一、01背包

题目 有N件物品和一个容量为V的背包。第i件物品的费用是c[i]，价值是w[i]。

求解将哪些物 品装入背包可使价值总和最大。

基本思路 这是最基础的背包问题，特点是：每种物品仅有一件，可以选择放或不放。 用子问题定义状态：

           即表示前i件物品恰放入一个容量为v的背包可以获得的最大价值。 则其状态转移方程便是：

            f[i][v] =Max { f[i−1][v] ，f[i−1][v−c[i]] + w[i] }

              这个方程非常重要

               把第i个物品在考虑加进去的时候，计算一下，加入这个物品后，背包的价值能不能提高，这就是状态方程              的意义。

|  |
| --- |
| 题意：给出N件物品，背包的容量为V，给出每件物品的重量和价值，求背包容量的限制下，能取到的的最大价值  思路，01背包  最基础的背包问题，特点是：每件物品仅有一件，可以选择放或者不放。用子问题定义状态：  即表示前i件物品恰放入容量为v的背包可以获得的最大价值，其状态状态转换方程为：  f[i][v]=Max{ f[i-1][v],f[i-1][v-c[i]]+v[i]}  #include <iostream>  #include<string.h>  #include<algorithm>  using namespace std;  /\*  核心代码  for(int i=1;i<=n;i++0  {  for(it v=V,v>0;v--)  {  f[v]=max(f[v],f[v-c[i]+w[i]);  }  }  \*/  int main()  {  //cout << "Hello world!" << endl;  int T,N,V;  int bag[1010],v[1010],w[1010];  cin>>T;  int i,j;  while(T--)  {  memset(bag,0,sizeof(bag));  cin>>N>>V;  for(int i=0;i<N;i++)  cin>>v[i];  for(int i=0;i<N;i++)  cin>>w[i];  for(int i=0;i<N;i++)//第i个物品  {  for(j=V;j>=w[i];j--)  {  bag[j]=max(bag[j],bag[j-w[i]]+v[i]);  }  }  cout<<bag[V]<<endl;  }  return 0;  } |

|  |
| --- |
| 2、hdu 2546 饭卡  思路：01背包  把饭卡的余额处理成背包的容量，如果卡的余额小于5元，则直接输出卡的余额  否则首先点N-1份菜，求当背包的容量为M-5时，能点的菜的价格的最大值bag[M-5]，然后用再减去最贵的菜。  #include <iostream>  #include<string.h>  #include<algorithm>  using namespace std;  /\*  核心代码  for(int i=1;i<=n;i++0  {  for(it v=V,v>0;v--)  {  f[v]=max(f[v],f[v-c[i]+w[i]);  }  }  \*/  int main()  {  //cout << "Hello world!" << endl;  int N,M;  int temp;  int bag[1010],v[1010],w[1010];  int i,j;  while(cin>>N&&N!=0)  {  memset(bag,0,sizeof(bag));  for(i=0;i<N;i++)  {  cin>>v[i];  //w[i]=v[i];  }  sort(v,v+N);  cin>>M;  //要将卡内的余额减到5，最后选取最贵的菜  temp=M-5;  //如果卡内的余额小于5，则输出  if(M<5)  {  cout<<M<<endl;  continue;  }  for(i=0;i<N-1;i++)//第i个物品,注意只用N-1个菜  {  for(j=temp;j>=v[i];j--)//容量为j的背包  {  bag[j]=max(bag[j],bag[j-v[i]]+v[i]);//表示把背包装到这么大的时候能获取的最大值  }  }  //最后的值应为卡的余额-bag[temp]-v[N-1]  cout<<M-v[N-1]-bag[temp]<<endl;  }  return 0;  } |

## 二、完全背包

题目 有N种物品和一个容量为V的背包，每种物品都有无限件可用。第i种物品的费用是c[i]， 价值是w[i]。求解将哪               些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最 大。  
基本思路 这个问题非常类似于01背包问题，所不同的是每种物品有无限件。也就是从每种 物品的角度考虑，与它相                  关的策略已并非取或不取两种，而是有取0件、取1件、取2件……等很 多种。如果仍然按照解01背包时                      的思路，令f[i][v]表示前i种物品恰放入一个容量为v的背包的最 大价值。可以按照每种物品不同的                       策略写出状态转移方程：  
                                  f[i][v] = max { f[i−1][ v−k×w[i] ] + k×v[i] }          0 <= k×c[i] <= V

             但是实际代码不是这么直接套用的，而是在当前状态下，继续添加当前物品，直到背包装满。

1. hdu 1114

|  |
| --- |
| 思路：  完全背包。每种硬币可以无限放。但是这里不是求最大价值，而是求最小价值，所以我们先假设所有状态为正无穷，然后不断更新dp，以求得最小价值。这样在动态规划时需要注意一点，状态0始终未0，即f [ 0 ] =0恒成立  #include <iostream>  using namespace std;  const int inf=0x6fffffff;  int f[10002];  /\*  for(int i=0;i<N;i++)  {  for(int j=w[i];j<=V;j++)//V是背包最大容量  {  f[j]=max(f[j],f[j-w[i]]+v[i]);//数组w为重量，v为及价值  }  }  \*/  int main()  {  //  //cout << "Hello world!" << endl;  int T,E,F,M,N,P,W;  int i,j;  cin>>T;  while(T--)  {  for(i=0;i<10002;i++)f[i]=inf;  f[0]=0;//给动态规划一个开头  cin>>E>>F;  M=F-E;  //  cin>>N;  for(i=0;i<N;i++)//第i种硬币  {  cin>>P>>W;  for(j=W;j<=M;j++)//不断往里放钱，取钱最少的方案  {  f[j]=min(f[j],f[j-W]+P);  }  }  if(f[M]==inf)  {  cout<<"This is impossible."<<endl;  }  else  {  cout<<"The minimum amount of money in the piggy-bank is "<<f[M]<<"."<<endl;  }  }  return 0;  } |