# 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：

```
FILE *fp1,*fp2,*fp3;
// 打开输入输出文件
if((fp1 = fopen("../input/1_1_input.txt","r"))==NULL)
{
    printf("cannot open input file\n");
    exit(0);
}
if((fp2 = fopen("../output/result.txt","w"))==NULL)
{
    printf("cannot open result.txt\n");
    exit(0);
}
if((fp3 = fopen("../output/time.txt","w"))==NULL)
{
    printf("cannot open time.txt\n");
    exit(0);
}
```

```
    ......
    ......
    ......
    fscanf(fp1, "%d", &n);  //读取矩阵个数
    for (i = 0; i <= n; i++)
    {
        fscanf(fp1, "%lld", &p[i]);    //读取矩阵大小向量
    }
    ......
    ......
    ......
    fprintf(fp2, "%lld\n", m[0][n-1]);  // 写入最少乘法运算次数
    fprintf(fp2, "\n");
    fprintf(fp3, "%dns\n", end - begin);  // 写入运行时间
    ......
    ......
    ......
    //关闭输入输出文件
    fclose(fp1);
    fclose(fp2);
    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：

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

  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：

```
        for ( i = 0; i < n; i++)
          {
             m[i][i] = 0;
          }
        for (l = 2; l <= n; l++)
          {
             for (i = 0; i < n - l + 1; i++)
             {
                j = i + l - 1;
                m[i][j] = 9223372036854775807;
                for (k = i ; k < j ; k++)
                {
                   long long q = m[i][k] + m[k + 1][j] + p[i]*p[k + 1]*p[j + 1];
                   if (q < m[i][j])
                   {
                      m[i][j] = q;
                      s[i][j] = k;
                   }
                }
             }
          }
```

- 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：

```
// 输出最优括号化方案
void print_parens(FILE *fp,int s[][n], int i, int j)
{
    if(i == j)
    {
        fprintf(fp, "A%d", i+1);
    }
    else
    {
        fprintf(fp, "(");
        print_parens(fp,s,i,s[i][j]);
        print_parens(fp,s,s[i][j]+1,j);
        fprintf(fp, ")");
    }
}
```

- 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：

```
#include<time.h>
{
    begin = clock();
    for(i = 0; i < 1000000; i++)
    {
        ......
        ......
        ......
    }
    end = clock();
}
```

## 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：

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

```
        exit(0);
    }
    if((fp3 = fopen("../output/result_15.txt","w"))==NULL)
    {
        printf("cannot open result_15.txt\n");
        exit(0);
    }
    if((fp4 = fopen("../output/result_20.txt","w"))==NULL)
    {
        printf("cannot open result_20.txt\n");
        exit(0);
    }
    if((fp5 = fopen("../output/result_25.txt","w"))==NULL)
    {
        printf("cannot open result_25.txt\n");
        exit(0);
    }
    if((fp6 = fopen("../output/result_30.txt","w"))==NULL)
    {
        printf("cannot open result_30.txt\n");
        exit(0);
    }
    if((fp7 = fopen("../output/time.txt","w"))==NULL)
    {
        printf("cannot open time.txt\n");
        exit(0);
    }
    ......
    ......
    ......
    fscanf(fp1, "%d", &n);  //读取序列长度
    fscanf(fp1, "%s", x);   //读取x序列
    fscanf(fp1, "%s", y);   //读取y序列
    ......
    ......
    ......
        if (n == 10)
        {
            int tell = ftell(fp2);
            fprintf(fp2, "%d\n", 0);
            print_lcs(fp2, x, n, n, z, b, c);
            fseek(fp2, tell, SEEK_SET);
            fprintf(fp2, "%d\n", sum);
            fseek(fp2, 0, SEEK_END);
        }
        else if (n == 15)
        {
            int tell = ftell(fp3);
            fprintf(fp3, "%2d\n", 0);   // 输入一个空行
            print_lcs(fp3, x, n, n, z, b, c);
            fseek(fp3, tell, SEEK_SET);     // 回到文件头部
            fprintf(fp3, "%d\n", sum);      // 输入LCS个数
            fseek(fp3, 0, SEEK_END);    // 回到文件末尾
        }
        else if (n == 20)
        {
            int tell = ftell(fp4);
            fprintf(fp4, "%3d\n", 0);
            print_lcs(fp4, x, n, n, z, b, c);
            fseek(fp4, tell, SEEK_SET);
            fprintf(fp4, "%d\n", sum);
```

```
            fseek(fp4, 0, SEEK_END);
        }
        else if (n == 25)
        {
            int tell = ftell(fp5);
            fprintf(fp5, "%4d\n", 0);
            print_lcs(fp5, x, n, n, z, b, c);
            fseek(fp5, tell, SEEK_SET);
            fprintf(fp5, "%d\n", sum);
            fseek(fp5, 0, SEEK_END);
        }
        else if (n == 30)
        {
            int tell = ftell(fp6);
            fprintf(fp6, "%3d\n", 0);
            print_lcs(fp6, x, n, n, z, b, c);
            fseek(fp6, tell, SEEK_SET);
            fprintf(fp6, "%d\n", sum);
            fseek(fp6, 0, SEEK_END);
        }
     //关闭输入输出文件
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
    fclose(fp5);
    fclose(fp6);
    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：

```
        for (i = 1; i <= n; i++)
            c[i][0] = 0;
        for (j = 0; j <= n; j++)
            c[0][j] = 0;
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= n; j++)
            {
                if(x[i - 1] == y[j - 1])
                {
                    c[i][j] = c[i - 1][j - 1] + 1;
                    b[i][j] = 1;
                }
                else if (c[i - 1][j] >= c[i][j - 1])
                {
                    c[i][j] = c[i - 1][j];
                    b[i][j] = 0;
                }
                else
                {
                    c[i][j] = c[i][j - 1];
                    b[i][j] = 0;
                }
            }
        }
```

- 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：

```
// 打印所有LCS
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])
{
    if (c[i][j] == 0)
    {
        sum++;
        fprintf(fp, "%s\n", z + 1);
        return;
    }
    int count = c[i][j];
    while (c[i][j] == count && count > 0)
    {
        for (int k = j; c[i][k] == count; k--)
        {
            if(b[i][k] == 1)
            {
                z[count] = x[i - 1];
                print_lcs(fp, x, i-1, k-1, z, b, c);
            }
        }
        i--;
    }
}
```

- 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：

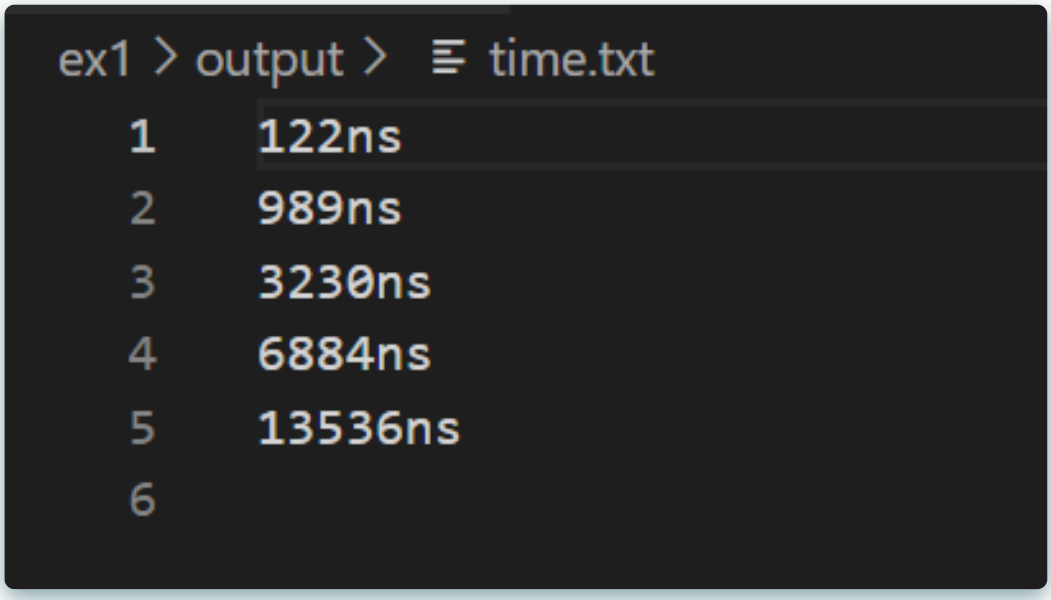
```
#include<time.h>
{
    begin = clock();
    for(i = 0; i < 1000000; i++)
    {
        ......
        ......
        ......
    }
    end = clock();
}
```

# 四.实验结果和分析

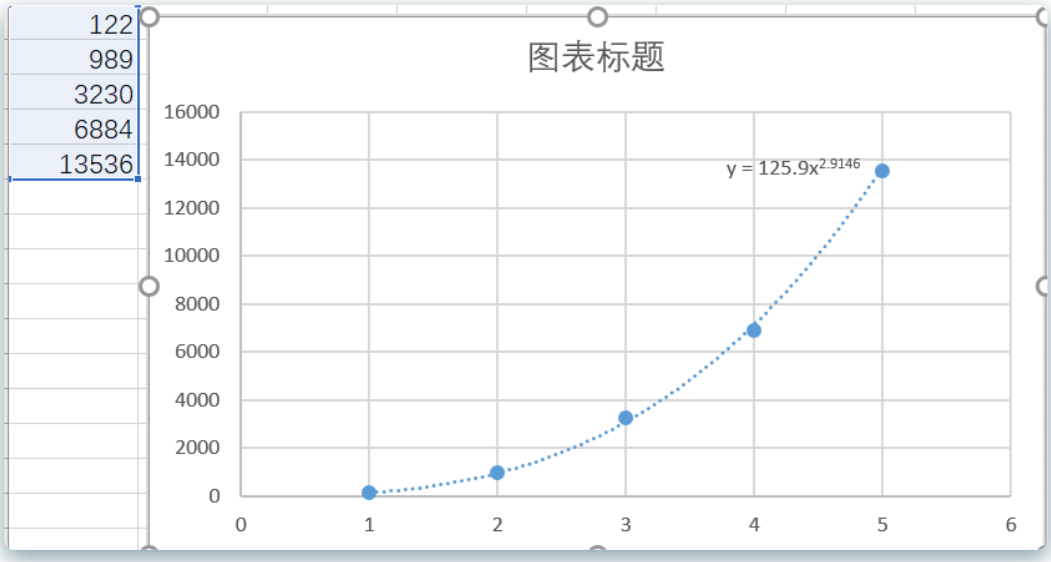
## 1.矩阵链乘最优方案

- 时间复杂度分析

  程序运行后 `time.txt` 中时间如下图所示：




  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
```

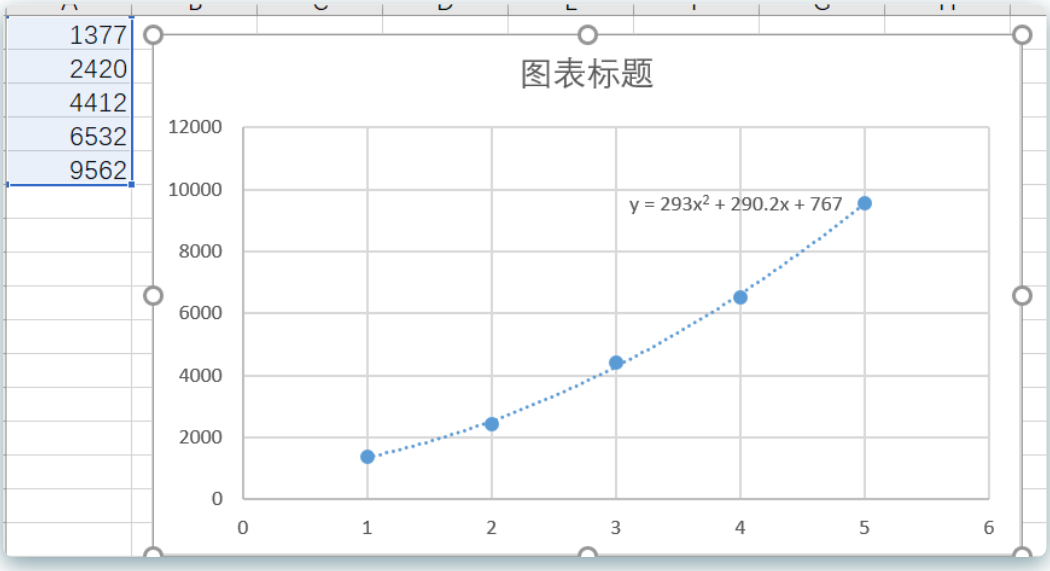
## 2.所有最长公共子序列

- 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
```

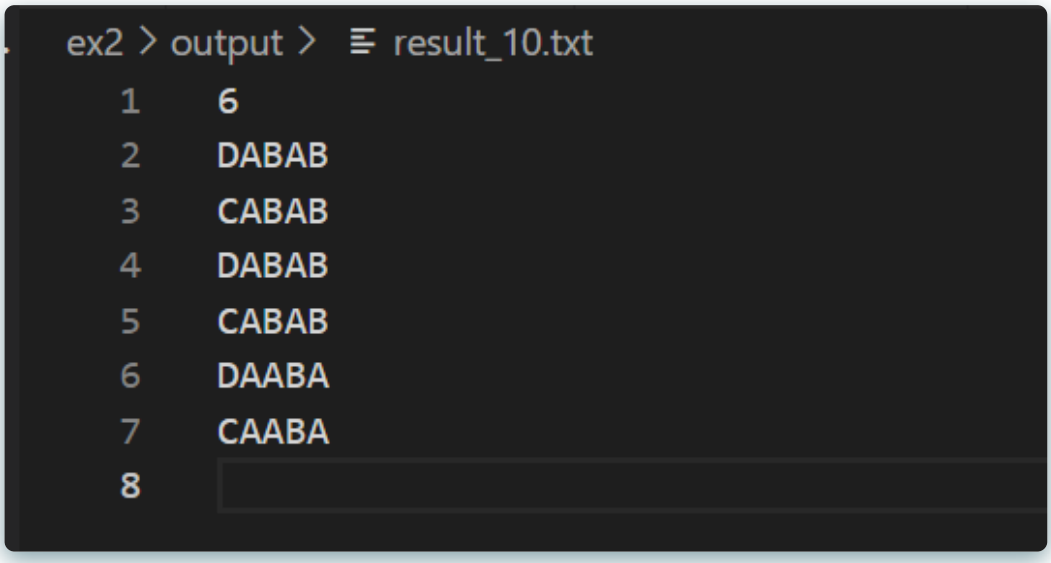
将数据在excel中线性拟合如下图所示：




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_15.txt

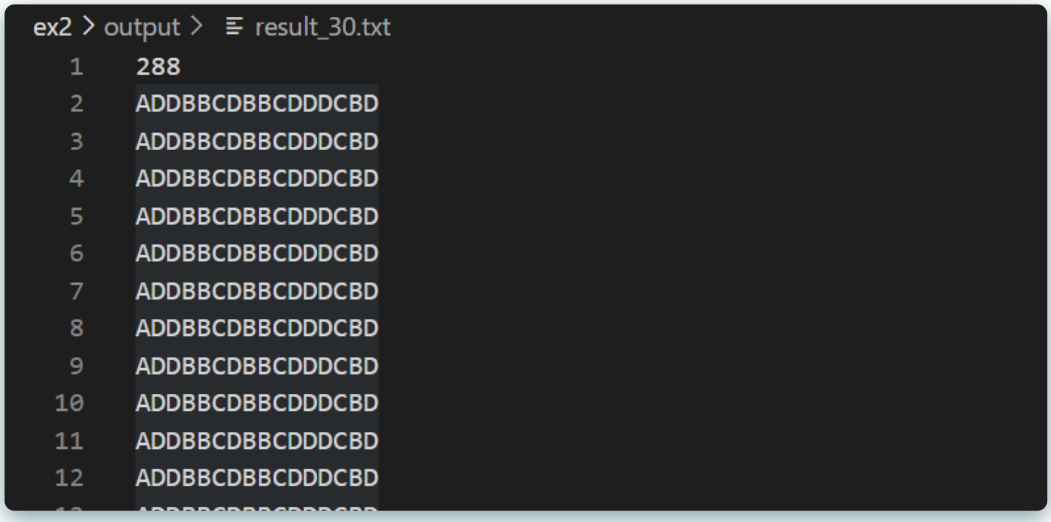
```
16
ACCBDCCC
ACCBDCCC
ACCBDCCD
ACCBDCCD
ACCBCCCD
CADBCCCD
CADBCCCD
BADBCCCD
CADBCCCD
CADBCCCD
BADBCCCD
ACCBCCCD
ACCBDCCD
ACCBDCCD
ACCBDCCD
ACCBDCCD

```

ex2 > output > result_20.txt

```
221
CBACDADCBBDA
CBACDADCBBDA
CBACAADCBBDA
CBACAADCBBDA
BACAAADCBBDA
CBACAADCBBDA
CBACAADCBBDA
CBACDADCBBDA
CBACDADCBBDA
CBACAADCBBDA
CBACAADCBBDA
CBACDADCBBDA
CBACDADCBBDA
CBACDADCBBAA
```

ex2 > output > result_25.txt

```
5547
CDDBBBCCDDBADD
CDDBBBCCDDBADD
CDDBBBCCDDBADD
CDDBBBCCDDBADD
CCDBBBCCDDBADD
DCDBBBCCDDBADD
DCDBBBCCDDBADD
CDDBBBCCDDBADD
CDDBBBCCDDBADD
CCDBBBCCDDBADD
DCDBBBCCDDBADD
DCDBBBCCDDBADD
CCDBBBCCDDBADD
DCDBBBCCDDBADD
DCDBBBCCDDBADD
```




## 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。