Homework 7   
对于矩阵链乘积(Matrix Chain-Products)问题，请问是否存在其它的子问题计算次序，使得可以保证在计算到每一个子问题时，其所需要的子问题的解已经存在。如果有，请给出这样的计算次序。

解答：

由于矩阵乘法满足结合律，故计算矩阵的连乘积可以有许多不同的计算次序。这种计算次序可以用加括号的方式来确定。若一个矩阵连乘积的计算次序完全确定，也就是说该连乘积已完全加括号，则可以依此次序反复调用2个矩阵相乘的标准算法计算出矩阵连乘积。我们可以使用 动态规划迭代方式解决此问题，可依据其递归式自底向上的方式进行计算。在计算过程中，保存已解决的子问题的答案。每个子问题只计算一次，而在后面需要时只需简单检查一下，从而避免了大量的重复计算，最终得到多项式时间的算法。

#include "stdafx.h"

#include <iostream>

**using namespace** std;

**cons int** L = 7;

**int** MatrixChain(**int** n,**int** \*\*m,**int** \*\*s,**int** \*p);

**void** Traceback(**int** i,**int** j,**int** \*\*s);//构造最优解

**int** main()

{

**int** p[L]={30,35,15,5,10,20,25};

**int** \*\*s = **new** **int** \*[L];

**int** \*\*m = **new int** \*[L];

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

{

s[i] = **new** **int**[L];

m[i] = **new** **int**[L];

}

cout<<"矩阵的最少计算次数为："<<MatrixChain(6,m,s,p)<<endl;

cout<<"矩阵最优计算次序为："<<endl;

Traceback(1,6,s);

**return** 0;

}

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

{

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

{

m[i][i] = 0;

}

**for**(**int** r=2; r<=n; r++)//r为当前计算的链长（子问题规模）

{

**for**(**int** i=1; i<=n-r+1; i++)//n-r+1为最后一个r链的前边界

{

**int** j=i+r-1;//计算前边界为r，链长为r的链的后边界

m[i][j] =m[i+1][j]+p[i-1]\*p[i]\*p[j];//将链ij划分为A(i)\*(A[i+1:j])

s[i][j] = i;

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

{

//将链ij划分为(A[i:k])\*(A[k+1:j])

**int** t = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j];

**if**(t<m[i][j])

{

m[i][j]=t；

s[i][j]=k;

}

}

}

}

**return** m[1][L-1];

}

**void** Traceback(**int** i,**int** j,**in** \*\*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;

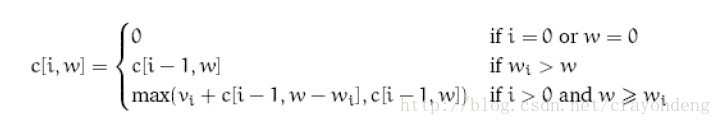
}

Homework 8   
用动态规划方法求解背包问题。

解答：

动态规划解决0-1背包问题步骤如下：

0-1背包问题子结构：选择一个给定物品i，则需要 **比较  选择 i 的形成的子问题的最优解**与 **不选择 i 的子问题的最优解**。分成两个子问题，进行选择比较，选择最优的。0-1背包问题递归过程：设有n个物品，背包的重量为w，C[i][w]为最优解。即：



#include <iostream>

using namespace std;

//物品数据结构

typedef struct commodity

{

int value; //价值

int weight; //重量

}commodity;

const int N = 3; //物品个数

const int W = 50; //背包的容量

//初始物品信息

commodity goods[N+1]={{0,0},{60,10},{100,20},{120,30}};

int select[N+1][W+1];

int max\_value();

int main()

{

int maxvalue = max\_value();

cout<<"The max value is: ";

cout<<maxvalue<<endl;

int remainspace = W;

//输出所选择的物品列表：

for(int i=N; i>=1; i--)

{

if (remainspace >= goods[i].weight)

{

if ((select[i][remainspace]-select[i-1][remainspace-goods[i].weight]==goods[i].value))

{

cout << "item " << i << " is selected!" << endl;

remainspace = remainspace - goods[i].weight;//如果第i个物品被选择，那么背包剩余容量将减去第i个物品的重量 ;

}

}

}

return 0;

}

int max\_value()

{

//初始没有物品时候，背包的价值为0

for(int w=1;w<=W;++w)

select[0][w] = 0;

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

{

select[i][0] = 0; //背包容量为0时，最大价值为0

for(int w=1;w<=W;++w)

{

if(goods[i].weight <= w) //当前物品i的重量小于等于w，进行选择

{

if( (goods[i].value + select[i-1][w-goods[i].weight]) > select[i-1][w])

select[i][w] = goods[i].value + select[i-1][w-goods[i].weight];

else

select[i][w] = select[i-1][w];

}

else //当前物品i的重量大于w，不选择

select[i][w] = select[i-1][w];

}

}

return select[N][W]; //最终求得最大值

}

再来看看其中最优值和最优解的求解过程。

最优值：求解整个背包最后的总价值达到最优。求解背包问题的最优值，关键是要弄清楚上面的递推式子。

最优解：当背包价值达到最大时，列出所选取的物品都是那些。

代码的实现过程是这样的：

int remainspace = W;

//输出所选择的物品列表：

for(int i=N; i>=1; i--)

{

if (remainspace >= goods[i].weight)

{

if ((select[i][remainspace]-select[i-1][remainspace-goods[i].weight]==goods[i].value))

{

cout << "item " << i << " is selected!" << endl;

remainspace = remainspace - goods[i].weight;//如果第i个物品被选择，那么背包剩余容量将减去第i个物品的重量 ;

}

}

}