运筹学课程实验实验报告

一、最小成本循环流的强多项式算法

二、动态规划算法

（一）实验要求

实现动态规划的一种快速算法。

1. 调研一种动态规划的快速算法，在报告中详细写出算法迭代过程；
2. 本作业需至少构造一个实际案例（多重背包问题/凑零钱问题/投资问题/排序问题等）；
3. 算法程序应至少在该案例上进行测试，鼓励构造更多实例，充分测试。

（二）问题描述

本实验选取矩阵链乘问题[1]作为动态规划算法的实例。

矩阵链乘问题：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。

例子：$A \in \mathcal{R}^{10\times5}, B \in \mathcal{R}^{5\times15}, C \in \mathcal{R}^{15\times10}$, 如果需要计算$ABC$，那么$((AB)C)$需要$10*5*15+10*15*10=2250$次运算，而$(A(BC))$只需要$5*15*10+10*5*10=1250$次运算。该问题即求解不同维度的矩阵相乘的最优组合方案。

（三）算法原理

1. 理论说明

首先寻找问题的状态转移方程：

1. 定义$m[i,j]$为计算$A_{i\cdots j}=A_i\cdot A_{i+1}\cdots A_j$的最优链乘方案。
2. 写出相应的递归方程
3. 于是可以根据最终的背包方案是否取第 i 个元素分为如下两种情况：

$$m[i,j]=\begin{cases}0 & i=j\\ \min_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$

第一种情况显然成立。第二种情况选取第$i\cdots k$个矩阵作为分割，分别取前后矩阵链乘的最优解和两部分矩阵相乘的和的运算次数之和作为$m[i,j]$的值，而动态规划算法选取最大的作为最优解。

上式中的p_i定义如下，$A_i \in \mathcal{R}^{p_{i-1} \times p_i}$。

之后可以根据状态转移方程编程实现。

2. 编程实现

代码的主要部分如下所示：

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

该部分采用bottum-up的方法实现矩阵链乘问题的求解。

下面对程序进行几点说明：

1. $s[i,j]$记录最佳的分割点，方便后续的打印方案。
2. 三层循环：
    1. 第一层循环根据链乘的长度循环，自底向上一步一步求出所有$m[i,j]$的值；
    2. 第二层循环遍历所有的长度为l的可能的$m[i,j]$的值；
    3. 第三层循环，遍历所有的长度为l的可能的$m[i,j]$的值，取出最大值，记录$m[i,j]$的值和$s[i,j]$的值

（四）数据集说明

1. 数据集包含在./input文件夹内。
2. 文件夹内容如下所示：

```
5
24328 18379 35569 67278 49933 36006
10
71048 47356 1383 82762 75138 55089 45196 19829 71323 56242
64531
15
86 45957 28899 70941 56147 69560 77373 58331 76247 80282 66558
66841 73727 55239 89817 81459
```

   1. 首先一行给出矩阵数量n；
   2. 下一行依次给出n个矩阵的维度，共有n+1个值。

（五）程序输入、输出

1. 程序输入

无需输入，程序会自动读取数据集内部的数据。

2. 程序输出

使用如下函数打印最佳链乘结果：

```
void print_s(vector< vector<int> > s, int i,int j,string &str)
{
    if(i==j)
    {
        stringstream ss;
        ss<<i;

        str += "A" + string(ss.str());
    }
    else
    {
        str += "(" ;
        print_s(s,i,s[i][j],str);
        print_s(s,s[i][j]+1,j,str);
        str += ")" ;
    }

}
```

程序会自动将矩阵链乘的最少链乘次数和最优方案写入./output文件夹内的result.txt，并会给出相应的时间time.txt。

（六）程序测试结果

1. 文件输出

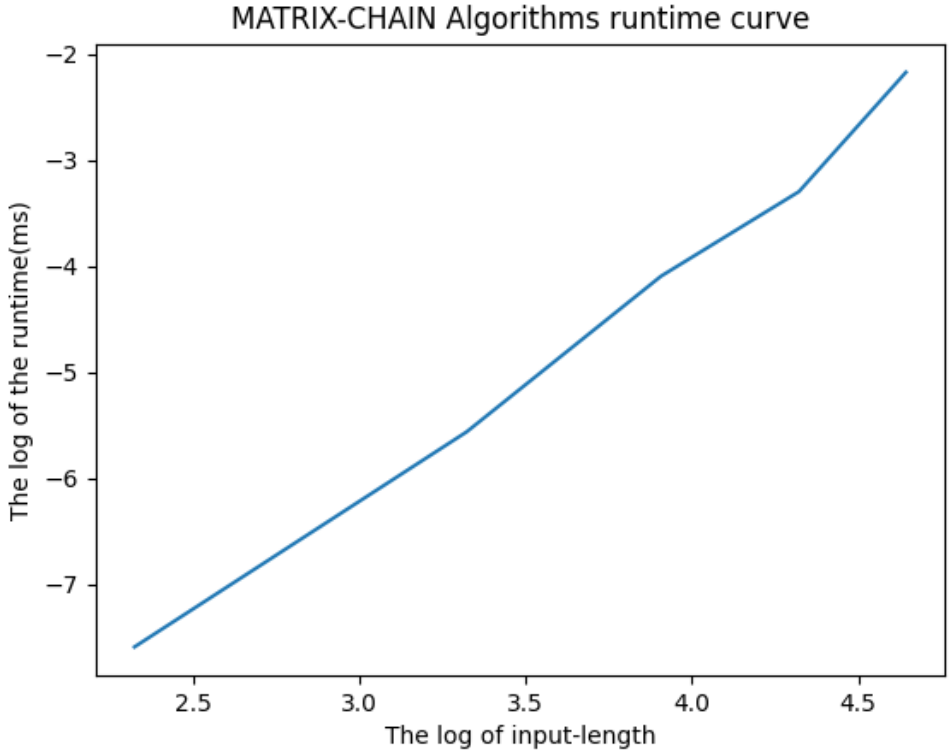
程序测试结果如下所示：

```
154865959097238
(A1(((A2A3)A4)A5))
42524697503391
((A1A2)((((((A3A4)A5)A6)A7)A8)A9)A10))
5400945319618
((((((((((((A1A2)A3)A4)A5)A6)A7)A8)A9)A10)A11)A12)A13)A14)A15)
```

依次为上述5,10,15个矩阵链乘的输出结果。

2. 可视化

- 运行时间分析
    - 下图的横坐标为输入规模n的对数，纵坐标即为运行时间（ms）的对数；
    - 从下图可以看出，取对数后，使用时间与运行规模呈正相关，表明运行时间是多项式时间复杂度；
    - curve的斜率约为2.5左右，与理论分析的时间复杂度$O(n^3)$有一些差距；
        - 可能的原因是数据量较小，三次for循环并没有完全进行，没有遇到最坏情况，导致时间复杂度有所转好。
        - 也有可能是数据量较少，不能体现完整的规律。



（七）分析总结

关于动态规划算法的矩阵链乘问题，有以下分析：

1. 该问题利用了动态规划算法的精髓，即状态转移方程来构建问题的解；
2. 该程序的算法时间复杂度为 $O(n^3)$ ，很显然由程序的三重循环可以得到；
3. 该程序输出的结果只选取了一种可能的最优解，并不能输出问题的所有最优解，通过修改输出函数，也可以输出所有的最优解。

三、非精确一维线搜索Wolfe-Powell算法

[1] 详见算法导论Chapter15.2。 ↩