背包 DP

1、背包基础完全多重混合

dp[i][j]：代表有 i 个物品，背包容量为 j 时，能够承载的最大价值。
w[i]：代表第 i 个物品的重量。
v[i]：代表第 i 个物品的价值。
二维求解状态转移方程：dp[i][j]=max(dp[i-1][j],v[i]+dp[i-1][j-w[i]])

```
#include <bits/stdc++.h>//1-1282 -1    背包基础问题    javacn
using namespace std;

/*
dp[i][j]：代表有 i 个物品，背包容量为 j 时，能够存储的最大价值
对于第 i 个物品：
情况一：背包容量 j< 物品重量 w[j]，放不下，能够存放的价值 dp[i-1][j]
情况二：背包容量 j>= 物品重量 w[j]，放得下
能够得到的最大价值 = max(dp[i-1][j],v[i]+dp[i-1][j-w[i]])
*/
int dp[110][20010];
int n,w[110],v[110],maxw,i,j;
```

```
int main()
{
	cin>>maxw>>n;
	// 读入每个物品的重量和价值
	for(i = 1;i <= n;i++)
	{
		cin>>w[i]>>v[i];
	}

	// 推导
	// 循环 n 个物品
	for(i = 1;i <= n;i++)
	{
		// 循环背包容量从 1~maxw
		for(j = 1;j <= maxw;j++)
		{
			// 如果放不下：背包容量 < 物品重量
			if(j < w[i])
			{
				dp[i][j] = dp[i-1][j];
			}
			else
			{
				// 放得下，就看放进来价值高，还是不放价值高
				dp[i][j] = max(dp[i-1][j],v[i]+dp[i-1][j-w[i]]);
			}
		}
	}

	// 输出 n 个物品，背包容量为 maxw 的情况下最大价值
	cout<<dp[n][maxw];
}
```

一维求解状态转移方程：dp[j]=max(dp[j], dp[j-w[i]]+v[i])

```
#include <bits/stdc++.h>//1-1282 -2    背包基础问题    javacn
using namespace std;

int maxw;// 背包承重
int w,v;// 物品的重量和价值
// 讨论：有 i 个物品背包容量为 j 时能够存放的最大价值
int dp[20010];
int n;

int main()
{
	cin>>maxw>>n;

	// 读入 n 个物品的重量和价值，并计算
	// 循环 n 个物品
	for(int i = 1;i <= n;i++)
	{
		cin>>w>>v;// 读入重量和价值

		// 计算
		// 逆序从背包容量循环到当前物品的重量
		for(int j = maxw;j >= w;j--)
		{
			dp[j] = max(dp[j],v+dp[j-w]);
		}
	}

	cout<<dp[maxw];
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//2-1780    采灵芝    javacn
using namespace std;

/*
完全背包状态转移方程
二维写法：f[i][j] = max(f[i-1][j], f[i][j-w[i]]+v[i])
一维写法：f[j]=max(f[j],f[j-w[i]]+v[i])
一维状态转义方程和 01 背包一致，要注意，完全背包要从前往后推导。
*/
int t,m;
int f[100010];
int ti,vi;// 每个物品的采摘时间和价值

int main()
{
	cin>>t>>m;
	for(int i = 1;i <= m;i++)
	{
		cin>>ti>>vi;
		// 从当前物品的重量（采摘时间）~ 背包容量（最大时间）循环
		for(int j = ti;j <= t;j++)
		{
			f[j] = max(f[j],f[j-ti]+vi);
		}
	}

	cout<<f[t];// 背包能够存储的最大价值
	return 0;
}
```

解法一：将多重背包的 si 个物品分别装入 w 和 v 数组，直接转换为 01 背包！

```
#include <bits/stdc++.h>//3-1888-1    多重背包（1） javacn
using namespace std;

/*
01 背包：每种物品有 1 件
完全背包：每种物品有无限件数
多重背包：每种物品有 Si 件
解题思路：将多重背包转换为 01 背包
将 Si 件物品都存起来，转换为有 Si 个物品，每个物品有 1 件
*/
int n,c;//c 背包容量
int v[10010],w[10010];
int dp[110];
int vi,wi,si,k;//k 代表数组下标
```

```cpp
int main()
{
	cin>>n>>c;
	for(int i = 1;i <= n;i++)
	{
		cin>>vi>>wi>>si;
		// 第 i 个物品有 si 件，都存入数组
		for(int j = 1;j <= si;j++)
		{
			k++;
			v[k] = vi;
			w[k] = wi;
		}
	}

	//01 背包
	for(int i = 1;i <= k;i++)
	{
		// 逆序从背包容量循环到当前物品体积
		for(int j = c;j >= v[i];j--)
		{
			dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
		}
	}

	cout<<dp[c];
	return 0;
}
```

解法二：在做 01 背包时，体现一下有 Si 件物品这个条件。

```
#include <bits/stdc++.h>//3-1888-2    多重背包（1）  javacn
using namespace std;
01 背包：每种物品有 1 件
完全背包：每种物品有无限件数
多重背包：每种物品有 Si 件
解题思路：将多重背包转换为 01 背包
将 Si 件物品都存起来，转换为有 Si 个物品，每个物品有 1 件
int n,c;//c 背包容量
int v[110],w[110],s[110];
int dp[110];
int main()
{
	cin>>n>>c;
	for(int i = 1; i <= n; i++)
	{
		cin>>v[i]>>w[i]>>s[i];
	}
	//01 背包
	// 有 n 个物品
	for(int i = 1; i <= n; i++)
	{
		for(int k = 1;k <= s[i];k++)
		{
			// 逆序从背包容量循环到当前物品体积
			for(int j = c; j >= v[i]; j--)
			{
				dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
			}
		}

	}
	cout<<dp[c];
	return 0;
}
```

关于 DP 要理解的关键点：

1、DP 的本质

求有限的集合中的最值（个数）

本质上，DP 代表了走到阶段 i 的所有路线的最优解；

2、DP 需要思考的点：

（1）DP 的状态是什么？状态要求什么：最大、最小、数量？

（2）DP 的状态计算？

状态转义方程；

求解方法：a、递推 b、考虑阶段 i（最后一个阶段的值）的值是如何得来的；

（3）DP 的边界是什么？

关键术语：阶段、状态、决策（状态转移方程）、边界；

以数塔问题（1216: 【基础】数塔问题）为例，理解 DP 的本质，再理解 01 背包的本质（1282: 【提高】简单背包问题）；

经典的 DP 模板题要熟练掌握，熟记状态转义方程！

本题解题的关键点：二进制优化（类似压缩的思想）

（1）有 n 个不同的物品，要讨论 2n 种选择的可能（每个物品选或者不选）；

（2）一个物品有 n 件，虽然要讨论 2n 种选择的可能，但由于 n 个物品是一样的，那么就减少了讨论数量，比如：有 4 个物品，如果是不同物品的选 2 个，选 1 2、2 3 是不同的选择，但如果是相同的物品，选哪两个就都是一样的了。

因此，n 个物品，要讨论的可能就分别是：选 0 个、选 1 个、选 2 个、选 3 个…选 n 个。

（3）要将 0~n 个不同的选择表达出来，比较简单的方法是将 n 二进制化。

比如：整数 7，只需要用 1 2 4 三个数任意组合，就能组合出 0~7 这 8 种可能。

再比如：整数 10，只需要用 1 2 4 3（注意最后一个数），就能组合出 0~10 这 11 种可能，这样 n 这个值就被二进制化了。

因此如果要讨论 10 个一样的物品，就转化为讨论 4 个不同的物品了；而 n 个一样的物品，就转化为 log₂n 个不同的物品进行讨论。

```
dp[j]=max(dp[j], dp[j-v[i]]+w[i])
```

```cpp
#include <bits/stdc++.h>//4-1889      多重背包（2）      javacn
using namespace std;
const int N = 20010;
int v[N],w[N],dp[2010];
int n,m;//n 种物品，背包容量为 m
int vi,wi,si;
int k = 0;
int main()
{
      cin>>n>>m;
      for(int i = 1;i <= n;i++)
      {
            cin>>vi>>wi>>si;
              对 si 二进制化，比如：有 10 件一样的物品
              我们转换为有 4 件不同的物品：1 2 4 3
              这 4 种物品的体积分别是：1*vi 2*vi 4*vi 3*vi
            int t = 1;// 权重，表示 2 的次方
            while(t <= si)
            {
                  k++;
                  v[k] = t * vi;
                  w[k] = t * wi;
                  si = si - t;
                  t = t * 2;
            }
            // 如果二进制化有剩余，存入
            if(si > 0)
            {
                  k++;
                  v[k] = si * vi;
                  w[k] = si * wi;
            }
      }
```

```
	//01 背包
	for(int i = 1;i <= k;i++)
	{
		for(int j = m;j >= v[i];j--)
		{
			dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
		}
	}

	cout<<dp[m];
	return 0;
}
```

多重背包转换为 01 背包，接下来分 01 背包、完全背包两种情况讨论：

```
#include <bits/stdc++.h>//5-1905    混合背包    javacn
using namespace std;
const int N = 20000;
int v[N],w[N],s[N];
int vi,wi,si;
int k = 0;// 表示存入数组的数据量
int dp[1010];
int n,m;
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		cin>>vi>>wi>>si;
		if(si > 0)		// 如果是多重背包，做二进制拆分
		{
			int t = 1;
			while(t <= si)
			{
				k++;
				w[k] = t * wi;
				v[k] = t * vi;
				s[k] = -1;// 转换为 01 背包
				si = si - t;
				t = t * 2;
			}

			if(si > 0)
			{
				k++;
				w[k] = si * wi;
				v[k] = si * vi;
				s[k] = -1;//01 背包
			}
		}
		else
		{
			k++;
			w[k] = wi;
			v[k] = vi;
			s[k] = si;
		}
	}
```

```
        //计算
        //循环 k 个物品
        for(int i = 1;i <= k;i++)
        {
                //判断是 01 背包还是完全背包
                if(s[i] == -1)
                {
                        for(int j = m;j >= v[i];j--)
                        {
                                dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
                        }
                }
                else
                {
                        for(int j = v[i];j <= m;j++)
                        {
                                dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
                        }
                }
        }

        cout<<dp[m];
        return 0;
}
```

2、二维费用背包

二维费用的背包问题是指：对于每件物品，具有两种不同的费用；选择这件物品必须同时付出这两种代价；对于每种代价都有一个可付出的最大值（背包容量）。问怎样选择物品可以得到最大的价值。设这两种代价分别为代价 1 和代价 2，第 i 件物品所需的两种代价分别为 v[i] 和 w[i]。

两种代价可付出的最大值（两种背包容量）分别为 maxv 和 maxw，物品的价值为 c[i]。

解决方法：费用加了一维，只需状态也加一维即可。

设 f[i][j][k] 表示前 i 件物品付出两种代价分别，背包体积为 j，背包的承重为 k 时可获得的最大价值。

状态转移方程就是： f[i][j][k] = max(f[i-1][j][k],f[i-1][j-v[i]][k-w[i]]+c[i])

空间优化后，可以用二维数组求解。

f[j][k] = max(f[j][k],f[j-v[i]][k-w[i]]+c[i])

```cpp
#include <bits/stdc++.h>//1-2075    最大卡路里    javacn
using namespace std;

const int N = 410;
int dp[N][N];// 代表求解体积为 j，重量为 k 时能够得到的最大价值
int n,v,w,c;
int maxv,maxw;// 背包的上限

int main()
{
	cin>>maxv>>maxw;
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		cin>>v>>w>>c;
		//01 背包
		// 从最大体积 ~ 当前物品体积降序循环，同理重量也要降序循环
		for(int j = maxv;j >= v;j--)
		{
			for(int k = maxw;k >= w;k--)
			{
				dp[j][k] = max(dp[j][k],dp[j-v][k-w]+c);
			}
		}
	}

	cout<<dp[maxv][maxw];// 最大价值
	return 0;
}
```

二维费用背包：创建一个三维数组 yh[n+1][w+1][v+1] 记录在第 n 个物品，w 重量，v 体积时 最优策略

二维费用的背包问题，在 01 背包问题的基础上增加一个费用维度 状态转移方程： dp[i][j][k]=max(dp[i-1][j][k],dp[i-1][j-w[i]][k-v[i]]+c[i])

```
#include <bits/stdc++.h>//2-1949    最大购物优惠    javacn
using namespace std;

/*
二维费用背包：创建一个三维数组 yh[n+1][w+1][v+1] 记录在第 n 个物品，w 重量，v 体积时 最优策略
二维费用的背包问题，在 01 背包问题的基础上增加一个费用维度
状态转移方程：
dp[i][j][k]=max(dp[i-1][j][k],dp[i-1][j-w[i]][k-v[i]]+c[i])
*/
int w,v,n;
// 记录有 i 个物品，承载重量为 j，背包容量为 k 时的最大价值
int dp[110][110][110];
int a[110],b[110],c[110];// 分别代表每个物品的重量、体积、让利金额
string r[110][110][110];// 代表哪了哪些物品
int main()
{
	// 读入数据
	cin>>w>>v>>n;
	for(int i = 1; i <= n; i++)
	{
		cin>>a[i]>>b[i]>>c[i];
	}

	for(int i = 0; i <= n; i++)
	{
		for(int j = 0; j <= w; j++)
		{
			for(int k = 0; k <= v; k++)
			{
				r[i][j][k] = “” ;
			}
		}
	}
```

```
        //循环物品数量
        for(int i = 1; i <= n; i++)
        {
                //循环重量
                for(int j = 1; j <= w; j++)
                {
                        //循环体积
                        for(int k = 1; k <= v; k++)
                        {
                                //承重 和 体积都够，尝试拿这个物品
                                if(j >= a[i] && k >= b[i])
                                {
                                        dp[i][j][k] = max(dp[i-1][j][k],dp[i-1][j-a[i]]
[k-b[i]]+c[i]);
                                        //判断该物品是否拿
                                        //拿了
                                        if(dp[i-1][j][k]<dp[i-1][j-a[i]][k-b[i]]+c[i])
                                        {
                                                r[i][j][k] = r[i-1][j-a[i]][k-b[i]] + (char)
(i + '0') + "";
                                        }
                                        else
                                        {
                                                r[i][j][k] = r[i-1][j][k];
                                        }
                                }
                                else
                                {
                                        //没拿
                                        dp[i][j][k] = dp[i-1][j][k];
                                        r[i][j][k] = r[i-1][j][k];
                                }
                        }
                }
        }
```

```
    //最大让利金额
      cout<<dp[n][w][v]<<endl;
      if(r[n][w][v] !="")           r[n][w][v] = r[n][w][v].substr(0,r[n][w][v].
size()-1);
      cout<<r[n][w][v];//注意结果会有一个尾随空格
      return 0;
}
```

3、有依赖的背包

典型的 01 背包，但要求同一组的物品都要购买。我们可以采用并查集将同一组有关系的礼品的价值、价格汇总到该集合的根节点上，这样就保证了一个集合中的礼品都购买的情况。

```
#include<bits/stdc++.h>//1-1928    采购礼品    javacn
using namespace std;
int f[10100];// 存储物品之间的关系
int q[10100],v[10100];// 价钱、价值
int dp[10100];// 以拥有的钱来定义背包容量
// 查：查询元素的根
int find(int x)
{
	return f[x]==x?x:f[x]=find(f[x]);
}
// 并：合并元素 xy
void merge(int x,int y)
{
	int fx = find(x);
	int fy = find(y);
	if(fx != fy)
	{
		f[fx] = fy;
	}
}
int main()
{
	int n,m,w;
	cin>>n>>m>>w;
	//n 个物品的价钱和价值
	for(int i = 1;i <= n;i++)
	{
		cin>>q[i]>>v[i];
		// 并查集初始化
		f[i] = i;
	}
```

```
    //m 个物品的关系
    int x,y;
    for(int i = 1;i <= m;i++)
    {
        cin>>x>>y;
        merge(x,y);
    }

    // 将有关系的物品合并到这组物品的根上
    for(int i = 1;i <= n;i++)
    {
        // 该物品不是根，则将价钱和价值都合并到根上
        if(f[i] != i)
        {
            q[find(i)]+=q[i];
            v[find(i)]+=v[i];
            // 将该组物品的价钱和价值清零
            q[i]=0;
            v[i]=0;
        }
    }

    //01 背包计算结果
    for(int i = 1;i <= n;i++)
    {
        // 从背包容量（有多少钱）~ 该物品的价钱降序
        for(int j = w;j >= q[i];j--)
        {
            dp[j] = max(dp[j],dp[j-q[i]]+v[i]);
        }
    }
    cout<<dp[w];
    return 0;
}
```

对于每件物品有 4 种情况可以讨论： （1）主件是否购买 （2）主件 + 附件 1 是否购买 （3）主件 + 附件 2 是否购买 （4）主件 + 附件 1+ 附件 2 是否购买 因此状态转移方程为：

f[j] = max(f[j],f[j- 主件价格]+ 主件价值 ,f[j- 主件价格 - 附件 1 价格]+ 主件价值 + 附件 1 价值 ,f[j- 主件价格 - 附件 2 价格]+ 主件价值 + 附件 2 价值 ,f[j- 主件价格 - 附件 1 价格 - 附件 2 价格]+ 主件价值 + 附件 1 价值 + 附件 2 价值)
注意判断：要买的物品的价格 <=j

```
#include <bits/stdc++.h>//2-1820    金明的预算方案    javacn
using namespace std;

/*
1. 只要不超过 N 元钱就行
2. 要买归类为附件的物品，必须先买该附件所属的主件
   每个主件可以有 0 个、1 个或 2 个附件
3. 使每件物品的价格与重要度的乘积的总和最大
价值 = 价格 * 重要度
*/

struct node{
      // 分别表示：主件价格价值、附件 1 价格价值、附件 2 价格价值
      int zv,zp,fv1,fp1,fv2,fp2;
};

node a[100];
int maxn,n;
int dp[35000];
```

```
int main()
{
	cin>>maxn>>n;
	int v,p,q;
	// 读入 n 个物品的信息
	for(int i = 1;i <= n;i++)
	{
		cin>>v>>p>>q;
		// 判断是主件还是附件
		if(q == 0)
		{
			a[i].zv = v;
			a[i].zp = v * p;
		}
		else
		{
			// 说明是 q 号主件的附件
			if(a[q].fp1 == 0)
			{
				a[q].fv1 = v;
				a[q].fp1 = v * p;
			}
			else
			{
				a[q].fv2 = v;
				a[q].fp2 = v * p;
			}
		}
	}

	// 背包计算
```

```
        //背包计算
        for(int i = 1;i <= n;i++)
        {
                if(a[i].zp == 0) continue;//忽略值为 0 的情况

        //分别讨论：买主件，主件 + 附件 1，主件 + 附件 2，主件 + 附件 1+ 附件 2 的情况
                for(int j = maxn;j >= a[i].zv;j--)
                {
                        dp[j] = max(dp[j],dp[j-a[i].zv] + a[i].zp);//买主件
                        if(a[i].zv+a[i].fv1<=j)
                                dp[j] = max(dp[j],dp[j-a[i].zv-a[i].fv1]+a[i].zp+a[i].
fp1);
                        if(a[i].zv+a[i].fv2<=j)
                                dp[j] = max(dp[j],dp[j-a[i].zv-a[i].fv2]+a[i].zp+a[i].
fp2);
                        if(a[i].zv+a[i].fv1+a[i].fv2<=j)
                                dp[j] = max(dp[j],dp[j-a[i].zv-a[i].fv1-a[i].fv2]+a[i].
zp+a[i].fp1+a[i].fp2);
                }
        }

        cout<<dp[maxn];
        return 0;
}
```

4、背包拓展求方案数等

背包问题求方案数量：

对于一个给定了背包容量、物品费用、物品间相互关系（01、完全、多重、依赖等）的背包问题，除了再给定每个物品的价值后求可得到的最大价值外，还可以得到装满背包或将背包装至某一指定容量的方案总数。

对于这类改变问法的问题，一般只需将状态转移方程中的 max 改成 sum 即可。

例如若每件物品均是 01 背包的物品，转移方程即为：f[i][j] = sum(f[i - 1][j],f[i][j-v[i]])

一维数组优化的结果：f[j] = sum(f[j],f[j-v[i]])

初始条件：f[i][0]=1。（也就是什么都不选也是一种方案）

这样做可行的原因在于状态转移方程已经考察了所有可能的背包组成方案。

```cpp
#include <bits/stdc++.h>//1-1890    小明买书    javacn
using namespace std;

long long n,m,v,dp[10010];//v 代表了第 i 个物品的价格
int main()
{
	cin>>n>>m;
	dp[0] = 1;
	for(int i = 1;i <= n;i++)
	{
		cin>>v;
		//01 背包
		for(int j = m;j >= v;j--)
		{
			dp[j] = dp[j] + dp[j-v];
		}
	}

	cout<<dp[m];
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//2-1904    数字的组合    javacn
using namespace std;

/*
n 个不同的物品，第 i 个物品的价格为 ai
选择其中若干物品，其总价格为 m

f(i,j)：代表有 i 个数，要求出和为 j 有多少种不同的方案
第 i 个数有 2 种选择
（1）不要：有 i-1 个数，要求出和为 j -> f(i-1,j)
（2）要：从剩余的 i-1 个数中，求出和为 j-ai -> f(i-1,j-ai)

f(i,j) = f(i-1,j) + f(i-1,j-ai)
f(j) = f(j) + f(j-ai)
*/

int n,m,v;
int f[10010];
int main()
{
	cin>>n>>m;
	//边界
	f[0] = 1;
	for(int i = 1;i <= n;i++)
	{
		cin>>v;
		for(int j = m;j >= v;j--)
		{
			f[j] = f[j] + f[j-v];
		}
	}

	cout<<f[m];
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//3-1891  开心的金明    javacn
using namespace std;
int w[30],v[30],f[50000];//w 数组为重要度，v 数组为 money，f 是用来 dp 的数组
int n,m;//n 是总物品个数，m 是总钱数
int main()
{
	cin>>m>>n;// 输入
	for(int i=1; i<=n; i++)
	{
		cin>>v[i]>>w[i];
		w[i]*=v[i];//w 数组在这里意义变为总收获（重要度 *money）
	}
	//01 背包（参照第二类模板“一维数组优化”）
	for(int i=1; i<=n; i++)
	{
		for(int j=m; j>=v[i]; j--)
		{ // 注意从 m 开始
			if(j>=v[i])
			{
				f[j]=max(f[j],f[j-v[i]]+w[i]);//dp
			}
		}
	}
	cout<<f[m]<<endl;// 背包大小为 m 时最大值
	return 0;
}
```

背包问题求方案数：

```
#include <bits/stdc++.h>//4-1911    背包问题求方案数    javacn
using namespace std;
int n,m;
// 体积恰好是 j 的情况，表示体积要用完
int f[1010];// 记录体积恰好是 j 的情况下的最大价值
int g[1010];// 记录体积恰好是 j 的情况下的方案数
int mod = 1000000000 + 7;
int v,w;
int t,s;
int main()
{
	cin>>n>>m;
	// 背包什么都不装也是一种方案
	for(int i = 0;i <= m;i++) g[i] = 1;
	// 循环物品数
	for(int i = 1;i <= n;i++)
	{
		cin>>v>>w;
		//01 背包循环背包容量 ~ 当前物品的体积
		for(int j = m;j >= v;j--)
		{
			t = max(f[j],f[j-v]+w);// 选和不选的最大价值
			// 判断从哪个决策转移过来的
			s = 0;
			// 两个决策如果都是最优的，那么方案是就是两者之和
			if(t == f[j]) s = (s+g[j])%mod;// 没选是最优的
			if(t == f[j-v] + w) s = (s+g[j-v])%mod;// 选了是最优的
			f[j] = t;
			g[j] = s;
		}
	}
	cout<<g[m];
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//5-2073  码头的集装箱   javacn
using namespace std;

int n,c,w;
int f[40000];

int main()
{
	cin>>n>>c;
	for(int i = 1;i <= n;i++)
	{
		cin>>w;
		for(int j = c;j >= w;j--)
		{
			f[j] = max(f[j],f[j-w]+w);
		}
	}

	cout<<f[c];
	return 0;
}
```

请注意，本题不能贪心解决，比如：背包容量为 10，物品重量为：4 5 6，如果贪心，那么剩余容量为 1，不是最优解！

```
#include <bits/stdc++.h>//6-1779    装箱问题    javacn
using namespace std;
/*
背包容量：v
n 个物品的体积（相当于之前重量的概念）
n 个物品的体积又相当于 01 背包的价值
因为，本题求：能够放入的最大体积
*/
/* 如果第 i 个物品（体积是 w）放入，剩余空间是 j-w
剩余空间能够存储的最大体积：dp[j-w]
因此，第 i 个物品放入之后，能够得到的最大体积 = w + dp[j-w]
*/
//dp[j] = max(dp[j],w + dp[j-w])
// 用来存放有 i 个物品，背包容积为 j 时，能够存储的最大体积
int dp[200100];
int maxw,n,w;//w 代表每个物品的体积
int main()
{
	cin>>maxw>>n;
	// 读入每个物品的体积
	for(int i = 1;i <= n;i++)
	{
		cin>>w;
		// 逆序从背包容积循环到当前物品的体积
		for(int j = maxw;j >= w;j--)
		{
			dp[j] = max(dp[j],w + dp[j-w]);
		}
	}

	cout<<maxw - dp[maxw];
	return 0;
}
```

我们可以把两个集合看作取不取这个数，那么这道题就变成了 0-1 背包问题，设 f[i][j] 为前 i 个数中让和为 j 的方案个数，可以发现方案数 = 不取 i 的方案数 + 取 i 的方案数，前提是能够取 i，即 j > i。

注意：如果选了一个数，那么方案数是不变的，所以状态转移方程为 f[i][j] = f[i-1][j] + f[i-1][j - i] (j > i). 然后发现方案数如果位置不同那么还是算同一个方案，那么问题就是求用 n 个数凑 num/2 的方案数 (num 是和)，当然，如果 num 为奇数则无解。

```
#include <cstdio>//7-1944   集合 Subset Sums      javacn
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
int n,num,f[40][800];
int main()
{
	scanf("%d", &n);
	num = n * (n + 1) / 2;
	if (num % 2)
		printf("0\n");
	else
	{
		f[1][0] = 1;
		f[1][1] = 1;
		for (int i = 2; i <= n; i++)
			for (int j = 0; j <= num; j++)
				if (j > i)
					f[i][j] = f[i - 1][j] + f[i - 1][j - i];
				else
					f[i][j] = f[i - 1][j];
		printf("%d\n", f[n][num / 2]);
	}

	return 0;
}
```

5、完全背包拓展

首页 问题列表 钱币兑换 题解列表

搜索

输入 ID 或标题或来源

#11885- 参考解法

javacn 更新时间： 2021-11-16 22:32:51

解法一：找出状态转移方程，二维数组求解

要凑 5 分：令 dp[i][j]==x 表示用前 i 种硬币构造 j 美分共有 x 种方法。

初始化： dp 为全 0 且 dp[0][0]==1。

状态转移： dp[i][j] = sum(dp[i-1][j] , dp[i][j-val[i]])

sum 是求和， val[i] 是第 i 种硬币的面值。

上述方程 前者是指第 i 值硬币一个都不选， 后者是指至少选 1 个第 i 种硬币。

```cpp
#include <bits/stdc++.h>//1-1885-1   钱币兑换  javacn
using namespace std;
int n,i,j,dp[4][40000];

int main()
{
	int n;
	cin>>n;

	for(int i = 1;i <= 3;i++)
	{
		dp[i][0] = 1;// 初始化
		//j 从 i 开始循环，防止背包容量不够
		for(int j = 1;j <= n;j++)
		{
			if(j < i)
			{
				dp[i][j] = dp[i - 1][j];
			}
			else
			{
				dp[i][j] = dp[i - 1][j] + dp[i][j - i];
			}
		}
	}

	cout<<dp[3][n];
	return 0;
}
```

解法二：使用一维数组滚动求解

```
#include <bits/stdc++.h>//1-1885-2  钱币兑换  javacn
using namespace std;

/*
用 1,2,3，凑出和为 n，有多少种不同的方法
每个数字可以使用多次，因此本题是完全背包
f(i,j)：代表有 i 种数字，求出和为 j 的方案数
f(i,j) = f(i-1,j) + f(i-1,j-ai)
f(j) = f(j) + f(j-ai)
*/

int f[40000];
int n;// 背包容量
int main()
{
	cin>>n;
	f[0] = 1;
	// 循环 3 种物品，第 i 种物品的值是 i
	for(int i = 1;i <= 3;i++)
	{
		for(int j = i;j <= n;j++)
		{
			f[j] = f[j] + f[j-i];
		}
	}

	cout<<f[n];
	return 0;
}
```

完全背包求方案数：

```
#include <bits/stdc++.h>//2-1903      自然数的拆分方案总数    javacn
using namespace std;

/*
背包容量是：N
相当于：
1. 有 N-1 个物品，他们的体积是 1~N-1
2. 选出若干物品，体积的和为 N
3. 每个物品的使用次数没有限制
完全背包的问题！
f(i,j) = f(i-1,j) + f(i-1,j-i)
*/
int n;
int f[4010];

int main()
{
	cin>>n;
	f[0] = 1;//边界
	for(int i = 1;i < n;i++)
	{
		//完全背包
		for(int j = i;j <= n;j++)
		{
			//u 代表 unsigned，否则大整数输出在非 C99 的环境会有警告
			f[j] = (f[j] + f[j-i]) % 2147483648u;
		}
	}
	cout<<f[n];

}
```

本题相当于是有 10 个数（1~10），每个数对应一个价值，这 n 个数可以随意用 0 次或多次，问如果要用这 10 个数，凑出整数 n，最小的价值是多少？

因此，是一个完全背包问题。

```
/*
背包容量是：n（总公里数，可以理解为背包的体积）
有：1~10 公里，走不同的公里数，对应的费用
转换为：
有 10 个物品，第 i 个物品的体积就是 i（wi），价值是 vi（金额）
问：如果要装满背包，最少要花多少钱？
由于：可以无限次换车，因此是完全背包！

求最小：
（1）f（dp）数组要初始化为 0x3f3f3f3f
（2）边界：f[0] = 0

f[j] = min(f[j],f[j-w[i]]+v[i])
*/
```

```cpp
#include <bits/stdc++.h>//3-2072    公交乘车     javacn
using namespace std;
int n;
//w:公里数（1~10）,v:不同公里数的价格
int f[110],w[20],v[20];

int main()
{
	for(int i = 1;i <= 10;i++)
	{
		w[i] = i;//走 i 公里
		cin>>v[i];//走 i 公里的价格
	}
	cin>>n;

	//背包计算，先将值设置无穷大，方便求最小
	memset(f,0x3f,sizeof(f));
	//边界
	f[0] = 0;

	for(int i = 1;i <= 10;i++)
	{
		for(int j = w[i];j <= n;j++)
		{
			f[j] = min(f[j],f[j-w[i]]+v[i]);
		}
	}

	cout<<f[n];

	return 0;
}
```

完全背包求方案数。

```
#include <bits/stdc++.h>//4-2074    货币问题    javacn
using namespace std;

// 完全背包
long long f[5010];
long long n,m,v;

int main()
{
	cin>>n>>m;
	f[0] = 1;// 边界，组成 0 元有 1 种方案
	for(int i = 1;i <= n;i++)
	{
		cin>>v;
		for(int j = v;j <= m;j++)
		{
			f[j] = f[j] + f[j-v];
		}
	}

	cout<<f[m];

	return 0;
}
```

6、多重背包拓展

f[i,j]：有 i 个数字，如果要组合出数字 j，有多少种不同的方法！

问：用当前的数字，能组合出多少种不同的数，相当于再求二维数组最后一行下标为 1~maxn 之间非 0 的元素有几个！

状态转移方程：

```
f[i,j] = f[i-1,j] + f[i-1][j-wi]
```

```
f[j] = f[j] + f[j-wi]
```

```
/*
有 6 种数字：1 2 3 5 10 20，每种有若干个
问：能组合出多少种不同的数字
多重背包问题：直接存入背包转换为 01 背包求解

举例：有 1g 1 个，2g 0 个，3g 2 个
转换 01 背包问题：1 3 3
能组合出 1~7 中的哪些数：1 3 4 6 7
*/
```

```cpp
#include <bits/stdc++.h>//1-1892-1  砝码称重    javacn
using namespace std;
int a[10] = {0,1,2,3,5,10,20};
int maxn;// 背包容量：能够称出的最大重量（所有砝码的重量和）
int f[1010];
int w[1210];// 存储每个砝码
int k = 0;// 表示 w 数组的下标
int main()
{
	int c;
	for(int i = 1;i <= 6;i++)
	{
		cin>>c;
		// 直接存入背包
		for(int j = 1;j <= c;j++)
		{
			k++;
			w[k] = a[i];
			maxn = maxn + a[i];// 砝码重量求和
		}
	}
	// 背包方案数求解
	f[0] = 1;
	for(int i = 1;i <= k;i++)
	{
		for(int j = maxn;j >= w[i];j--)
		{
			f[j] = f[j] + f[j-w[i]];
		}
	}
	int r = 0;  // 求解能够称出多少种不同的重量
	for(int i = 1;i <= maxn;i++)		{	if(f[i] != 0) r++;	}
	cout<<r;
	return 0;
}
```

解法二：二维数组求解

```
#include <bits/stdc++.h>//1-1892-2  砝码称重    javacn
using namespace std;
int f[1500][4010];
int w[1500];
int a[7] = {0,1,2,3,5,10,20};
int k = 0,maxn,c;
int main()
{
	for(int i = 1;i <= 6;i++)
	{
		cin>>c;
		for(int j = 1;j <= c;j++)
		{
			k++;
			w[k] = a[i];
			maxn = maxn + a[i];
		}
	}
	f[0][0] = 1;
	for(int i = 1;i <= k;i++)
	{
		for(int j = 0;j <= maxn;j++)
		{
			if(j < w[i]) f[i][j] = f[i-1][j];
			else f[i][j] = f[i-1][j] + f[i-1][j-w[i]];
		}
	}

	int r = 0;
	for(int i = 1;i <= maxn;i++)		{	if(f[k][i] != 0) r++;	}
	cout<<r;
	return 0;
}
```

多重背包：

```
#include<iostream>//2-2077    奖品采购    javacn
#include<algorithm>
using namespace std;
const int N = 505, M = 6005;
int p[N],v[N],s[N],f[M]={0};

int main()
{
	int n,m;
	cin>>n>>m;
	for (int i=1; i<=n; i++)
		cin>>p[i]>>v[i]>>s[i];

	for (int i=1; i<=n; i++)
		for (int j=m; j>=0; j--)
			for (int k=0; k<=s[i]; k++)
				if (j-k*p[i]>=0)    f[j]=max(f[j],f[j-k*p[i]]+k*v[i]);
	cout<<f[m]<<endl;
	return 0;
}
```