#### **具体的建立数组解决问题**

有了前面的限制情况和0,1的分析就可以建立数组了

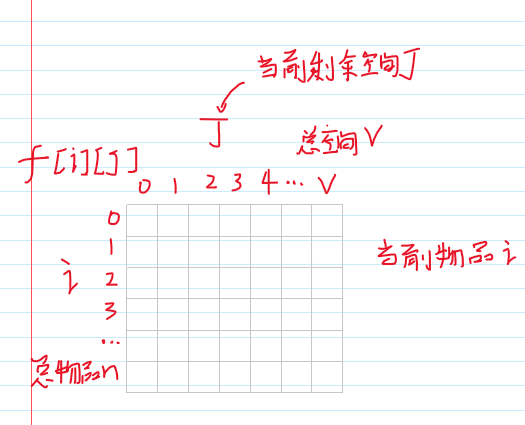
对于这个数组,结合题目要求来说,数组的意义肯定是当前的总价值,也就是第i个物品的总价值,那么题目还有一个限制条件,只靠一个n层的一维数组是不够的,还需要二维数组的横轴来分析当前的剩余容量

所以我们有了一个数组可以来解决问题了,这个数组就叫f好了,然后它是一个二维数组,它的纵轴有i层,我希望它从i=1~n,不想从下标0开始是为了美观,然后这个二维数组的横轴代表着当前剩余的空间,就用j来表示,j=0~V,0就是没有空间的意思,V前面说了,是这个背包的总容量

我们把这个二维数组建立在int main()的上面,所以它一开始全部都是0,省去了接下来赋初值为0的功夫

有了数组f[i][j],然后对于每个f[i][j],它表示的是已经处理到第i个物品了,当剩余空间还有j时,能带有的最大价值,也就是说f[i][j]存储的是总价值

说是总价值,可是涉及到放物品还是不放物品的问题,所以再细致点就是:当前剩余空间为j,用这j空间取分析第i个物品装不装如,处理执行完行为后,f[i][j]就表示了当前能装入的最大价值



#### **5.推导递推方程**

#### **PS:谈一下对于动态规划递推的理解:处理到第i层时,假设前i-1层的数据都知道而且可以根据1~i-1层的数据推出i,那么就成功了一半了,因为第i层如此,那么第i-1层也可以根据1~i-2层推出,接下来只需要定义好数组的初始条件和注意边缘问题以及一些细节就可以了**

对于第i个物品,假设前i-1个物品都已经处理完

如果第i个物品不能放入:这种情况就是背包已经满了,也就是当前剩余空间j小于第i个物品的占用空间C[i],

这种情况下,空间没有变化,价值也没有变化,对于空间没有变化,即第i个物品的空间和第i-1个物品的空间j相同,对于价值没有变化,也就是数组f的值相同,然后开始利用前面的数据,也就是f[i][j]]=f[i-1][j]

如果第i个物品不想放入,那么和不能放入其实是一样的,动机不同但结果相同,f[i][j]]=f[i-1][j]

如果第i个物品放入了,那么f[i][j]=f[i-1][j-c[[i]]+w[i],下面解释一下这个公式,第i个物品的占用空间为c[i],价值为w[i],f[i-1][j-c[[i]]+w[i]表示前i-1个物品在给它们j-c[[i]空间时能带来的最大价值

再回到第i个物品的角度,此时有j个空间,如果已经确定要放入,为了使空间充分利用,肯定是这j个空间只分c[i](刚好够塞下第i个物品),剩下的j-c[[i]全部给前面i-1个物品自由发挥,反正前面f[i-1][j-c[[i]]已经知道了,然后前面i-1个物品用j-c[i]的空间能带来最大的利益f[i-1][j-c[[i]],第i个物品用c[i]的空间带来利益w[i],所以如果第i个物品放入后,总利益是f[i][j]=f[i-1][j-c[[i]]+w[i]

但是,长远来说,有一些偏极端情况,放入这个物品,也许它价值w[i]很高,但是它占用空间c[i]也大,它的性价比可能很低,所以这时候就需要max函数了

当还有空间时:F[i,j] = max[F[i−1,j],F[i−1,j−C[i]] + W[i]

当空间不够时:F[i,j] = F[i−1,j]

下面一个个解释:

当还有空间时:这时有两种方法,放还是不放,如果放,那么利益由两段组成1~i-1是一段,i是另一段;如果不妨,那么利益和上一层剩j空间时相同,这两个东西大小需要比较,因为如果放入,虽然加上了w[i],利益,可是冲击了前i-1个物品的利益,如果不放,那么没有收获到第i个物品的利益,但是把原来属于1~i的空间j,分给了1~i-1个物品,说不定前1~i-1的每个物品都空间小,价值高,性价比高呢?

当空间不够时,它也只能F[i,j] = F[i−1,j]了,没有选择的余地

IMG_257

[IMG_258](https://www.cnblogs.com/zyacmer/p/javascript:void(0);)

#include<bits/stdc++.h>//万能头文件#define ll long longusing namespace std;const ll maxn=100;

ll n,v,f[maxn][maxn];

ll c[maxn];//每个物品占用空间

ll w[maxn];//每个物品的价值int main()

{

cin>>n>>v;

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

scanf("%lld",&c[i]);

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

scanf("%lld",&w[i]);

for(ll i=1;i<=n;i++)//第i个物品

for(ll j=v;j>=0;j--)//剩余空间j {

if(j >= c[i])//如果装得下

f[i][j]=max( f[i-1][j-c[i]]+w[i],f[i-1][j]);

else//如果装不下

f[i][j]=f[i-1][j];

}

cout<<f[n][v]<<endl;//输出答案

}