

# 算法基础project1实验报告

PB19071535徐昊天

## 一.实验内容

1. 矩阵链乘最优方案
2. 所有最长公共子序列

## 二.实验设备和环境

- windows操作系统
- vscode、dev-c++
- Excel

## 三.实验方法和步骤

### 1.矩阵链乘最优方案

- 处理文件

本次实验须通过相对路径读取 `input` 文件夹内的文件数据，经过算法处理后得到输出数据并通过相对路径写入 `output` 文件夹。涉及文件处理的代码如下：

```
1  FILE *fp1,*fp2,*fp3;
2  // 打开输入输出文件
3  if((fp1 = fopen("../input/1_1_input.txt","r"))==NULL)
4  {
5      printf("cannot open input file\n");
6      exit(0);
7  }
8  if((fp2 = fopen("../output/result.txt","w"))==NULL)
9  {
10     printf("cannot open result.txt\n");
11     exit(0);
12 }
13 if((fp3 = fopen("../output/time.txt","w"))==NULL)
14 {
15     printf("cannot open time.txt\n");
16     exit(0);
17 }
```

```

18 .....
19 .....
20 .....
21 fscanf(fp1, "%d", &n); //读取矩阵个数
22 for (i = 0; i <= n; i++)
23 {
24     fscanf(fp1, "%lld", &p[i]); //读取矩阵大小向量
25 }
26 .....
27 .....
28 .....
29 fprintf(fp2, "%lld\n", m[0][n-1]); // 写入最少乘法运算次数
30 fprintf(fp2, "\n");
31 fprintf(fp3, "%dns\n", end - begin); // 写入运行时间
32 .....
33 .....
34 .....
35 //关闭输入输出文件
36 fclose(fp1);
37 fclose(fp2);
38 fclose(fp3);

```

- 计算最优代价

基于以下公式，利用递归算法计算最优代价：

$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} * p_k * p_j)$$

过程用一个辅助表m保存代价，另一个辅助表s保存最优值对应的分割点，实现代码如下：

```

1     for (i = 0; i < n; i++)
2     {
3         m[i][i] = 0;
4     }
5     for (l = 2; l <= n; l++)
6     {
7         for (i = 0; i < n - l + 1; i++)
8         {
9             j = i + l - 1;
10            m[i][j] = 9223372036854775807;
11            for (k = i; k < j; k++)
12            {
13                long long q = m[i][k] + m[k + 1][j] + p[i]*p[k + 1]*p[j + 1];
14                if (q < m[i][j])
15                {
16                    m[i][j] = q;
17                    s[i][j] = k;
18                }
19            }
20        }
21    }

```

- 构造最优解

根据辅助表s及其下标，将矩阵链乘的最优括号方案写入 `result.txt` 文件，实现代码如下：

```
1 // 输出最优括号化方案
2 void print_parens(FILE *fp,int s[][n], int i, int j)
3 {
4     if(i == j)
5     {
6         fprintf(fp, "A%d", i+1);
7     }
8     else
9     {
10        fprintf(fp, "(");
11        print_parens(fp,s,i,s[i][j]);
12        print_parens(fp,s,s[i][j]+1,j);
13        fprintf(fp, ")");
14    }
15 }
```

- 记录算法运行时间

利用头文件 `#include<time.h>` 中 `clock()` 函数记录时间，由于结果以毫秒为单位，故需将**计算最优代价**部分重复运行**1000000**次，最终得到单次运行时间以纳秒为单位，实现代码如下：

```
1 #include<time.h>
2 {
3     begin = clock();
4     for(i = 0; i < 1000000; i++)
5     {
6         .....
7         .....
8         .....
9     }
10    end = clock();
11 }
```

## 2.所有最长公共子序列

- 处理文件

本次实验须通过相对路径读取 `input` 文件夹内的文件数据，经过算法处理后得到输出数据并通过相对路径写入 `output` 文件夹。涉及文件处理的代码如下：

```
1 FILE *fp1,*fp2,*fp3,*fp4,*fp5,*fp6,*fp7;
2 // 打开输入输出文件
3 if((fp1 = fopen("../input/1_2_input.txt","r"))==NULL)
4 {
5     printf("cannot open input file\n");
6     exit(0);
7 }
8 if((fp2 = fopen("../output/result_10.txt","w"))==NULL)
9 {
10    printf("cannot open result_10.txt\n");
```

```

11     exit(0);
12 }
13 if((fp3 = fopen("../output/result_15.txt","w"))==NULL)
14 {
15     printf("cannot open result_15.txt\n");
16     exit(0);
17 }
18 if((fp4 = fopen("../output/result_20.txt","w"))==NULL)
19 {
20     printf("cannot open result_20.txt\n");
21     exit(0);
22 }
23 if((fp5 = fopen("../output/result_25.txt","w"))==NULL)
24 {
25     printf("cannot open result_25.txt\n");
26     exit(0);
27 }
28 if((fp6 = fopen("../output/result_30.txt","w"))==NULL)
29 {
30     printf("cannot open result_30.txt\n");
31     exit(0);
32 }
33 if((fp7 = fopen("../output/time.txt","w"))==NULL)
34 {
35     printf("cannot open time.txt\n");
36     exit(0);
37 }
38 .....
39 .....
40 .....
41 fscanf(fp1, "%d", &n); //读取序列长度
42 fscanf(fp1, "%s", x); //读取x序列
43 fscanf(fp1, "%s", y); //读取y序列
44 .....
45 .....
46 .....
47 if (n == 10)
48 {
49     int tell = ftell(fp2);
50     fprintf(fp2, "%d\n", 0);
51     print_lcs(fp2, x, n, n, z, b, c);
52     fseek(fp2, tell, SEEK_SET);
53     fprintf(fp2, "%d\n", sum);
54     fseek(fp2, 0, SEEK_END);
55 }
56 else if (n == 15)
57 {
58     int tell = ftell(fp3);
59     fprintf(fp3, "%2d\n", 0); // 输入一个空行
60     print_lcs(fp3, x, n, n, z, b, c);
61     fseek(fp3, tell, SEEK_SET); // 回到文件头部
62     fprintf(fp3, "%d\n", sum); // 输入LCS个数
63     fseek(fp3, 0, SEEK_END); // 回到文件末尾
64 }
65 else if (n == 20)
66 {
67     int tell = ftell(fp4);
68     fprintf(fp4, "%3d\n", 0);
69     print_lcs(fp4, x, n, n, z, b, c);
70     fseek(fp4, tell, SEEK_SET);
71     fprintf(fp4, "%d\n", sum);

```

```

72     fseek(fp4, 0, SEEK_END);
73     }
74     else if (n == 25)
75     {
76         int tell = ftell(fp5);
77         fprintf(fp5, "%4d\n", 0);
78         print_lcs(fp5, x, n, n, z, b, c);
79         fseek(fp5, tell, SEEK_SET);
80         fprintf(fp5, "%d\n", sum);
81         fseek(fp5, 0, SEEK_END);
82     }
83     else if (n == 30)
84     {
85         int tell = ftell(fp6);
86         fprintf(fp6, "%3d\n", 0);
87         print_lcs(fp6, x, n, n, z, b, c);
88         fseek(fp6, tell, SEEK_SET);
89         fprintf(fp6, "%d\n", sum);
90         fseek(fp6, 0, SEEK_END);
91     }
92     //关闭输入输出文件
93     fclose(fp1);
94     fclose(fp2);
95     fclose(fp3);
96     fclose(fp4);
97     fclose(fp5);
98     fclose(fp6);
99     fclose(fp7);

```

- 计算LCS的长度

建立辅助表b、c，b[i,j]指向的表项对应计算c[i,j]时所选择的子问题最优解，实现代码如下：

```

1     for (i = 1; i <= n; i++)
2         c[i][0] = 0;
3     for (j = 0; j <= n; j++)
4         c[0][j] = 0;
5     for (i = 1; i <= n; i++)
6     {
7         for (j = 1; j <= n; j++)
8         {
9             if (x[i - 1] == y[j - 1])
10            {
11                c[i][j] = c[i - 1][j - 1] + 1;
12                b[i][j] = 1;
13            }
14            else if (c[i - 1][j] >= c[i][j - 1])
15            {
16                c[i][j] = c[i - 1][j];
17                b[i][j] = 0;
18            }
19            else
20            {
21                c[i][j] = c[i][j - 1];
22                b[i][j] = 0;
23            }
24        }
25    }

```

- 构造所有LCS

利用动态规划，根据辅助表b、c通过深度遍历找出所有LCS并写入对应文件，实现代码如下：

```
1 // 打印所有LCS
2 void print_lcs(FILE *fp, char x[n], int i, int j, char z[n], int b[n+1][n+1], int c[n+1][n+1])
3 {
4     if (c[i][j] == 0)
5     {
6         sum++;
7         fprintf(fp, "%s\n", z + 1);
8         return;
9     }
10    int count = c[i][j];
11    while (c[i][j] == count && count > 0)
12    {
13        for (int k = j; c[i][k] == count; k--)
14        {
15            if (b[i][k] == 1)
16            {
17                z[count] = x[i - 1];
18                print_lcs(fp, x, i - 1, k - 1, z, b, c);
19            }
20        }
21        i--;
22    }
23 }
```

- 记录算法运行时间

利用头文件 `#include<time.h>` 中 `clock()` 函数记录时间，由于结果以毫秒为单位，故需将**计算LCS长度**部分重复运行**1000000**次，最终得到单次运行时间以纳秒为单位，实现代码如下：

```
1 #include<time.h>
2 {
3     begin = clock();
4     for(i = 0; i < 1000000; i++)
5     {
6         .....
7         .....
8         .....
9     }
10    end = clock();
11 }
```

## 四.实验结果和分析

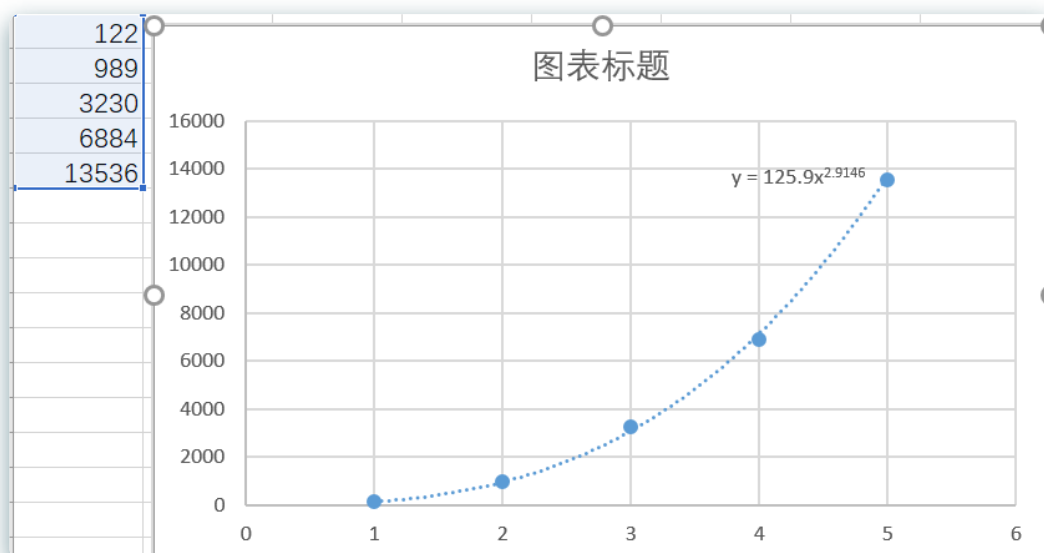
### 1.矩阵链乘最优方案

- 时间复杂度分析

程序运行后 `time.txt` 中时间如下图所示：

```
ex1 > output > ≡ time.txt
1      122ns
2      989ns
3     3230ns
4     6884ns
5    13536ns
6
```

将数据在excel中线性拟合如下图所示：



理论时间复杂度为 $O(n^3)$ ，线性拟合得实际时间复杂度为 $O(n^{2.9146})$ ，故实际时间复杂度与理论情况较为接近。

- 打印辅助表

仿照P214图15-5，打印 $n=5$ 时的结果，如下图所示：

m:	4	3	2	1	0	
0	154865959097238	128049683226820	74062781976714	15903764653528	0	
1	138766801119366	105723424955724	43981152513978	0		
2	183439291324068	119490227350806	0			
3	120958281818244	0				
4	0					
s:						
	3	2	1	0		
0	0	0	0	0		
1	3	2	1			
2	3	2				
3	3					

上图中，横坐标为  $j$  ,纵坐标为  $i$  。

- 输出结果

执行程序后， `result.txt` 中输入的最优括号化方案及最少乘法运算次数如下图所示：

```
ex1 > output > result.txt
1 154865959097238
2 (A1(((A2A3)A4)A5))
3 42524697503391
4 ((A1A2)((((A3A4)A5)A6)A7)A8)A9)A10))
5 5400945319618
6 (((((((((((((A1A2)A3)A4)A5)A6)A7)A8)A9)A10)A11)A12)A13)A14)A15)
7 319329979644400
8 ((A1(A2(A3(A4(A5(A6(A7(A8(A9(A10(A11(A12(A13(A14A15)))))))))))))(((A16A17)A18)A19)A20))
9 574911761218280
10 ((A1(A2(A3(A4(A5(A6(A7(A8(A9(A10A11))))))))))((((((((((((((A12A13)A14)A15)A16)A17)A18)A19)A20)A21)A22)A23)A24)A25))
11
```

## 2.所有最长公共子序列

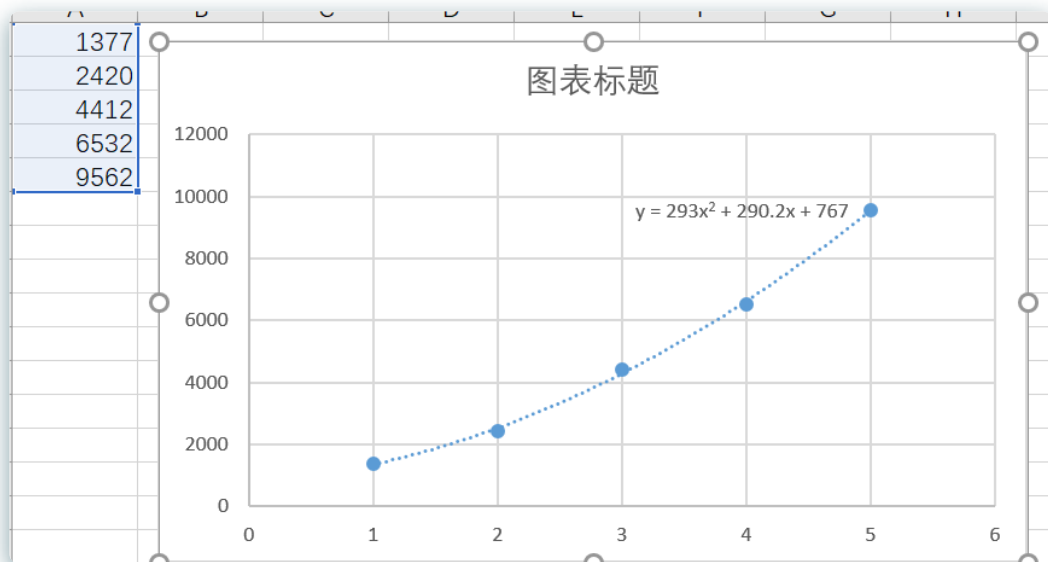
- 时间复杂度分析

程序运行后 `time.txt` 中时间如下图所示：

```
ex2 > output > time.txt
1 1377ns
2 2420ns
3 4412ns
4 6532ns
5 9562ns
6
```

将数据在excel中线性拟合如下图所示：





理论时间复杂度为 $O(n^2)$ ，线性拟合得实际时间复杂度为 $O(n^2)$ ，故实际时间复杂度与理论情况较为接近。

- 输出结果

执行程序后，`result_i.txt` 中输入的LCS个数及所有LCS如下图所示：

```
ex2 > output > ≡ result_10.txt
1      6
2      DABAB
3      CABAB
4      DABAB
5      CABAB
6      DAABA
7      CAABA
8      
```

ex2 > output > ≡ result\_15.txt

```
1 16
2 ACCBDCCC
3 ACCBDCCC
4 ACCBDCCD
5 ACCBDCCD
6 ACBCCCCD
7 CADBCCCC
8 CADBCCCC
9 BADBCCCC
10 CADBCCCC
11 CADBCCCC
12 BADBCCCC
13 ACBCCCCD
14 ACCBDCCD
15 ACCBDCCD
16 ACCBDCCD
17 ACCBDCCD
18
```

ex2 > output > ≡ result\_20.txt

```
1 221
2 CBACDADCBBDA
3 CBACDADCBBDA
4 CBACAADCBBDA
5 CBACAADCBBDA
6 BACAADCBBDA
7 CBACAADCBBDA
8 CBACAADCBBDA
9 CBACDADCBBDA
10 CBACDADCBBDA
11 CBACAADCBBDA
12 CBACAADCBBDA
13 CBACDADCBBDA
14 CBACDADCBBDA
15 CBACDADCBBAA
```

ex2 > output > ≡ result\_25.txt

```
1 5547
2 CDDBBBCCDDBADD
3 CDDBBBCCDDBADD
4 CDDBBBCCDDBADD
5 CDDBBBCCDDBADD
6 CCDDBBBCCDDBADD
7 DCDDBBBCCDDBADD
8 DCDDBBBCCDDBADD
9 CDDBBBCCDDBADD
10 CDDBBBCCDDBADD
11 CCDDBBBCCDDBADD
12 DCDDBBBCCDDBADD
13 DCDDBBBCCDDBADD
14 CCDDBBBCCDDBADD
15 DCDDBBBCCDDBADD
16 DCDDBBBCCDDBADD
```

```
ex2 > output > result_30.txt
```

```
1 288
2 ADDBBCDBBCDDDCBD
3 ADDBBCDBBCDDDCBD
4 ADDBBCDBBCDDDCBD
5 ADDBBCDBBCDDDCBD
6 ADDBBCDBBCDDDCBD
7 ADDBBCDBBCDDDCBD
8 ADDBBCDBBCDDDCBD
9 ADDBBCDBBCDDDCBD
10 ADDBBCDBBCDDDCBD
11 ADDBBCDBBCDDDCBD
12 ADDBBCDBBCDDDCBD
13 ADDBBCDBBCDDDCBD
```

## 五.实验收获与感想

1. 强化了对书本上有关矩阵链乘和LCS的动态规划算法的理解，通过对多种测试数据的执行，对算法的时间复杂度有了更深刻的理解。
2. 巩固了C语言知识，提升了实现文件输入输出功能的熟练度。
3. 首次使用计时器，作为用于记录程序运行时间的工具。