实验 2: MatrixChainMultiply & LCS

PB16111485 张劲暾

1. 实验要求

实验内容 1：实现矩阵链乘问题的求解算法。对 n 的取值分别为: 5、10、20、30 ,随机生成 n+1 个整数值(p_0 、 p_1 、… 、 p_n)代表矩阵的规模,其中第 i 个矩阵 ($1 \leqslant i \leqslant n$) 的规模为 $p_{i-1} \times p_i$,用动态规划法求出矩阵链乘问题的最优乘法次序,统计算法运行所需时间 ,画出时间曲线,进行性能分析。

实验内容 2：实现最长公共子序列问题的求解算法。序列 X 的长为 m ,序列 Y 的长为 n ,序列 X 和 Y 的元素从 26 个大写字母中随机生成, m 和 n 的取值:

第 1 组: (15 , 10) , (15 , 20) , (15 , 30) , (15 , 40) ,(15 , 50) , (15 , 60)

第 2 组: (15 , 25) , (30 , 25) , (45 , 25) , (60 , 25) ,(75 , 25) , (90 , 25)

给出算法运行所需的时间,画出时间曲线 ,进行性能分析。

2. 实验环境

编译环境：gcc version 7.2.0 (Ubuntu 7.2.0-8ubuntu3.2)

机器内存：8G

时钟主频：2.50GHz × 4

3. 实验过程

1. 实验 1：

(1) 编写简单的 Python 程序生成所需的随机整数序列：

```python
import numpy as np
output = open("./PB16111485-project2/ex1/input/input.txt",'w')
input_integer = np.random.randint(5,200,size=(31,1))
for integer in input_integer:
    output.write(str(integer[0]) + "\n")
output.close()
```

(2) 编写 C 语言函数实现矩阵链乘法括号化方案：

编写矩阵链乘法函数和括号化方案打印函数，详见第 4 部分。

(3) 在 main 函数中对不同长度的随机整数序列在矩阵链乘法函数上依次测试并记录相关数据:

首先将整个矩阵维度序列读入,然后依次截取要求长度的序列在矩阵链乘法函数上测试并输出括号化方案和运行时间,一个典型的测试代码块如图所示:

```
/**********************n = 10 test**************************/
gettimeofday(&start,NULL);
TempResult = MatrixChainOrder(p,10);
gettimeofday(&end,NULL);
PrintOptimalParens(TempResult->s,1,10,ResultFile);
fprintf(ResultFile,"\n");
t = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec);
fprintf(TimeFile,"n = %d\t,time cost is %ld \tus.\n",10,t);
free(TempResult->m);
free(TempResult->s);
free(TempResult);
TempResult = NULL;
```

测试位置，输入矩阵维度向量和问题规模，返回代价矩阵和划分点矩阵

将括号化方案打印到输出文件

统计用时并记录到输出文件

(4) 实验数据分析

对函数用时统计做图分析并解释数量级与理论的异同，详见第 5 部分。

2. **实验 2：**

(1) 编写简单的 Python 程序生成所需的随机大写英文字母串：

```python
import numpy.random as random
def generate_random_str(randomlength=16):
    """
    生成一个指定长度的随机字符串
    """
    random_str = ''
    base_str = 'ABCDEFGHIGKLMNOPQRSTUVWXYZ'
    length = len(base_str) - 1
    for i in range(randomlength):
        random_str += base_str[random.randint(0, length)]
    return random_str

outputA = open("./PB16111485-project2/ex2/input/inputA.txt",'w')
outputB = open("./PB16111485-project2/ex2/input/inputB.txt",'w')
outputstring = generate_random_str(15)
outputA.write(outputstring + "\n")
outputstring = generate_random_str(10)
outputA.write(outputstring + "\n")
outputstring = generate_random_str(15)
outputA.write(outputstring + "\n")
outputstring = generate_random_str(20)
```

(2) 编写 C 语言函数求解最长公共子序列：

编写最长公共子序列求解函数和最长公共子序列打印函数，详见第四部分。

(3) 在 main 函数中对不同规模的数据在最长公共子序列求解函数上依次测试并记录相关数据:

首先从输入文件读入需要处理的字符串，进行最长公共子序列求解后将结果和用时统计输出到输出文件，一个典型的测试代码块如图所示:

```
/****************************** (15,20) test ******************************/
fscanf(InputFileA,"%s\n",&X[1]);
fscanf(InputFileA,"%s\n",&Y[1]);
gettimeofday(&start,NULL);
TempResult = FindLCS(X,Y,15,20);
gettimeofday(&end,NULL);
PrintLCS(TempResult->dirc,X,15,20,ResultFile);
fprintf(ResultFile,"\n");
t = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec);
fprintf(TimeFile,"size = (15,20),time cost is %ld \tus.\n",t);
free(TempResult->cost);
free(TempResult->dirc);
free(TempResult);
TempResult = NULL;
```

从输入文件读入两个字符串

测试位置：输入两个字符串及对应的字符串长度

将最长公共子序列输出到文件

统计用时并记录到输出文件

(4) 实验数据分析：

对函数用时统计做图分析并解释数量级与理论的异同，详见第 5 部分。

4. 实验关键代码截图（结合文字说明）

1. 实验 1：

(1) 矩阵链乘法求解函数：

```
Result* MatrixChainOrder(int* p,int n){
    /**
     * function：确定矩阵相乘顺序
     * input: p:矩阵规模向量
     *        n:矩阵个数
     * output:Result->m:链乘代价矩阵
     *        Result->s:链乘划分点矩阵
     **/
    int i,j,l,k;
    Result* FinalResult = (Result*)malloc(sizeof(Result));
    FinalResult->m = (int**)malloc((n + 1) * sizeof(int*));
    FinalResult->s = (int**)malloc((n + 1) * sizeof(int*));
    for(i = 0;i <= n;i++){
        FinalResult->m[i] = (int*)malloc((n + 1) * sizeof(int));
        FinalResult->s[i] = (int*)malloc((n + 1) * sizeof(int));
    }
    for(i = 1;i <= n;i++){
        FinalResult->m[i][i] = 0;
    }
    for(l = 2;l <= n;l++){
        for(i = 1;i <= n-l+1;i++){
            j = i + l -1;
            FinalResult->m[i][j] = __INT_MAX__;
            for(k = i;k < j;k++){
                int q = FinalResult->m[i][k] + FinalResult->m[k+1][j] + p[i-1] * p[k] * p[j];
                if(q < FinalResult->m[i][j]){
                    FinalResult->m[i][j] = q;
                    FinalResult->s[i][j] = k;
                }
            }
        }
    }
    return FinalResult;
}
```

为代价矩阵和划分点矩阵分配动态内存空间

初始解：当只有一个矩阵的时候，没有链乘代价

从子问题的最优解构造原问题的最优解，确定原问题的最优划分位置

(2) 矩阵链乘法括号方案打印函数：

```
void PrintOptimalParens(int** s,int i,int j,FILE* OutputFile){
    /**
     * function:打印矩阵括号方案
     * input: s:链乘划分点矩阵
     *        i,j:链乘起止位置编号
     *        OutputFile:输出文件指针
     * output:打印输出Ai到Aj的链乘括号化方案
     **/
    if(i == j){
        fprintf(OutputFile,"A[%d]",i);
    }
    else{
        fprintf(OutputFile,"(");
        PrintOptimalParens(s,i,s[i][j],OutputFile);
        PrintOptimalParens(s,s[i][j]+1,j,OutputFile);
        fprintf(OutputFile,")");
    }
    return;
}
```

递归求解：
当只有一个矩阵时，直接打印矩阵符号，否则打印一层括号之后递归打印子矩阵串

2. 实验2：

(1) 最长公共子序列求解函数：

```
Result* FindLCS(char* X,char* Y,int XLength,int YLength){
    /**
     * function:查找出一个最长公共子序列并记录对应的子序列长度矩阵和子序列构造辅助矩阵
     * input: X，Y：求解的两个字符串
     *        XLength，YLength：字符串对应长度
     * output:FinalResult->cost：子序列长度矩阵
     *        FinalResult->dirc：子序列构造辅助矩阵，3代表对角线，2代表向上，1代表向左
     **/
    int i,j;
    Result* FinalResult = (Result*)malloc(sizeof(Result));
    FinalResult->cost = (int**)malloc((XLength + 1) * sizeof(int*));
    FinalResult->dirc = (int**)malloc((XLength + 1) * sizeof(int*));
    for(i = 0;i <= XLength;i++){
        FinalResult->cost[i] = (int*)malloc((YLength + 1) * sizeof(int));
        FinalResult->dirc[i] = (int*)malloc((YLength + 1) * sizeof(int));
    }
    for(i = 0;i <= XLength;i++){
        FinalResult->cost[i][0] = 0;
    }
    for(i = 0;i <= YLength;i++){
        FinalResult->cost[0][i] = 0;
    }
    for(i = 1;i <= XLength;i++){
        for(j = 1;j <= YLength;j++){
            if(X[i] == Y[j]){
                FinalResult->cost[i][j] = FinalResult->cost[i-1][j-1] + 1;
                FinalResult->dirc[i][j] = 3;
            }
            else{
                if(FinalResult->cost[i-1][j] >= FinalResult->cost[i][j-1]){
                    FinalResult->cost[i][j] = FinalResult->cost[i-1][j];
                    FinalResult->dirc[i][j] = 2;
                }
                else{
                    FinalResult->cost[i][j] = FinalResult->cost[i][j-1];
                    FinalResult->dirc[i][j] = 1;
                }
            }
        }
    }
    return FinalResult;
}
```

为子序列长度矩阵和构造辅助矩阵分配动态内存空间

初始解：
当有一个串为空串时，最长公共子序列长度为0

根据当前位置两个字符串的字符是否相等及子问题最优解构建原问题最优解

(2) 最长公共子序列打印函数：

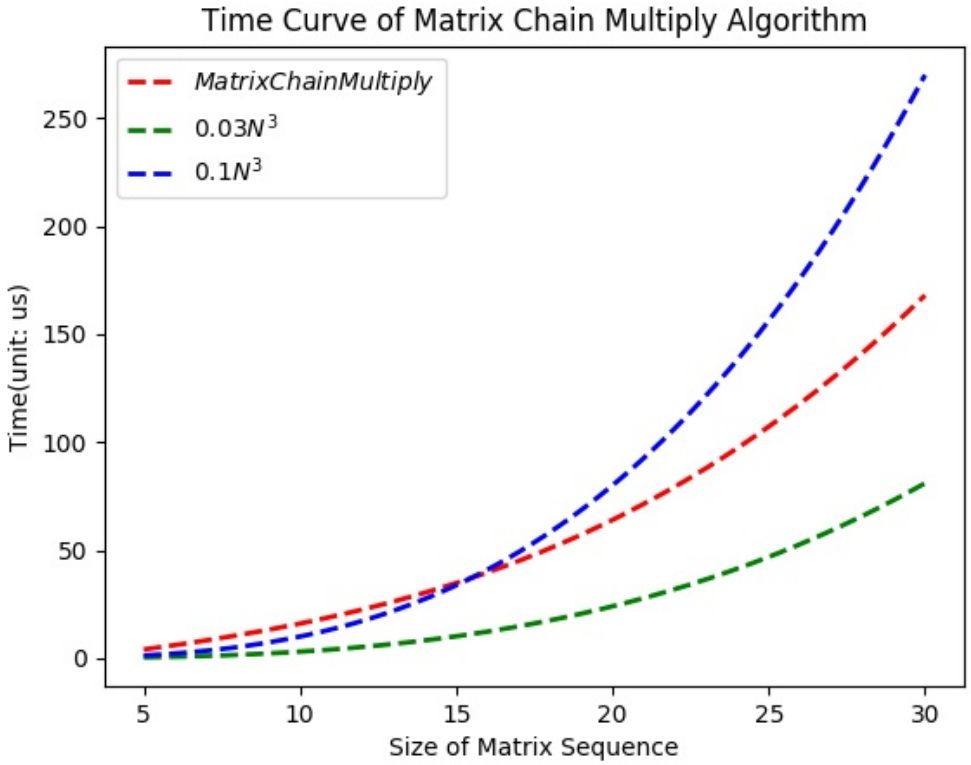
```
void PrintLCS(int** b,char* X,int i,int j,FILE* OutputFile){
    /**
     * function:打印 X[1,...,i] 与 Y[1,...,j] 的最长公共子序列
     * input:b:子序列构造辅助矩阵，3代表对角线，2代表向上，1代表向左
     *       i,j:所求公共子序列的长度
     *       OutputFile：输出文件指针
     * output:打印 X[1,...,i] 与 Y[1,...,j] 的最长公共子序列
     **/
    if(i == 0 || j == 0){
        return;
    }
    if(b[i][j] == 3){
        PrintLCS(b,X,i-1,j-1,OutputFile);
        fprintf(OutputFile,"%c",X[i]);
    }
    else{
        if(b[i][j] == 2){
            PrintLCS(b,X,i-1,j,OutputFile);
        }
        else{
            PrintLCS(b,X,i,j-1,OutputFile);
        }
    }
}
```

根据构造辅助矩阵中的辅助信息，递归地打印出最长公共子序列

5. 实验结果、分析（结合相关数据图表分析）

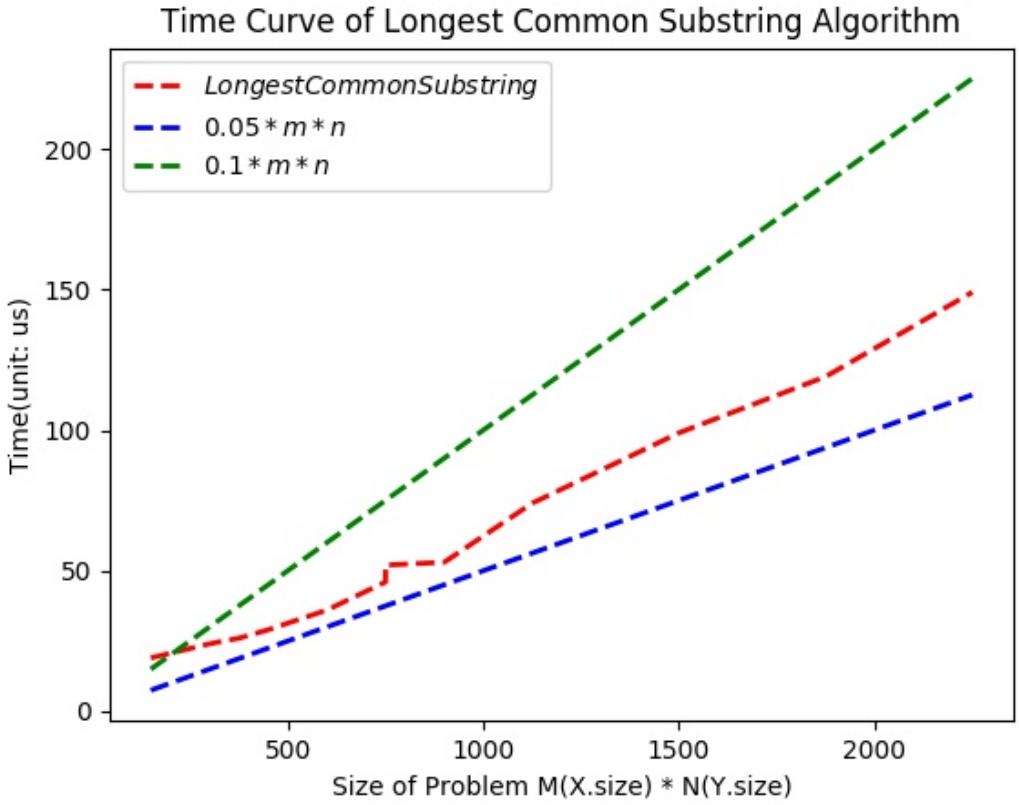
1. 实验1：结果基本正确

通过用时曲线可以看到，虽然矩阵链乘算法为 $O(N^3)$ 时间复杂度，但其实系数非常小，这主要是因为 k 和 i 的变化范围其实都小于 n。



2. **实验2：结果基本正确**

通过图线可以看到，最长公共子序列问题是O(mn)时间复杂度，但系数较小，具体曲线会有一定的抖动，这是因为在构造辅助矩阵时，产生一个“对角线”符号的代价小于“向左”或“向上”，所以根据具体的输入字符串会有一定的波动。



6. 实验心得

(1) 熟悉了动态规划算法的编程实现，提高了编程能力，加深了对于动态规划算法的理解。

(2) 通过直观的数字，体验到了动态规划算法的优越性。