

算法实验二

Pb18081616 谭园

一、实验要求

实验 2.1: 求矩阵链乘最优方案

n 个矩阵链乘, 求最优链乘方案, 使链乘过程中乘法运算次数最少。

n 的取值 5, 10, 15, 20, 25, 矩阵大小见 2_1_input.txt。

求最优链乘方案及最少乘法运算次数, 记录运行时间, 画出曲线分析。

仿照 P214 图 15-5, 打印 $n=5$ 时的结果并截图。

实验 2.2: FFT

多项式 $A(x) = \sum_{i=0}^{n-1} a_i x^i$, 系数表示为 $(a_0, a_1, \dots, a_{n-1})$ 。

n 取 $2^3, 2^4, \dots, 2^8$, 不同规模下的 A 见 2_2_input.txt。

用 FFT 求 A 在 $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$ 处的值。

记录运行时间, 画出曲线分析; 打印 $n=2^3$ 时的结果并截图。

二、实验环境

编译环境: DEV C++

机器内存: 16GB

时钟主频: 2.3GHz

三、实验过程 (核心代码)

矩阵链乘

```
//矩阵链乘动态规划算法
void matrix_chain_order(LONGLONG *A, LONGLONG **m, int **s, int plength){
    int n=plength-1;
    int i,j,k,l;
    LONGLONG q;
    for (i=1;i<=n;i++) m[i][i]=0;
    for (l=2;l<=n;l++){
        for(i=1;i<=n-l+1;i++){
            j=i+l-1;
            m[i][j]= MAXLONGLONG;
            for(k=i;k<=j-1;k++){
                q=m[i][k]+m[k+1][j]+A[i-1]*A[k]*A[j];
                if (q<m[i][j]){
                    m[i][j]=q;
                    s[i][j]=k;
                }
            }
        }
    }
}
```

FFT: 定义好四则运算, 然后递归 fft。

```
1 1= complex add(complex a,complex b){
2     complex c;
3     c.real=a.real+b.real;
4     c.imag=a.imag+b.imag;
5     return c;
6 }
7
8 1= complex sub(complex a,complex b){
9     complex c;
10    c.real=a.real-b.real;
11    c.imag=a.imag-b.imag;
12    return c;
13 }
14
15 1= complex mul(complex a,complex b){
16    complex c;
17    c.real=a.real*b.real-a.imag*b.imag;
18    c.imag=a.imag*b.real+a.real*b.imag;
19    return c;
20 }
```

```

77 //递归FFT
78 complex* RECURSIVE_FFT(complex *A,int n){
79     if(n==1) return A;
80     complex w,wn;
81     w.real=1;
82     w.imag=0;
83     wn.real=cos(2*M_PI/n);
84     wn.imag=sin(2*M_PI/n);
85     complex *A0,*A1;
86     A0=(complex *)malloc(sizeof(complex)*(n/2));
87     A1=(complex *)malloc(sizeof(complex)*(n/2));
88     int i;
89     for(i=0;i<n;i++){
90         if(i%2==0) A0[i/2]=A[i];
91         else A1[i/2]=A[i];
92     }
93     complex *Y0,*Y1;
94     Y0=RECURSIVE_FFT(A0,n/2);
95     Y1=RECURSIVE_FFT(A1,n/2);
96     complex *Y;
97     Y=(complex *)malloc(sizeof(complex)*n);
98     for(i=0;i<n/2;i++){
99         Y[i]=add(Y0[i],mul(w,Y1[i]));
100        Y[i+n/2]=sub(Y0[i],mul(w,Y1[i]));
101        w=mul(w,wn);
102    }
103    return Y;
104 }

```

四、实验结果：单位 (us)

N=5 时的结果

```

C:\Users\15564\Desktop\2020算法\lab2\123-谭园-PB18081616-project2\ex1\src\multiply1.exe
m row from 1 to 5, column from 5 to 1
154865959097238 128049683226820 74062781976714 15903764653528 0
138766801119366 105723424955724 43981152513978 0
183439291324068 119490227350806 0
120958281818244 0
0

s row from 1 to 4, column from 5 to 2
1 1 1 1
4 3 2
4 3
4
请按任意键继续. . .

```

```

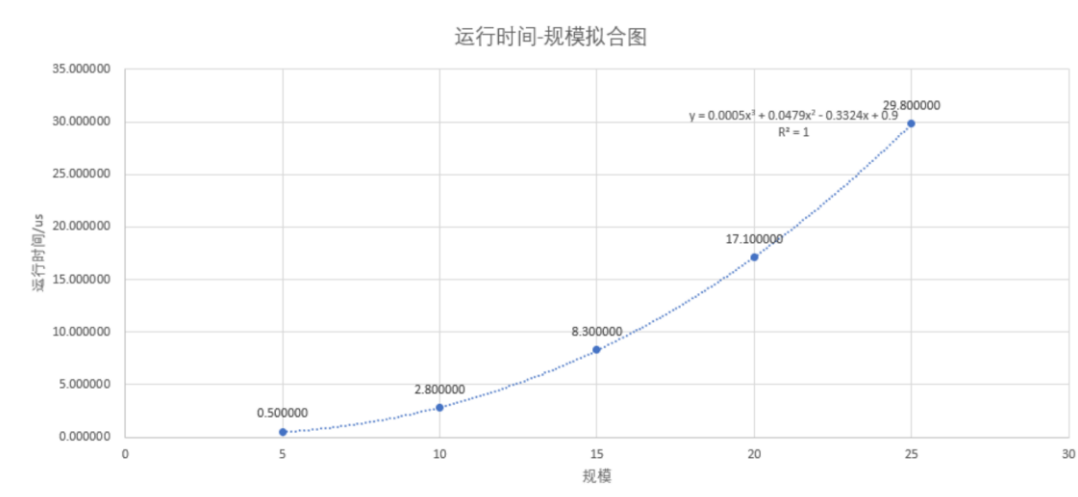
result.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
min multipul 154865959097238
my plan (A1(((A2A3)A4)A5))
min multipul 42524697503391
my plan ((A1A2)((((((A3A4)A5)A6)A7)A8)A9)A10))
min multipul 5400945319618
my plan (((((((((((((A1A2)A3)A4)A5)A6)A7)A8)A9)A10)A11)A12)A13)A14)A15)
min multipul 319329979644400
my plan ((A1(A2(A3(A4(A5(A6(A7(A8(A9(A10(A11(A12(A13(A14A15))))))))))))))(((A16A17)A18
min multipul 574911761218280
my plan ((A1(A2(A3(A4(A5(A6(A7(A8(A9(A10A11))))))))))((((((((((((((A12A13)A14)A15)A16)A17),

```

运行时间

规模n	运行时间/us
5	0.500000
10	2.800000
15	8.300000
20	17.100000
25	29.800000

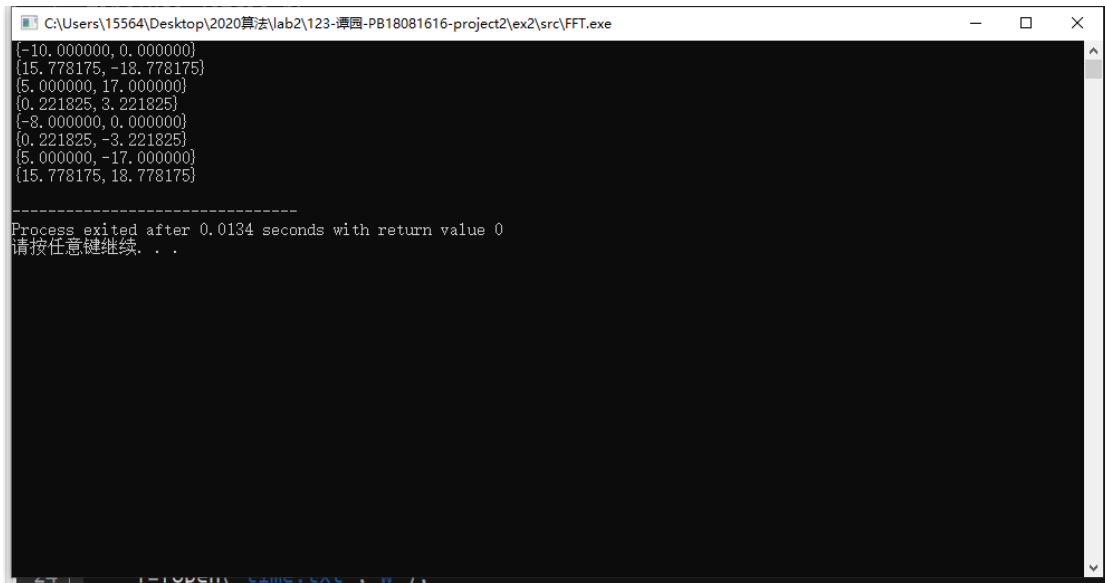
拟合图像



与三次函数拟合很好，可以说是完美满足 $O(n^3)$ 的复杂度，

FTT 实验

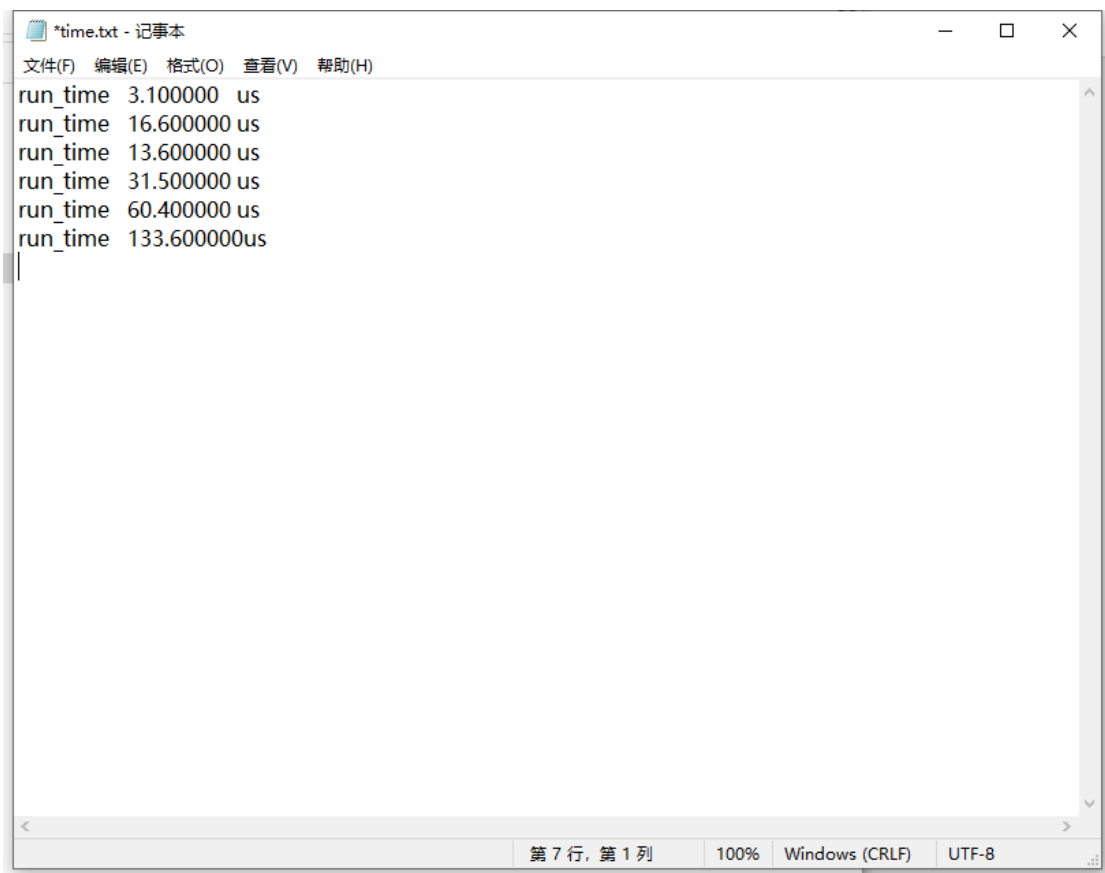
N=8 时结果截图



```
C:\Users\15564\Desktop\2020算法\lab2\123-谭园-PB18081616-project2\ex2\src\FFT.exe
{-10.000000, 0.000000}
{15.778175, -18.778175}
{5.000000, 17.000000}
{0.221825, 3.221825}
{-8.000000, 0.000000}
{0.221825, -3.221825}
{5.000000, -17.000000}
{15.778175, 18.778175}

-----
Process exited after 0.0134 seconds with return value 0
请按任意键继续. . .
```

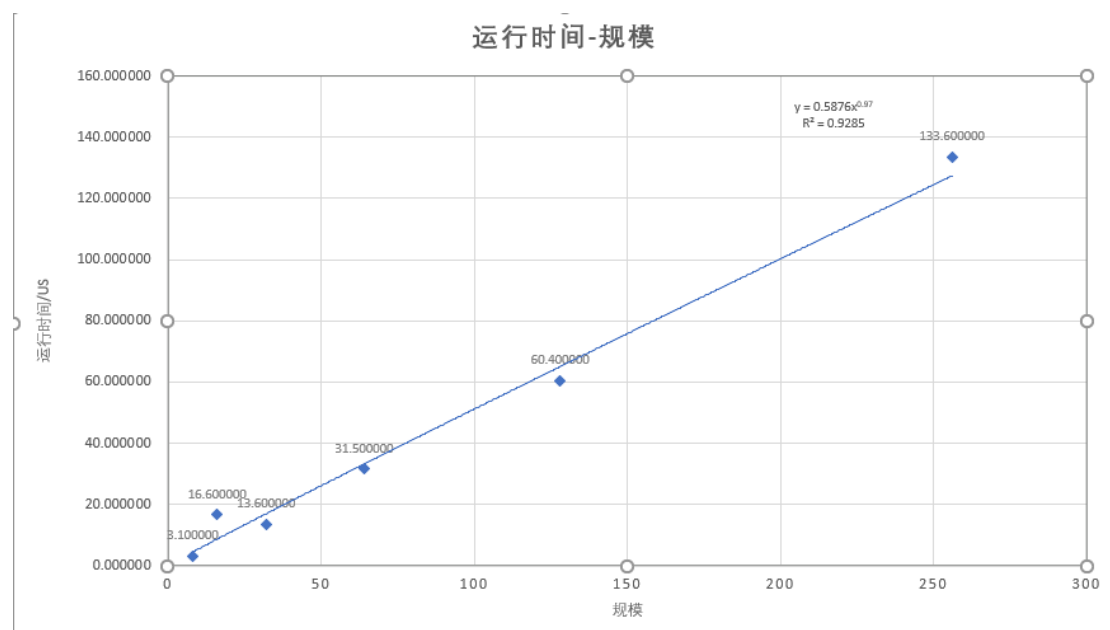
时间截图



```
*time.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
run_time 3.100000 us
run_time 16.600000 us
run_time 13.600000 us
run_time 31.500000 us
run_time 60.400000 us
run_time 133.600000us
```

result.txt - 记事本							
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)							
8							
-10.000000	15.778175	5.000000	0.221825	-8.000000	0.221825	5.000000	15.778175
16							
25.000000	7.757767	-15.121320	-18.175971	-30.000000	18.276476	-10.878	
32							
-11.000000	-7.722408	-3.295015	27.116538	-51.597980	30.126081	17.726594	-0.917
64							
7.000000	-0.483583	73.383261	-42.883963	-13.650759	-72.232194	23.283	
128							
-149.000000	-18.228541	-8.858639	-24.912755	-8.268675	62.857835	-24.68	
.117624	-76.454236	29.344085	-113.971584	-79.035989	-0.089988	20.029	
256							
-103.000000	-48.730429	82.814575	20.016996	-124.433610	69.157324	-16.60	
41.100505	83.481499	-106.466301	11.012878	-60.775968	-81.085913	159.62	
000000	-60.293669	-37.535106	-18.677348	30.117313	-9.265083	38.644	

拟合图像



而拟合的曲线方程基本可以满足 $O(n \lg n)$ 的时间复杂度，部分不太符合可能是由于数据规模太小的缘故。