一个结构体，表示复数
5     struct comp comp_plus(struct comp x, struct comp y)
6         // 输入两个复数 x, y, 返回复数 x+y
7     struct comp comp_subt(struct comp x, struct comp y)
8         // 输入两个复数 x, y, 返回复数 x-y
9     void comp_value(struct comp *x, struct comp y)
10        // 将复数 y 的数值存储到 x 对应地址
```

```

11     struct comp comp_multi(struct comp x, struct comp y)
12     // 输入两个复数 x, y, 返回复数 x*y
13     int recursive_FFT(struct comp *data, struct comp *y, int n)
14     // 输入 data 为一多项式的系数表示(复数), 返回 y, 表示 data 的 DFS
15     // n-1 为多项式的次数, n 为 2 的幂

```

由于FFT涉及复数运算，故在 `FFT.h` 文件中定义了 `comp` 结构体表示复数，包含 `re` 和 `im` 两个成员分别表示该复数的实部和虚部。

同样的，`input.txt` 文件包含 6 种规模的问题，故在 `main` 函数中，通过一个 `for` 循环对 6 种规模的问题进行处理。

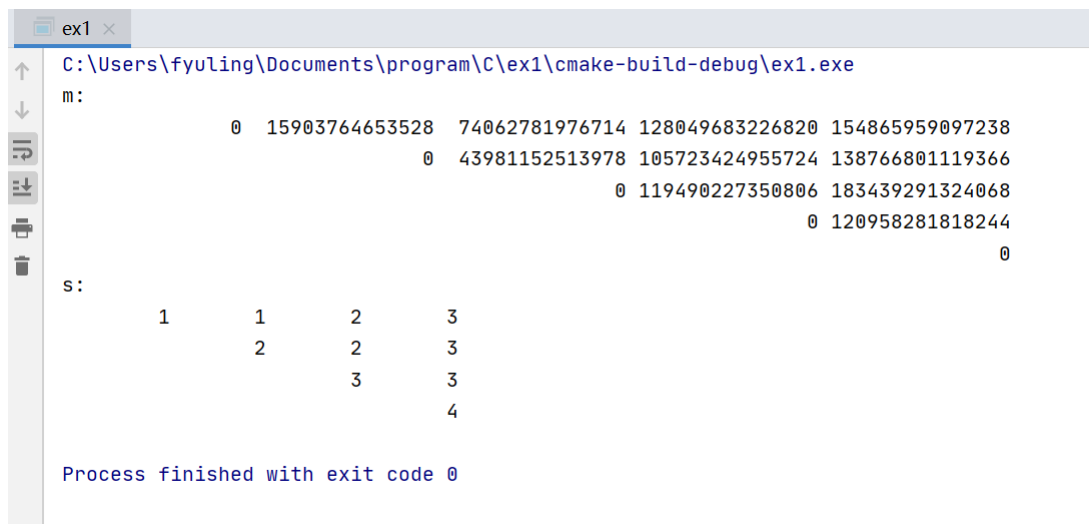
通过三个文件指针，分别实现 `input.txt` 读入、`result.txt` 写入、`time.txt` 写入。

实验结果和分析

实验一（矩阵链乘）

实验截图

$n = 5$ 时实验结果截图如下：



```

C:\Users\fyuling\Documents\program\C\ex1\cmake-build-debug\ex1.exe
m:
      0  15903764653528  74062781976714  128049683226820  154865959097238
      0  43981152513978  105723424955724  138766801119366
      0  119490227350806  183439291324068
      0  120958281818244
      0
s:
      1      1      2      3
      2      2      3
      3      3
      4
Process finished with exit code 0

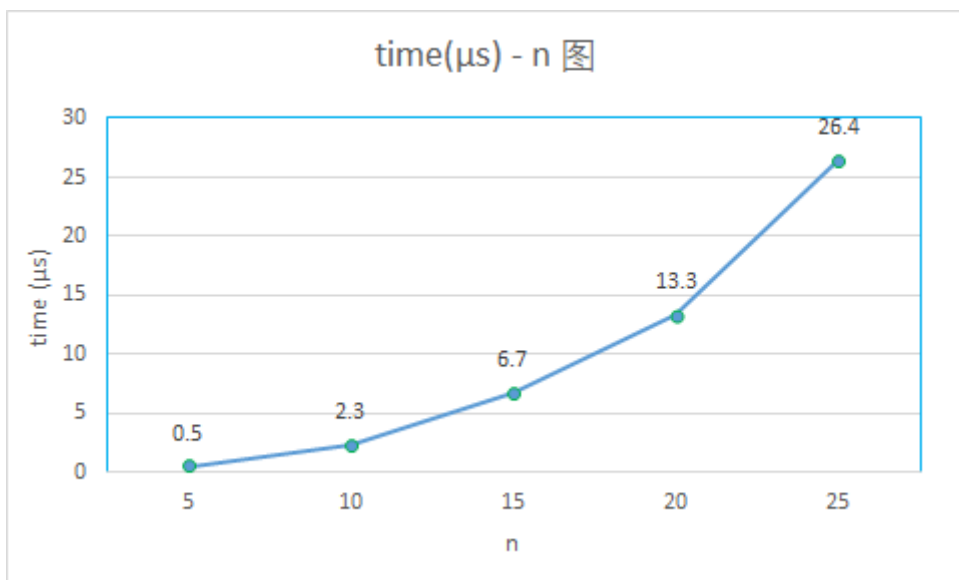
```

运行时间截图如下：

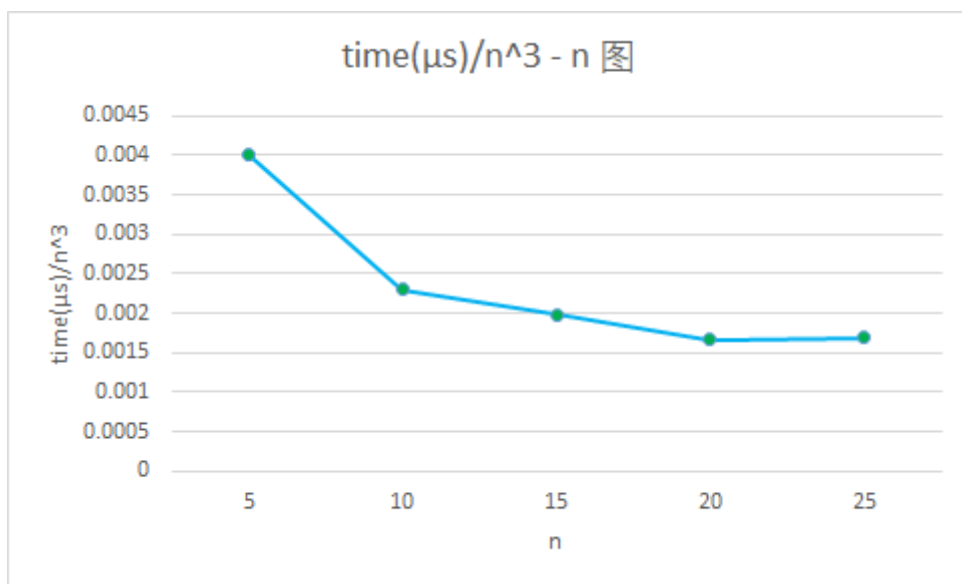
time.txt	
1	0.50 μs
2	2.30 μs
3	6.70 μs
4	13.30 μs
5	26.40 μs

实验结果分析

对实验运行时间做图（`time - n` 图）：



非线性关系较为明显，为了验证课本上所述的 $\mathcal{O}(n^3)$ 复杂度关系，我们可接着做做 $\text{time} / n^3 - n$ 图：

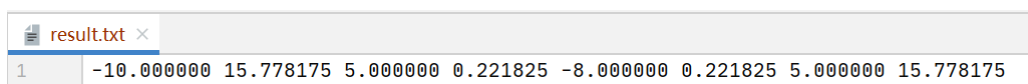


观察图像可发现，随着 n 的增大， time/n^3 接近一条直线，这说明 time/n^3 是渐进趋于一个常数，即算法复杂度为 $\mathcal{O}(n^3)$ 。

实验二 (FFT)

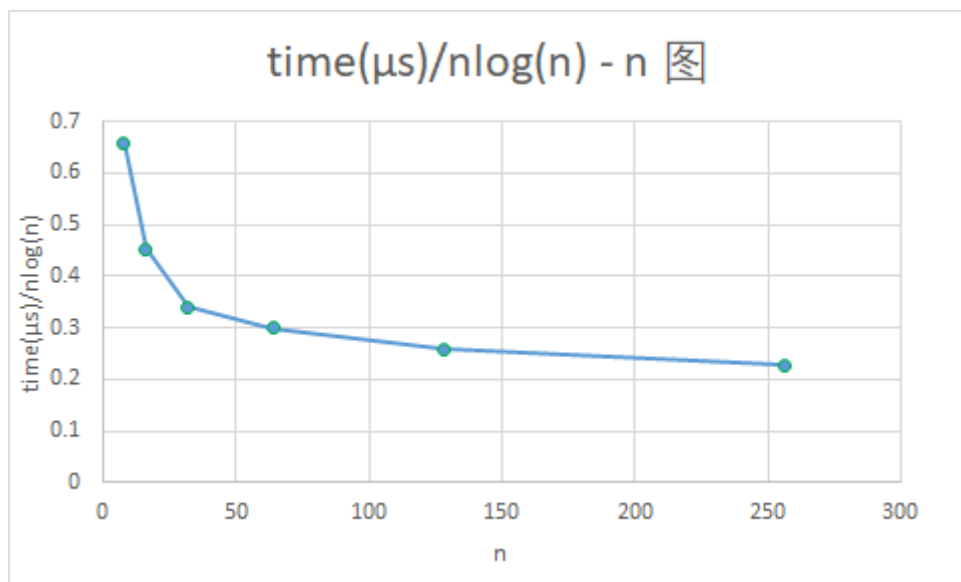
实验截图

$n = 2^3$ 时结果截图：



实验结果分析

FFT 算法理论复杂度为 $\Theta(n \log(n))$ ，故我们直接做 $\text{time}/n \log(n) - n$ 图，图像如下：



可看到，随着 n 的增大， $time/n \log(n)$ 曲线趋于平缓，这就验证了 FFT 算法的理论复杂度 $\Theta(n \log(n))$ 是正确的。

实验总结

通过对实验的运行时间作图，我们可以验证两个算法的时间复杂度是正确的。

实验一通过动态规划法求解矩阵链乘问题，实验二通过巧妙的数学构造求解 DFT，都是利用了精巧的方法，对复杂的问题优化得到复杂度显著降低的算法，从而能在更短时间内解决问题。

我们在以后的学习中应该深刻理解这两种算法的思想，并应用到实际中。