实验六

实验要求：背包问题的贪心算法。

注意：这是**背包问题**，而不是**0-1背包问题**，背包问题可以用贪心算法进行求解，但0-1无法用贪心算法求解，需要用动态规划算法求解；

首先对贪心算法做一下总结，以及它与动态规划算法的区别：

贪心算法两个最重要的性质：

（1）贪心选择性质；

（2）最优子结构性质；

其中，贪心选择性质：**自顶向下**进行决策，每次做出的决策都是局部最优解，且每次做出决策后问题规模都变小了；最优子结构性质：即问题的最优解结构中包含子问题的最优解；

动态规划算法的两个最重要的性质：

（1）重叠子问题性质；

（2）最优子结构性质；

其中最优解子结构性质和贪心算法相似，唯一不同的是重叠子问题性质，因为动态规划算法是**自底向上**的算法，它需要首先将原始问题分解为若干个相互有联系的子问题，在计算的时候有的子问题可能会被计算很多次，所以动态规划算法会将这些子问题的解存在一个表格中，使得最终对于这些子问题只需要求解一次（可以使原来需要再指数时间内解决的问题可以在多项式问题中得到解决）

背包问题求解代码如下：

#include <iostream>

using namespace std;

const int N = 3;

void Knapsack(int n,float M,float v[],float w[],float x[]);

int main()

{

float M = 50;//背包所能容纳的重量

//这里给定的物品按单位价值减序排序

float w[] = {0,10,20,30};//下标从1开始

float v[] = {0,60,100,120};

float x[N+1];

cout<<"背包所能容纳的重量为："<<M<<endl;

cout<<"待装物品的重量和价值分别为："<<endl;

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

{

cout<<"["<<i<<"]:("<<w[i]<<","<<v[i]<<")"<<endl;

}

Knapsack(N,M,v,w,x);

cout<<"选择装下的物品比例如下："<<endl;

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

{

cout<<"["<<i<<"]:"<<x[i]<<endl;

}

return 0;

}

void Knapsack(int n,float M,float v[],float w[],float x[])

{

//Sort(n,v,w);//这里假定w[],v[]已按要求排好序

int i;

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

{

x[i]=0;//初始化数组x[]

}

float c=M;

for (i=1;i<=n;i++)//物品整件被装下,x[i]=1

{

if (w[i]>c)

{

break;

}

x[i]=1;

c-=w[i];

}

//物品i只有部分被装下

if (i<=n)

{

x[i]=c/w[i];

}

}

