数据结构与算法 II 上机实验 (10.27)

中国人民大学　信息学院　崔冠宇　2018202147

注：请使用 C++17 或以上编译器！

上机题　实现矩阵链乘问题的备忘录算法。

一、问题描述　实现矩阵链乘问题的备忘录算法

1. 输入：n 个矩阵的行列大小构成的规模序列 $< p_0, p_1, \cdots, p_n >$。
2. 输出：一种加括号方案和最小乘法次数，满足按照这种方案做矩阵乘法时，所需要的标量乘法次数最少。

二、算法基本思路

备忘录算法本质上是一种有记录的自顶向下的递归算法，很像所谓"懒惰求值"(lazy evaluation)。由于递归过程并非直接进行，而是优先查表，如果没有查到再进行递归求解，于是省去了普通递归算法中很多不必要的重复、冗余计算，大大减少了时间复杂度（本例从指数级复杂度降低到了多项式级复杂度，具体见复杂度分析）。

对于本题，设输入数组为 $p[0...n]$，创建两个矩阵 $m[n][n]$ 和 $s[n][n]$，前者用来记录最小乘法次数，后者用来记录链断在何处（为了重构解）。在计算时需要考虑两种情况：

1. 当仅有一个矩阵时，不需要做乘法；
2. 如果矩阵链上有多于一个矩阵，则考虑从某处将链断开，分别递归计算（或直接查表找到）两部分所需乘法次数并求和，再加上两个子链得到的矩阵做乘法的次数，将其最小化即可。

从而得到 m 与 s 的状态方程：

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

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

除此之外还需要注意，为了实现备忘录算法，上述两个矩阵应该初始化为一个特殊值 -1，表示值未知待求。

重构解时，使用递归算法：

1. 当仅有一个矩阵时，直接打印矩阵；
2. 否则从 $s[i][j]$ 断开矩阵链，打印左括号，递归打印左半链，递归打印右半链，最后打印右括号。

下面给出算法伪代码：

```
// 查表（备忘录）与递归求解子问题
LookupChain(p, m, s, i, j):
    if m[i][j] > 0:
        return m[i][j]
    if i == j:
        return 0;
    int minM = LookupChain(i, i) + LookupChain(i + 1, j) + p[i - 1]p[i]p[j]
    for k = 1 to n:
        int M = LookupChain(i, k) + LookupChain(k + 1, j) + p[i - 1]p[k]p[j]
        if M < minM:
            minM = M
            s[i][j] = k;
    m[i][j] = minM;
    return minM;

// 调用 LookupChain 解决最优矩阵链问题
Memorized-Matrix-Chain(p, m, s):
    int n = p.size() - 1
    for i = 1 to n:
        for j = 1 to n:
            m[i][j] = -1
    return LookupChain(p, m, s, 1, n)

// 根据求得矩阵重构解
PrintSolution(s, i, j):
    if i == j:
        print("A"i)
    else
        print("(")
        PrintSolution(s, i, s[i][j])
        PrintSolution(s, s[i][j] + 1, j)
        print(")")
```

三、算法复杂性分析

1. 我们先分析不加备忘录的纯递归算法的时间复杂度：

设算法运行时间为 $T(n)$，根据状态转移方程

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

可以写出递归式：

$$T(n) \geq \begin{cases} 1, & n = 1 \\ 1 + \sum\limits_{i=1}^{n-1}(T(i) + T(n-i) + 1), & n > 1 \end{cases}$$

迭代展开，得到 $T(n) \geq 1 + (n-1) + \sum\limits_{i=1}^{n-1} T(i) + \sum\limits_{i=1}^{n-1} T(n-i) = n + 2\sum\limits_{i=1}^{n-1} T(i)$。用数学归纳法可以证明 $T(n) \geq 2^{n-1} = \Omega(2^n)$。所以不带备忘录的递归算法由于大量的重复计算复杂度很高。

2. 下面再分析带备忘录的递归算法的时空复杂度：

   带备忘录的递归算法与动态规划的主要区别是前者是自顶向下、懒惰求值；而后者是自底向上，全部求值。

   所以在最坏情况下，带备忘录的递归算法填表次数将退化为与动态规划相同，即 $O(n^3)$，所以带备忘录的递归算法的时间复杂度为 $T(n) = O(n^3)$；

   同时，与动态规划相同，前者也需要两个矩阵作为辅助空间，所以空间复杂度为 $S(n) = \Theta(n^2)$。

四、程序源代码

矩阵链最优括号化（备忘录算法）：**Matrix-Chain.cpp**:

```
#include <iostream>
#include <vector>

// 矩阵链最优括号化
// （备忘录法）

// LookupChain - 查表以及递归求解子问题
// 输入：
//       p - 矩阵规模序列
//       m - 乘法次数矩阵
//       s - 断开点矩阵
//       i, j - 下标
// 输出：
//       m[i][j]
// T(n)=O(n^3), S(n) = O(n^2)

int LookupChain(const std::vector<int> & p,
```

```
    std::vector<std::vector<int>> & m,
    std::vector<std::vector<int>> & s,
    int i, int j)
{
    // 已经算过，直接返回
    if(m[i][j] > 0)
        return m[i][j];
    // 单矩阵
    if(i == j)
        return 0;
    int u = LookupChain(p, m, s, i, i)
        + LookupChain(p, m, s, i + 1, j)
        + p[i - 1] * p[i] * p[j];
    s[i][j] = i;
    // 枚举断开点
    for(int k = i + 1; k < j; k++)
    {
        // 分成两半求解
        int t = LookupChain(p, m, s, i, k)
            + LookupChain(p, m, s, k + 1, j)
            + p[i - 1] * p[k] * p[j];
        // 更新最小值
        if(t < u)
        {
            u = t;
            s[i][j] = k;
        }
    }
    m[i][j] = u;
    return u;
}

// MemorizedMatrixChain - 备忘录法解决矩阵链最优括号化
// 输入:
//      p - 矩阵规模序列
//      m - 乘法次数矩阵
//      s - 断开点矩阵
```

```cpp
// 输出:
//       最小乘法次数
//

int MemorizedMatrixChain(const std::vector<int> & p,
    std::vector<std::vector<int>> & m,
    std::vector<std::vector<int>> & s)
{
    // 矩阵大小
    int n = p.size() - 1;
    // 初始化为-1, 表示没算过
    // T(n)=O(n^2)
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            m[i][j] = -1;
        }
    }
    // 右上角
    return LookupChain(p, m, s, 1, n);
}

// PrintSolution - 打印括号方案
// 输入:
//       s - 断开点矩阵
//       i, j - 下标
// 输出:
//       括号方案字符串
//

void PrintSolution(const std::vector<std::vector<int>> & s,
    int i, int j)
{
    // 一个矩阵
    if(i == j)
    {
```

```
        std::cout << "A" << i;
    }
    // 递归地打印
    else
    {
        std::cout << "(";
        PrintSolution(s, i, s[i][j]);
        PrintSolution(s, s[i][j] + 1, j);
        std::cout << ")";
    }
    // T(n) <= max{T(i)+T(n-i)}+O(1)
}

int main()
{
    // 规模序列
    std::vector<int> p = {30, 35, 15, 5, 10, 20, 25};
    // 初始矩阵
    std::vector<std::vector<int>> m, s;
    std::vector<int> zeroVecN;
    for(int i = 0; i <= p.size(); i++)
    {
        zeroVecN.push_back(0);
    }
    for(int i = 0; i <= p.size(); i++)
    {
        m.push_back(zeroVecN);
    }
    s = m;
    // 输出
    std::cout << "乘法次数:" << MemorizedMatrixChain(p, m, s) << std::endl << "加括号方式:";
    PrintSolution(s, 1, 6);
    std::cout << std::endl;
    return 0;
}
```

五、运行结果截图

运行 Matrix-Chain.cpp，程序以课件例子做测试

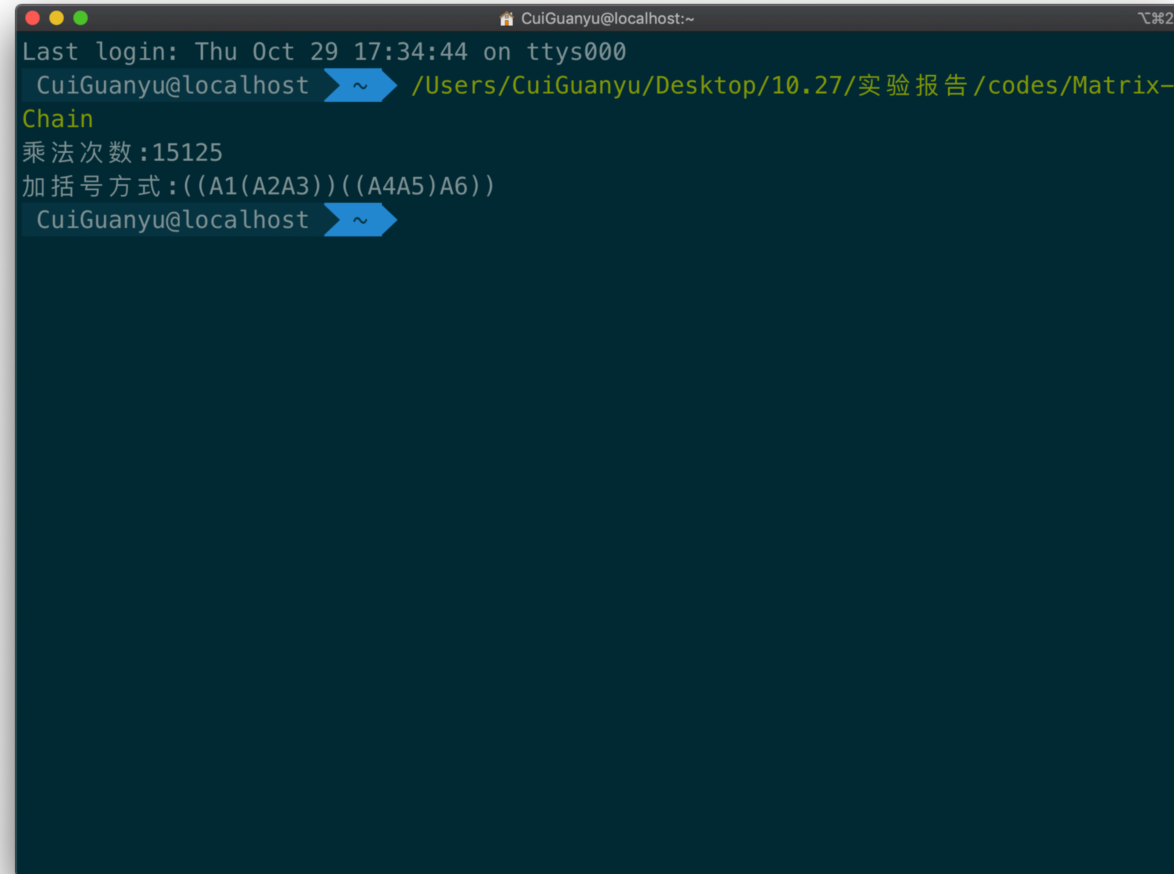

图 1: Matrix-Chain 测试

可见程序运行正确。