2019-2020 学年第一学期算法设计与分析

上机实验报告

实验名称：矩阵连乘问题

学　号	秦敏浩	姓　名	17122490	评　分	
专　业	计算机科学与技术	实验类型	综合	任课教师	岳晓冬
完成日期	2019.10.4	实验学时	2		

一、实验问题描述

1、实验内容

矩阵连乘问题：给定 n 个矩阵 A1，A2，…，An，其中，Ai 与 Aj+1 是可乘的，i=1，2，…，n-l。

你的任务是要确定矩阵连乘的运算次序，使计算这 n 个矩阵的连乘积 A1A2…An 时总的元素乘法次数达到最少。输出值和方案。

2、实验目的

- 完成矩阵连乘问题
- 理解动态规划的基本思想
- 写出此问题的动态方程
- 优化书上代码中计算最优次序的方法

3、实验环境

操作系统：Linux/Windows

编译环境：GNU G++14

二、算法设计与分析

1、算法基本思想

采用动态规划的思想求解该问题。已知所有长度为 2 的矩阵区间的计算开销为 p×q×r，且为最优。设长度为 2 到 n 的最优问题已求解完毕（n≥2），则长度为 n+1 的最优问题可以由子最优问题计算得出，从而自底向上求解直至解决整个问题。

考虑到子问题会被约 n 规模的重用，应该开辟空间存储重叠子问题的答案。

在更新当前最优问题时记录其由何处更新而来，即可在求解完成后回溯最优方案，后采用一定技巧即可输出题干要求的输出格式。

2、算法设计与求解步骤

a[x]：存储第 x 个矩阵的行和列的大小

lfa[x][y]：记录区间[x,y]的矩阵最优左乘矩阵的区间

rfa[x][y]：记录区间[x,y]的矩阵最优右乘矩阵的区间

L[x]：记录第 x 个矩阵左侧右括号个数

R[x]：记录第 x 个矩阵右侧左括号个数

dp[x][y]:记录区间[x,y]的矩阵合并最优计算量

- 初始化所有长度为 1 的区间最优计算量为 0，长度为 2 的区间最优计算量为 p×q×r
- 逐渐扩大区间长度，计算最优转移，更新时记录来源
- 输出总区间最优计算量，根据记录回溯最优方案，统计各位置括号数
- 去除冗余括号，输出方案

3、算法分析

给出一个与书上代码略有不同的转移方程，有利于简化代码实现：

$$dp[j,j+i]=\begin{cases}0 & ,\ i=0\\ a[j].x\cdot a[j].y\cdot a[j+i].y & ,\ i=1\\ \min\limits_{0\le k<i}\{dp[j,j+k]+dp[j+k+1,j+i]+a[j].x\cdot a[j+k].y\cdot a[j+i].y\} & ,\ i>1\end{cases}$$

考虑到程序计算量取决于程序中三重循环的循环次数，算法时间上界限为 $O\left(n^3\right)$，算法空间复杂度为 $O\left(n^2\right)$。

很容易注意到，在回溯过程中可能产生两种冗余的括号形式：单矩阵添加括号（即“(A1)((A2)(A3))”）和整体冗余括号（即“(A1(A2A3))”）。单矩阵括号可以通过回溯时提前退出解决，而观察到整体冗余括号一定有且仅有一对，可以在初始化时将 A1 的左括号数和 A4 的右括号数记为-1，然后正常更新即可。

考虑到每两个矩阵之间所有左括号出现在右括号前，在更新完 L，R 数组后，

输出方案已经确定。更具体的实现和更详细的代码解释可以参考附件或附录。

4、算法程序实现

篇幅原因，见附录或附件。代码附有详细注释。

三、调试与运行

输入：

3

10 100 5 50

4

50 10 40 30 5

6

10 30 20 50 8 42 70

输出：

```
3
10 100 5 50
Case 1
7500 (A1A2)A3
4
50 10 40 30 5
Case 2
10500 A1(A2(A3A4))
6
10 30 20 50 8 42 70
Case 3
44320 (A1(A2(A3A4)))(A5A6)
```

代码通过学校 oj 测试。

算法题1 矩阵连乘问题

发布时间：2017年6月19日 00:23　最后更新：2018年9月26日 19:04　时间限制：1000ms　内存限制：128M

运行结果：Accepted

时间：1ms 语言 ：C++

提交时间：2019年10月5日 21:54

四、结果分析

代码在正确时间空间复杂度下通过学校 oj 测试。

考虑现有代码可以有的一些优化：

- lfa，rfa 数组可以优化为同一个数组，因为最优左矩阵左边界一定是 x，最优右矩阵右边界限一定是 y，而最优左矩阵有边界和最优右矩阵左边界一定相邻。
- 同理，a 数组也可以优化至一半存储空间。

五、本次实验的收获、心得体会

相较于棋盘覆盖问题，矩阵连乘问题码量稍大。但在仔细理清转移方程后，就可以比较轻松的一遍写对，毕竟代码核心就一行。

在本次实验中，我尝试用不同于书上的转移方程解决问题，个人认为自己的转移方程可以更加清晰的展现问题的本质，第三重循环变量的意义发生了改变。

此外，我还进一步修改了书上回溯部分代码，使其能够适应题干要求的输出。为了解决冗余括号问题，使用到了一些技巧。

附录：代码实现（也可见附件）

```
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
#define x first
#define y second
```

```
#define INF 10000000 // big enough for this problem

pair<int,int> a[30]; // represents the size of each matrix
pair<int,int> lfa[30][30]; // {a,b}=lfa[x][y] means the optimized left parent of matrix[x to y] is matrix[a to b]
pair<int,int> rfa[30][30]; // {c,d}=rfa[x][y] means the optimized right parent of matrix[x to y] is matrix[c to d]
// apparently a==x && d==y
int L[30],R[30]; // the amount of "["s and "]"s before each matrix
int dp[30][30]; // dp[x][y] equals to the optimized cost to merge matrix[x to y]
int n;

void init(){
    for (int i=0;i<30;++i)
        for (int j=0;j<30;++j)
            dp[i][j]=INF;
    for (int i=0;i<30;++i)
        dp[i][i]=0; // the cost of a single matrix is 0
    for (int i=0;i<n-1;++i)
        dp[i][i+1]=a[i].x*a[i].y*a[i+1].y; // the cost of two connect matrix
    for (int i=0;i<30;++i)
        for (int j=0;j<30;++j)
            lfa[i][j]=rfa[i][j]=make_pair(i,j); // update where the answer is from
    memset(L,0,sizeof(L));
    memset(R,0,sizeof(R));
    --L[0];--R[n-1]; // remove the abundant outer "[]"
}

void dfs(pair<int,int> a){ // dfs to get the answer
    int l=a.x,r=a.y;
    if (l!=r){
        ++L[l];++R[r];
    }
    if (lfa[l][r]!=a) dfs(lfa[l][r]); // a is a single matrix == ( lfa[a.x][a.y]==a )
    if (rfa[l][r]!=a) dfs(rfa[l][r]);
//    printf("(%d,%d)\n",l,r);
}

int main(){
    int T,tmp,kase=0;
    while (~scanf("%d",&n)){ // scanf returns -1 if it reaches EOF, ~(-1)==0==false
        memset(a,0,sizeof(a));
        scanf("%d",&a[0].x);
        for (int i=0;i<n;++i){
            scanf("%d",&a[i].y);
            a[i+1].x=a[i].y;
        }
        a[n].x=0;
        init();
        for (int i=2;i<=n;++i){
            for (int j=0;j+i<n;++j){
                for (int k=0;k<=i-1;++k){
//                     dp[j to j+i] =====> min (dp[j to j+k] + dp[j+k+1 to j+i] + cost) for each k
                    tmp=dp[j][j+i];
                    dp[j][j+i]=min(dp[j][j+i],dp[j][j+k]+dp[j+k+1][j+i]+(a[j].x)*(a[j+k].y)*(a[j+i].y));
                    if (dp[j][j+i]!=tmp){ // update if the cost changes
                        lfa[j][j+i]=make_pair(j,j+k);
                        rfa[j][j+i]=make_pair(j+k+1,j+i);
                    }
                }
            }
        }
        dfs(make_pair(0,n-1)); // get the final answer
        printf("Case %d\n%d ",++kase,dp[0][n-1]);
        for (int i=0;i<n;++i){
            for (int j=0;j<L[i];++j) putchar('(');
            printf("A%d",i+1);
            for (int j=0;j<R[i];++j) putchar(')');
        }
        putchar('\n');
//        for (int j=0;j<n;++j){
//            for (int k=0;k<n;++k){
//                if (dp[j][k]==INF) printf("     .");
//                else printf("%6d",dp[j][k]);
//            }
//            putchar('\n');
//        }
    }
    return 0;
}
```