上海大学实验报告

专业：计算机科学
姓名：颜乐春
学号：15124542
日期：2017-10-09
地点：704

课程名称：算法设计於分析　指导老师：神人　成绩：59
实验名称：棋盘覆盖问题　实验类型：Null　同组学生姓名：None

一、 问题描述於实验目的

给定 n 个矩阵 $A_1\ A_2\ \cdots\ A_n$，其中，A_i 与 A_{j+1} 是可乘的，$i = 1\ 2\ \cdots\ n-l$。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 $A_1A_2\cdots A_n$ 时总的元素乘法次数达到最少。例如：3 个矩阵 $A_1\ A_2\ A_3$ 阶分别为 $10\times 100\ 100\times 5\ 5\times 50$，计算连乘积 $A_1A_2A_3$ 时按 ($A_1A_2A_3$ 所需的元素乘法次数达到最少，为 7500 次。

1. 输入

测试数据有若干组，每组测试数据有 2 行。

每组测试数据的第 1 行是一个整数 n，$(0 < n < 20)$，第 2 行是 $n+1$ 个正整数 $p_0\ p_1\ p_2\ \cdots\ p_n$，这些整数不超过 100，相邻两个整数之间空一格，他们表示 n 个矩阵 $A_1\ A_2\ \cdots\ A_n$，的阶 $p_{i-1}, p_i\ i = 1\ 2\ \cdots\ n$.

输入直到文件结束。

2. 输出

对输入中的每组测试数据，输出 2 行。先在一行上输出 “Case Num”，其中 “Num” 是测试数据的组号（从 1 开始），再在第 2 行上输出计算这 n 个矩阵的连乘积 $A_1A_2\cdots A_n$ 时最少的总的元素乘法次数，再空一格，接着在同一行上输出矩阵连乘的添括号形式。注意：最外层括号应去掉。

3. 输入样例

```
3
10 100 5 50
4
50 10 40 30 5
```

4. 输出样列

```
Case 1
7500 (A1A2)A3
Case 2
```

```
10500 A1(A2(A3A4))
```

二、 实验环境

Ubuntu 17.04 + gcc 6.3

三、 实验内容和步骤

1. 设计思路

计算 A[i:j] 的最优次序所包含的计算矩阵子链 A[i:k] 和 A[k+1:j] 的次序也是最优的，也即是说这个问题具有最优子结构性质，可以用动态规划解决。下面是状态转移方程:

$$m[i,j] = \min_{ik<j} m[i,k] + m[k+1,j] + p_{i-1}p_i \tag{1}$$

2. 算法描述

四、 实现程序

```
#include <iostream>
using namespace std;

const int BOARD_SZ = 8;
static int tile = 1;
static int board[BOARD_SZ][BOARD_SZ] = {0};

void chess_board(int tr, int tc, int dr, int dc, int size)
{
    if(size == 1)
        return;

    int t = tile++; //tile means 瓦片，基石，覆盖的步骤
    int sz = size / 2; //每次进行划分

    //cover top left corner
    if(dr < tr+sz && dc < tc+sz) //notice < < //注意一共四种情况，<>=这几个符号要控制好边界
        chess_board(tr, tc, dr, dc, sz);
    else{
        board[tr+sz-1][tc+sz-1] = t;
        chess_board(tr, tc, tr+sz-1, tc+sz-1, sz);
    }

    //cover top right corner
    if(dr < tr+sz && dc >= tc+sz) //notice < >=
        chess_board(tr, tc+sz, dr, dc, sz);
    else{
```

```
        board[tr+sz-1][tc+sz] = t;
        chess_board(tr, tc+sz, tr+sz-1, tc+sz, sz);
    }

    //cover lower left corner
    if(dr >= tr+sz && dc < tc+sz) //notice >= <
        chess_board(tr+sz, tc, dr, dc, sz);
    else{
        board[tr+sz][tc+sz-1] = t;
        chess_board(tr+sz, tc, tr+sz, tc+sz-1, sz);
    }

    //cover lower right corner
    if(dr >= tr+sz && dc >= tc+sz) //notice >= >=
        chess_board(tr+sz, tc+sz, dr, dc, sz);
    else{
        board[tr+sz][tc+sz] = t;                //标记一个假设的特殊点
        chess_board(tr+sz, tc+sz, tr+sz, tc+sz, sz); //递归该部分
    }
}

void print_chess_board()
{
    cout.setf(ios::left); //左对齐
    for(int i=0; i<BOARD_SZ; ++i){
        for(int j=0; j<BOARD_SZ; ++j){
            cout.width(3); //打印宽度为3
            cout<<board[i][j];
        }
        cout<<endl;
    }
}

int main()
{
    chess_board(0, 0, 3, 4, BOARD_SZ);
    print_chess_board();
    return 0;
}
```