背包问题1: 01背包

问题描述：共有N个物品，每个物品数量为1，每个物品的大小为C[i]，每个物品的价值为W[i]，背包的容量为V，背包所能装下的物品总价值最大为多少？

定义动态规划状态：f[i][v]表示前i个物品恰放入一个容量为v的背包可以获得的最大价值

动态规划状态转移方程：f[i][v]=max{f[i-1][v],f[i-1][v-C[i]]+w[i]}

解释：前i个物品装入容量为v的背包所能装入物品的最大总价值为两种情况下的最大值：第一种情况：第i个物品不放入背包中，此时最大价值是前i-1个物品装入容量为v的背包所能装入物品的最大总价值，即f[i][v]=f[i-1][v];第二种情况：第i个物品放入背包中，此时的最大价值是前i-1个物品装入容量为v-c[i]的背包所能装入物品的最大总价值，即f[i][v]=f[i-1][v-C[i]];

结果：f[N-1][ 0 ~ V-1 ]中的最大值，如果想让f[N-1][V-1]为最终的结果，可以在状态转换方程中加入f[i][v-1],即状态转换方程变为：

f[i][v]=max{f[i-1][v],f[i-1][v-C[i]]+w[i],f[i][v-1]};

实现：根据状态方程可知，需要一个二维数组，用来记录f[i][v]的数据，行数和列数分别为：N+1和V+1，其中f[0][0~V]和f[0~N][0]都初始化为0，从f[1][1]开始遍历。

优化空间复杂度：

针对状态转换方程为：f[i][v]=max{f[i-1][v],f[i-1][v-C[i]]+w[i]}，可以只使用一个一维数组f[v]，长度为V+1，遍历从前1个物品到前N个物品，其中求每个物品时遍历从容量V到容量1。因为求f[i][v]需要f[i-1][v-C[i]]的数据，所以容量必须从V到1遍历。



针对状态转换方程为：f[i][v]=max{f[i-1][v],f[i-1][v-C[i]]+w[i],f[i][v-1]};，需要一个2\*(V+1)的二维数组，具体同上，但是容量需要从1到V遍历，因为f[i][v]需要f[i-1][v-C[i]]和f[i][v-1]的数据。



背包问题2： 完全背包问题

问题描述：共有N个物品，每个物品数量不限，每个物品的大小为C[i]，每个物品的价值为W[i]，背包的容量为V，背包所能装下的物品总价值最大为多少？

定义动态规划状态：f[i][v]表示前i个物品恰放入一个容量为v的背包可以获得的最大价值，同背包问题1

动态规划状态转移方程：f[i][v]=max{f[i-1][v-k\*C[i]]+k\*W[i] | 0<=k\*C[i]<=V}，其中的k表示装入背包的第i间物品的数量。实际上背包问题1就是完全背包问题的一种特例，即k=0,1的情况。

方案： 1.直接按照转换方程求，时间复杂度O(N\*∑(V/C[i]))

2.将问题转换为01问题，即每个物品不论有多少个都当做一个单独的物品来处理，时间复杂度一样，只是更好理解

3.将每个物品分解为若干个物品，比如物品i分解为C[i]、2\*C[i]、4\*C[i]、8\*C[i]...2^k\*C[i],其中2^k\*C[i]<V,这样不论背包里有几个i物品，都可以分解为这几种组合的叠加，时间复杂度为O(N\*∑(log(V/C[i])))

4.时间复杂度为O(N\*V)的方案如下

实现：与问题1类似，只是每次求max要求比较更多的内容，所以时间复杂度增大了

优化：实际上求每个f[i][v]时都进行了多次重复的计算，可以省略。如：

求f[i][v-C[i]]时需要：



求f[i][v]时需要：



所以在求f[i][v]时可以由f[i-1][v]和f[i][v-C[i]]得到



状态转换方程变为：f[i][v]=max{f[i-1][v],f[i][v-C[i]]}

从而可以用长度为V+1的一维数组f[v]来记录数据，从前1个物品到前N个物品遍历，求每个物品时又从容量为1到V遍历，最终得到f[N][V]就是最终的结果。

背包问题3：多重背包问题

问题描述：共有N个物品，每个物品数量为n[i]，每个物品的大小为C[i]，每个物品的价值为W[i]，背包的容量为V，背包所能装下的物品总价值最大为多少？

状态转换方程：f[i][v]=max{f[i-1][v-k\*C[i]] | 0<=k\*C[i]<=V && k<=n[i]}

实现方案： 1.直接根据状态转换方程求解，时间复杂度为O(N\*∑(n[i]))

2.转换为01背包问题，时间复杂度为O(N\*∑(n[i]))

3.将第i个物品分解为若干个物品：C[i]、2\*C[i]、4\*C[i]、。。。2^k\*C[i]。。。(n[i]-2^k)\*C[i],时间复杂度O(N\*∑(log(n[i])))

4.时间复杂度O(NV)的算法，基于基本算法的状态转移方程，但需要应用单调队列使每个状态的值可以以均摊O(1)的时间求解。单调队列优化的DP已经超出了NOIP的范围，出处：楼天成的“男人八题”幻灯片上

其中问题3采用的方案3实现

前三个问题的C++代码：

#include <iostream>

#include <vector>

#include <string>

#include <map>

#include <list>

#include <deque>

#include <algorithm>

#include <cmath>

#include <iomanip>

using namespace *std*;

int zeroOnePack(*vector*<int>&C,*vector*<int>&W,int V){

int N=C.*size*();

*vector*<int> f(V+1,0);

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

for(int j=V;j>0;j--){

f[j]=max(f[j-1],j-C[i-1]>0?f[j-C[i-1]]+W[i-1]:0);

}

}

int max=0;

for(int i=0;i<=V;i++)

if(f[i]>max)

max=f[i];

*cout*<<max<<*endl*;

return max;

}

int completePack(*vector*<int>&C,*vector*<int>&W,int V){

int N=C.*size*();

*vector*<int> f(V+1,0);

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

for(int j=1;j<=V;j++){

f[j]=max(f[j],j-C[i-1]>0?f[j-C[i-1]]+W[i-1]:0);

}

}

int max=0;

for(int i=0;i<=V;i++)

if(f[i]>max)

max=f[i];

*cout*<<max<<*endl*;

return 0;

}

int multiplePack(*vector*<int>& C,*vector*<int>& W,*vector*<int>& Nums,int V){

int len=C.*size*();

for(int i=0;i<len;i++){

int tmp=2;

while(tmp<=Nums[i]){

C.*push\_back*(tmp\*C[i]);

W.*push\_back*(tmp\*W[i]);

tmp<<=1;

}

tmp>>=1;

tmp=Nums[i]-tmp;

if(tmp&(tmp-1)){

C.*push\_back*(tmp\*C[i]);

W.*push\_back*(tmp\*W[i]);

}

}

int N=C.*size*();

*vector*<int> f(V+1,0);

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

for(int j=V;j>=1;j--){

f[j]=max(f[j],j-C[i-1]>0?f[j-C[i-1]]+W[i-1]:0);

}

}

int max=0;

for(int i=0;i<=V;i++)

if(f[i]>max)

max=f[i];

*cout*<<max<<*endl*;

return 0;

}

int *main*(){

/\*int data1[]={77,22,29,50,99};

int data2[]={92,22,87,46,90};

int N=5,V=100;\*/

int data1[]={79,58,86,11,28,62,15,68};

int data2[]={83,14,54,79,72,52,48,62};

int data3[]={2,3,1,4,3,2,5,4};

int N=8,V=200;

*vector*<int> C(data1,data1+N);

*vector*<int> W(data2,data2+N);

*vector*<int> News(data3,data3+N);

//zeroOnePack(C,W,V);

//completePack(C,W,V);

multiplePack(C,W,News,V);

return 0;

}

背包问题4 混合三种背包问题

将之前的三种背包问题混合到一起，解决方法：

针对01背包和完全背包问题的混合：根据物品的类别选择顺序或逆序的循环

针对多重背包，可以直接转换为01背包问题就可以

背包问题5 二维费用的背包问题

问题：共有N个物品，每个物品数量为1，每个物品的费用1为C1[i]费用2为C2[i]，每个物品的价值为W[i]，背包的容纳费用1的容量为V，容纳费用2的容量为U，背包所能装下的物品总价值最大为多少？(背包中如果装了物品i，那么对于的C1[i]和C2[i]都要考虑)

动态规划状态定义：f[i][v][u]表示前i个物品装入一个费用1的容量为v且费用2的容量为u的背包中，所有装法中总价值的最大值

动态规划转移方程：f[i][v][u]=max{f[i-1][v][u],f[i-1][v-C1[i]][u-C2[i]]}

注意：有时题目会隐式的表示二维费用，比如普通的01背包问题，加上一个限定条件，最大装M件物品，这就是给每件物品加了一个“件数”的费用。

实现：和一维费用的处理一样

背包问题6 分组的背包问题

问题：共有N个物品，每个物品数量为1，每个物品的大小为C[i]，每个物品的价值为W[i]，这些物品被划分为若干组，每组中的物品互相冲突，最多选一件，背包的容量为V，背包所能装下的物品总价值最大为多少？

动态规划状态定义：f[k][v]表示前k组物品话费费用v能取得的最大价值

动态规划转移方程：f[k][v]=max{f[k-1][v],f[k-1][v-C[i]]+W[i] | i属于第k组}

背包问题7 有依赖的背包问题

问题描述：共有N个物品，每个物品数量为1，每个物品的大小为C[i]，每个物品的价值为W[i]，所有物品不是主件就是附件，一个主件可以有多个附件，一个附件只能有一个主件且附件不能再有附件，同时附件如果放入背包中那么主件就一定要放入背包中，即附件依赖于主件，背包的容量为V，背包所能装下的物品总价值最大为多少？

分析：此问题如果按照一般的背包问题分析，将一个主件和它的所有附件中的任意几个放到一起看成一个物品，这些物品的集合看成一组就可以回到背包问题6的情况，但是这样的分配，一组中的组合就会非常多，设有n个附件的话，就有2^(n+1)中组合，是指数级，显然不好。此时可以考虑背包问题6中对一个组中物品的一种简化策略，即所有费用相同的物品只留一个价值最大的物品，不会影响结果(C[i]<=C[j] && W[i]>=W[j]，就可以忽略第j个物品)。根据这一点可以对一组中的所有物品进行一次01背包算法，得到费用为0~V-C[i]的所有组合中价值最大的组合，此时一组中的物品就从原来的x^(n+1)个减少为V-C[i]+1个，之后就可以在进行背包问题6的算法了。

更一般的问题：主件的附件仍然可以具有自己的附件集合，但是每个物品最多只依赖一个物品，且不出现循环依赖现象。此时就不能把每个主件的附件当做01背包问题中的一个物品了，因为主件的附件也可能还有附件，所以应该对主件的所有附件进行分组背包问题算法即背包问题6，求得每个主件的附件组合在所有情况的容量下的最大价值，之后再对这个主件和附件的组合合并为一组，对所有组进行分组背包问题算法求解。

背包问题8 泛化物品

定义：一种物品，他没有固定的费用和价值，而是他的价值随着你分配给他的费用而变化。

严格定义：在背包容量为V的背包问题中，泛化物品时一个定义域为0~V中的整数的函数h，当分配给它的费用为v时，能得到的价值就是h(v)

一个物品组可以看做一个泛化物品h，h(v)表示物品组中所有费用为v的物品的最大价值。背包问题7中的主件和附件相互组合组成的物品组就可以看成一个泛化物品

泛化物品的和：对于两个泛化物品h和l，对给定的费用0~V中的v，最大价值f(v)=max{h(k)+l(v-k) | 0<=k<=v}，f(v)是一个根据v变化的函数，所以f是一个有泛化物品h和l决定的泛化物品。如果泛化物品f满足f(v)=max[h(k)+1(v-k) | 0<=k<=v]，则称f是h和l的和，即f=h+1。

泛化物品性质：在一个背包问题中，若将两个泛化物品代以他们的和，不影响问题的答案