**实验五**

**实验目的与要求**：掌握动态规划方法的基本思想与设计策略。

**1．最长递增子序列问题**

**【问题描述】**

**给定一个整数数组，设计一个动态规划算法求出该数组中的最长递增子序列。**

首先找到他的子问题：对于长度为的数组，假设我们想求以结尾的最大递增子序列长度，设为，那么,.这样，想求结尾的最大递增子序列的长度，我们就需要遍历之前的所有位置(0到），找出，计算这些中，能产生最大的i，之后就可以求出。之后我对每一个中的元素都计算以他们各自结尾的最大递增子序列的长度，这些长度的最大值，就是我们要求的问题——数组A的最大递增子序列。

时间复杂度：由于每一次都要与之前的所有做比较，这样时间复杂度为。

int LIS\_DP**(**int **\***array**,** int n**)**

**{**

int LIS**[**n**];**

**for(**int i **=** 0**;** i **<** n**;** i**++)**

LIS**[**i**]** **=** 1**;**

**for(**int i **=** 1**;** i **<** n**;** i**++)**

**{**

int maxLen **=** 0**;**

**for(**int j **=** 0**;** j **<** i**;** j**++)**

**{**

**if(**array**[**i**]** **>** array**[**j**])**

**{**

**if(**maxLen **<** LIS**[**j**])**

maxLen **=** LIS**[**j**];**

**}**

**}**

LIS**[**i**]** **=** maxLen **+** 1**;**

**}**

int maxLIS **=** 0**;**

**for(**int i **=** 0**;** i **<** n**;** i**++)**

**{**

**if(**maxLIS **<** LIS**[**i**])**

maxLIS **=** LIS**[**i**];**

**}**

**return** maxLIS**;**

**}**

**2．矩阵连乘问题**

**【问题描述】**

**给定个矩阵,其中是可乘的，,考察这n个矩阵的连乘积，设计一个动态规划算法，求出这个矩阵连乘积问题的最优计算顺序。**

**实现要求：随机生成n个合法的可连乘的矩阵，以完全加括号的方式输出其最优计算顺序。**

完全加括号的矩阵连乘积可递归地定义为：

（1）单个矩阵是完全加括号的；

（2）矩阵连乘积A是完全加括号的，则可表示为2个完全加括号的矩阵连乘积和的乘积并加括号，即。

**递推关系：**

设计算，所需要的最少数乘次数，则原问题的最优值为。当时，，因此，；当时，若的最优次序在和之间断开，,则：。由于在计算是并不知道断开点的位置，所以还未定。不过的位置只有个可能。因此，是这个位置使计算量达到最小的那个位置。

综上，有递推关系如下：



**构造最优解：**

若将对应的断开位置记为，在计算出最优值后，可递归地由构造出相应的最优解。中的数表明，计算矩阵链的最佳方式应在矩阵和之间断开，即最优的加括号方式应为。因此，从记录的信息可知计算的最优加括号方式为，进一步递推，的最优加括号方式为。同理可以确定的最优加括号方式在处断开……照此递推下去，最终可以确定的最优完全加括号方式，及构造出问题的一个最优解。

int MemoizedMatrixChain(int n, int \*\*m, int \*\*s, int \*p)

{

for(int i = 1; i <= n; i++)

for(int j = 1; j <= n; j++)

m[i][j] = 0;

return LookupChain(1, n, m, s, p);

}

int LookupChain(int i, int j, int \*\*m, int \*\*s, int \*p)

{

if(m[i][j] > 0)

return m[i][j];

if(i == j)

return 0;

int u = LookupChain(i, i, m, s, p) + LookupChain(i+1, j, m, s, p) + p[i-1]\*p[i]\*p[j];

s[i][j]=i;

for(int k = i+1; k < j; k++)

{

int t = LookupChain(i, k, m, s, p) + LookupChain(k+1, j, m, s, p) + p[i-1]\*p[k]\*p[j];

if(t < u)

{

u = t;

s[i][j] = k;

}

}

m[i][j] = u;

return u;

}

void Traceback(int i, int j, int \*\*s)

{

if(i == j)

return;

Traceback(i, s[i][j], s);

Traceback(s[i][j]+1, j, s);

cout << "Multiply A" << i << "," << s[i][j];

cout << " and A" << (s[i][j]+1) << "," << j << endl;

}