上海大学

Shanghai University

算法设计与分析

实验 1-3

姓名：孙义严

学号：19122487

专业：计算机科学与技术

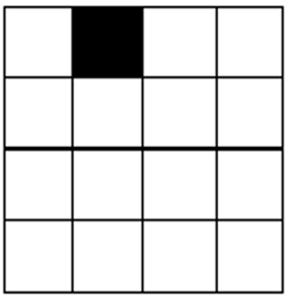
2021 年 10 月 21 日

目录

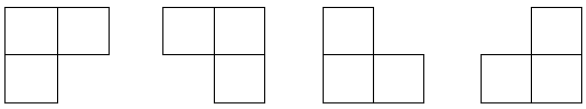
1 棋盘覆盖问题

问题描述

设 $n = 2^k\,(k \geqslant 0)$。在一个 $n \times n$ 个方格组成的棋盘中，恰有 1 个方格与其他方格不同，称该方格为特殊方格。显然，特殊方格在棋盘中可能出现的位置有 4^k 种，因而有 n^2 种不同的棋盘，下图所示是 $k = 2, n = 4$ 时 16 种棋盘中的一个。棋盘覆盖问题要求用下图所示的 4 种不同形状的 L 型骨牌覆盖给定棋盘上除特



殊方格以外的所有方格，且任何 2 个 L 型骨牌不得重叠覆盖。



易知，在任何一个 $n \times n$ 的棋盘中，用到的 L 型骨牌个数恰为 $(n^2 - 1)/3$。若给定棋盘大小 $k > 1, n = 2^k$，请设计一个算法实现棋盘的一种覆盖。

问题分析

采用分治的思想，将棋盘分割成 4 个 $2^{k-1} \times 2^{k-1}$ 的子棋盘，对无特殊方格的 3 个子棋盘，在靠近中心的位置上各添一个特殊方格，使其转化为特殊棋盘。再对 4 个含特殊方格的子棋盘进行同样处理。

算法的时间复杂度递推式为：

$$T(n) = \begin{cases} 4T(n/2) + O(1) & (n > 1) \\ O(1) & (n = 1) \end{cases}$$

实验程序代码

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
int tile;
int board[100][100];

void chessBoard(int tr,int tc,int dr,int dc,int size){
    if(size==1)
        return;
    int t=tile++;
    int s=size/2;
    if(dr<tr+s&&dc<tc+s)
        chessBoard(tr,tc,dr,dc,s);
    else
    {
        board[tr+s-1][tc+s-1]=t;
        chessBoard(tr,tc,tr+s-1,tc+s-1,s);
    }
    if(dr<tr+s&&dc>=tc+s)
        chessBoard(tr,tc+s,dr,dc,s);
    else
    {
        board[tr+s-1][tc+s]=t;
        chessBoard(tr,tc+s,tr+s-1,tc+s,s);
    }
    if(dr>=tr+s&&dc<tc+s)
        chessBoard(tr+s,tc,dr,dc,s);
    else
    {
        board[tr+s][tc+s-1]=t;
        chessBoard(tr+s,tc,tr+s,tc+s-1,s);
    }
    if(dr>=tr+s&&dc>=tc+s)
        chessBoard(tr+s,tc+s,dr,dc,s);
    else
    {
        board[tr+s][tc+s]=t;
        chessBoard(tr+s,tc+s,tr+s,tc+s,s);
    }
}

int main()
{
    int k,n=0;
    int index_x,index_y;
    while(cin>>k>>index_x>>index_y)
    {
        n++;
        tile=1;
        int size=int(pow(2,k));
        chessBoard(0,0,index_x-1,index_y-1,size);
        cout<<"Case "<<n<<": \n"<<"n="<<size<<endl;

        for(int i=0;i<size;i++)
        {
            for(int j=0;j<size;j++){
                k=board[i][j];
                if (k==0){
                    cout<<setw(4)<<"#";
                }else{
                    cout<<setw(4)<<board[i][j];
                }
            }
            cout<<endl;
        }
    }

    return 0;
}
```

运行结果

```
(base) sunyiyan@SUNYIYANs-MacBook-Pro exp_1 % make run
g++ -std=c++17 -Wall -Wextra -g -Iinclude -o output/main src/main.o  -Llib
Executing all complete!
./output/main
1 2 1
Case 1:
n=2
   1   1
   #   1
2 2 3
Case 2:
n=4
   2   2   3   3
   2   1   #   3
   4   1   1   5
   4   4   5   5
```

数字形式输出棋盘覆盖

实验中遇见的问题

本次实验参考了教科书上有例程，太久不写 C++，一些语言细节已经记不清，导致在设置二维数组的时候，忘记不能设置固定的空间，而要动态地根据数据规模来申请空间。同时过程中发现使用 C++ 来绘制多色棋盘界面非常困难，由于时间关系并没有继续做下去。

2 矩阵连乘问题

问题描述

给定 n 个矩阵 $A_1, A_2, ..., A_n$ ，其中，A_i 与 A_{j+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 $A_1A_2...A_n$ 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 10×100100×55×50 ，计算连乘积 $A_1 * A_2 * A_3$ 时按 $(A_1 * A_2) * A_3$ 所需的元素乘法次数达到最少，为 7500 次。

问题分析

由题意知可将矩阵 A_1 表示为 $p * q$ ，矩阵 A_2 表示为 $q * t$ ，则矩阵 $A_1 * A_2$ 所需乘法次数为 $p * q * t$ 。设 $f[L, R]$ 代表所有将 $[L, R]$ 进行完全加括号得到最终结果的方式的集合，所以 $f[L, R]$ 的值同时也等于该集合的乘法数量最少次数将 $f[L, R]$ 划分子集，其子集有

1. $f[L, L]$ 的集合, 最少次数为 $f[L + 1, R] + p[L - 1] * p[L] * p[R]$
2. $f[L, L+1]$ 的集合，最少次数为 $f[L, L+1]+p[L-1]*p[L+1]*p[R]+f[L+2, R]$
3. $cf[L, L+2]$ 的集合，最少次数为 $f[L, L+2]+p[L-1]*p[L+2]*p[R]+f[L+3, R]$
4. ...
5. $f[L, k]$ 的集合，最少次数为 $f[L][k] + p[L - 1] * p[k] * p[R] + f[k + 1][R]$，其中 k 属于 $[L + 1, R]$

所以动态方程为

$$f[i][j] = \begin{cases} 0 & (i = j) \\ \min\limits_{i \leqslant k < j} \{f[i][k] + f[k + 1][j] + p_{i-1}p_kp_j\} & (i \neq j) \end{cases}$$

实验程序代码

```
#include <iostream>
#include <fstream>
#include <cassert>
#include <string>
#include <sstream>
using namespace std;
const int N = 100;
int m[N][N], s[N][N];
void PrintAnswer(int i, int j)
{
    if (i == j)
        cout << "A" << i;
    else
    {
        cout << "(";
        PrintAnswer(i, s[i][j]);
        PrintAnswer(s[i][j] + 1, j);
        cout << ")";
    }
}
void Matrix_Multi(int *p, int length)

{
    int n = length - 1;
    int l, i, j, k, q = 0;
    //m[i][i]只有一个矩阵，相乘次数为零，所以m[i][i]=0
    for (i = 1; i < length; i++)
    {
        m[i][j] = 0;
    }
    for (l = 2; l <= n; l++) //宽度从2到n
    {
        for (i = 1; i <= n - l + 1; i++)
        {
            j = i + l - 1; //以i为起始位，j为末位，长度为l
            m[i][j] = 0x7fffffff;
            for (k = i; k <= j - 1; k++)
            {
                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;
                }
            }
        }
    }
}

void inPut()
{
    int t;
    int i = 1;
    while (cin>>t)
    {
        int x[t+1];
        for(int k=0;k<=t;k++)
        {
            cin>>x[k];
        }
        Matrix_Multi(x, t + 1);
        cout << "Case " << i << endl;
        i++;
        cout << m[1][t] << " ";
        PrintAnswer(1, t);
        cout << endl;
    }
}
int main()
{
    inPut();
    return 0;
}
```

运行结果

```
(base) sunyiyan@SUNYIYANs-MacBook-Pro exp_2 % make run
g++ -std=c++17 -Wall -Wextra -g -Iinclude -o output/main src/main.o  -Llib
Executing all complete!
./output/main
3
10 100 5 50
Case 1
7500 ((A1A2)A3)
4
50 10 40 30 5
Case 2
10500 (A1(A2(A3A4)))
```

实验中遇见的问题

矩阵连乘积计算次序问题的最优解包含着其子问题的最优解，这也是动态规划题的特征性质，遇见这一类问题解决的关键问题在于根据所给条件找出状态转移方程。

3 最长公共子序列问题

问题描述

序列 $Z =< B, C, D, B >$ 是序列 $X =< A, B, C, B, D, A, B >$ 的子序列，相应的递增下标序列为 $< 2, 3, 5, 7 >$。

一般地，给定一个序列 $X =< x_1, x_2, \ldots, x_m >$，则另一个序列 $Z =< z_1, z_2, \ldots, z_k >$ 是 X 的子序列，是指存在一个严格递增的下标序列 $< i_1, i_2, \ldots, i_k >$ 使得对于所有 $j = 1, 2, \ldots, k$ 使 Z 中第 j 个元素 z_j 与 X 中第 i_j 个元素相同。

给定 2 个序列 X 和 Y，当另一序列 Z 既是 X 的子序列又是 Y 的子序列时，称 Z 是序列 X 和 Y 的公共子序列。请给定 2 个序列 X、Y，并求 X 和 Y 的最长公共子序列 Z。

问题分析

要到 X 和 Y 中最长的公共子序列，首先需要考虑 X 的最后一个元素和 Y 的最后一个元素。

1. 如果 $x_n = y_m$ ，即 X 的最后一个元素与 Y 的最后一个元素相同，这说明该元素位于公共子序列中, 因此 $LCS(X_{n-1}, Y_{m-1})$ 是原问题的一个子问题。

2. 如果 $x_n! = y_m$，产生了两个子问题：$LCS(X_{n-1}, Y_m)$ 和 $LCS(X_n, Y_{m-1})$。因为序列 X 和序列 Y 的最后一个元素不相等，那说明最后一个元素不是最长公共子序列中的元素。所以 $LCS(X, Y) = \max(LCS(X_{n-1}, Y_m), LCS(X_n, Y_{m-1}))$

动态方程为：

$$c[i][j] = \begin{cases} 0 & (i = 0 \text{ 或} j = 0) \\ c[i-1][j-1] + 1 & (i, j > 0 \text{ 且} x_i = y_j) \\ \max\{c[i][j-1], c[i-1][j]\} & (i, j > 0 \text{ 且} x_i \neq y_j) \end{cases}$$

实验程序代码

```
#include <iostream>
#include <fstream>
#include <cassert>
#include <string>
#include <sstream>
#include <set>
#include <iomanip>
using namespace std;

int c[100][100];
int d[100][100];
set<string> setOfLCS;

string Reverse(string str)
{
    int low = 0;
    int high = str.length() - 1;
    while (low < high)
    {
        char temp = str[low];
        str[low] = str[high];
        str[high] = temp;
        ++low;
        --high;
    }
    return str;
}

void LCSLength(int m, int n, char *x, char *y)
{
    int i, j;
    for (i = 1; i <= m; i++)
        c[i][0] = 0;
    for (i = 1; i <= n; i++)
        c[0][i] = 0;
    for (i = 1; i <= m; i++)
    {
        for (j = 1; j <= n; j++)
        {
            if (x[i] == y[j])
            {
                c[i][j] = c[i - 1][j - 1] + 1;
                d[i][j] = 1;
            }
            else if (c[i - 1][j] > c[i][j - 1])
            {
                c[i][j] = c[i - 1][j];
                d[i][j] = 2;
            }
            else
            {
                c[i][j] = c[i][j - 1];
                d[i][j] = 3;
            }
        }
    }
}
void traceBack(int i, int j, char *x, char *y, string lcs_str)
{
    while (i > 0 && j > 0)
    {
        if (x[i] == y[j])
        {
            lcs_str.push_back(x[i]);
            --i;
            --j;
        }
        else
        {
            if (c[i - 1][j] > c[i][j - 1])
                --i;
            else if (c[i - 1][j] < c[i][j - 1])
                --j;
            else
```

```
            {
                traceBack(i - 1, j, x, y, lcs_str);
                traceBack(i, j - 1, x, y, lcs_str);
                return;
            }
        }
    }
    setOfLCS.insert(Reverse(lcs_str));
}

void LCS(int i, int j, char *x)
{
    if (i == 0 || j == 0)
        return;
    if (d[i][j] == 1)
    {
        LCS(i - 1, j - 1, x);
        cout << x[i];
    }
    else if (d[i][j] == 2)
        LCS(i - 1, j, x);
    else
        LCS(i, j - 1, x);
}

int main()
{
    int T, m, n, times = 1;
    char s1[100], s2[100];
    cin >> T;

    cout << endl;
    while (T--)
    {
        cout << "case:" << times << endl;
        cin >> m >> n;
        for (int i = 1; i <= m; i++)
            cin >> s1[i];
        for (int i = 1; i <= n; i++)
            cin >> s2[i];
        cout << "Case " << times++ << endl;
        LCSLength(m, n, s1, s2);
        cout << c[m][n] << " ";
        cout << "LCS(X,Y):";
        cout << endl;
        string str;
        setOfLCS.clear();
        traceBack(m, n, s1, s2, str);
        set<string>::iterator beg = setOfLCS.begin();
        for (; beg != setOfLCS.end(); ++beg)
        {
            for (int i = 0; i < (*beg).length(); i++)
            {
                cout << setw(2) << (*beg)[i];
            }
            cout << endl;
        }
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= m; j++)
            {
                cout << c[j][i] << " ";
            }
            cout << endl;
        }
    }
    return 0;
}
```

```
(base) sunyiyan@SUNYIYANs-MacBook-Pro exp_2 % make run
g++ -std=c++17 -Wall -Wextra -g -Iinclude -o output/main src/main.o  -Llib
Executing all complete!
./output/main
2

case:1
7 6
A B C B D A B
B D C A B A
Case 1
4 LCS(X,Y):
 B C A B
 B C B A
 B D A B
0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1
0 0 1 1 1 2 2 2
0 0 1 2 2 2 2 2
0 1 1 2 2 2 3 3
0 1 2 2 3 3 3 4
0 1 2 2 3 3 4 4
case:2
8 9
b a a b a b a b
a b a b b a b b a
Case 2
6 LCS(X,Y):
 a a b a b a
 a a b a b b
 a a b b a b
 a b a b a b
 b a a b b a
 b a b a b a
 b a b a b b
 b a b b a b
0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1
0 1 1 1 2 2 2 2 2
0 1 2 2 2 3 3 3 3
0 1 2 2 3 3 4 4 4
0 1 2 2 3 3 4 4 5
0 1 2 3 3 4 4 5 5
0 1 2 3 4 4 5 5 6
0 1 2 3 4 4 5 5 6
0 1 2 3 4 5 5 6 6
```

运行结果

实验中遇见的问题

在输出最长公共子序列的长度的基础上，输出一个最长公共子序列并不难，但是输出全部的最长公共子序列变得困难起来，涉及一个对动态规划过程的回溯，需要自定义状态值线索表来保存状态才能实现目标。