背包 DP

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])

2075. 最大卡路里

神州飞船准备运送一批食品到太空站，该飞船能够运送食品的重量、体积有严格的限制。

现已知 n 件完全不同的食品，每种食品的重量、体积及该食品能够提供的卡路里的值，请你编程计算出，该飞船最多能够运送多少卡路里的食物？

输入：第一行有两个整数，表示飞船装载食物的体积最大值 (<400) 和质量最大值 (<400)；

第二行，一个整数 食品总数 N (<50)；

第三行～第 3+N 行，每行三个数，表示第 i 件食品的体积 (<400) 质量 (<400) 所含卡路里 (<500)。

输出：一个整，表示所能达到的最大卡路里的值（ int 范围内 ）

输入

320 350

4

160 40 120

80 110 240

220 70 310

40 400 22

输出

550

```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;
}
```

1949. 最大购物优惠

小惠听说超市正在打折促销，要制订一个得到最大优惠的购物计划。

小惠的体力可以提起 w 单位重量的东西，还有一个能装 v 个单位体积的购物袋，并详细了解了各打折商品的重量、体积及此商品实际优惠的金额。她想在自己体力的限度和购物袋容积限度内，尽可能多地得到购物优惠。

超市规定这些打折商品每种只能购买一件。

请你编写程序，制定一个购买商品的计划，求出小惠能得到的最大优惠金额和实际应购买的各商品序号。

输入

第一行：依次为 w、v 和 n(n 为商品种类数)，所有数值均为不超过 100 的正整数

接下来的 n 行：每行有三个整数，依次为某种商品的重量、体积和让利金额，数值间以空格分开，所有数值均为不超过 100 的正整数

输出

第一行：小惠能够得到的最大让利金额

第二行：依次为从小到大排列的商品序号，序号从 1 开始，序号间用空格分开。若第二行输出的序列不唯一，则输出其最小字典序。

输入

10 9 4

8 3 6

5 4 5

3 7 7

4 5 4

输出

9

2 4

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;

int dp[110][110][110];// 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 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()
{

      cin>>w>>v>>n;      // 读入数据
      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] = “” ;
                  }
            }
      }
```

```
    for(int i = 1; i <= n; i++)    // 循环物品数量
    {
        for(int j = 1; j <= w; j++)                // 循环重量
        {
            for(int k = 1; k <= v; k++)          // 循环体积
            {
                // 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
                    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];
                }
            }
        }
    }

    cout<<dp[n][w][v]<<endl;    // 最大让利金额
    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

1928. 采购礼品

王老师来到商店为同学们采购礼品。

这家店有 n 种礼品（编号是 1~n ），每种礼品只有 1 件。老板为了促销，对礼品进行搭配销售，有关联性的礼品必须都要采购（奇怪的规定），比如 1 号礼品和 3 号礼品搭配了，3 号和 8 号礼品搭配了，那么王老师想要买 1 号礼品的话，就需要把 3 号和 8 号礼品都买了。

现给定每种礼品的价钱和价值，请问在有限的钱 w 的情况下，能够买到礼品的最大价值是多少？

输入

第一行输入三个整数，n,m,w，表示有 n 种礼品，m 个搭配和你现有的钱的数目。第二行至 n+1 行，每行有两个整数 c、d，表示第 i 种礼品的价钱和价值。（$1 \leqslant c,d \leqslant 10^5$）

第 n+2 至 n+1+m 行 ，每行有两个整数，u、v，表示 u 号礼品和 v 号礼品是有关联的，已经形成搭配销售的关系。

数据范围：$1 \leqslant n,w \leqslant 10^4, 0 \leqslant m \leqslant 5\times10^3$ 。

输出

一行，表示可以获得的最大价值。

输入

5 3 10

3 10

3 10

3 10

5 100

10 1

1 3

3 2

4 2

输出

1

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。

1890. 小明买书

新的学习开始了，小明来到书店采购辅导书。

小明有 M 元，书店有 N 种不同的书，第 i 种书卖 Ai 元，假设小明每种书最多只能买 1 本，且要花完 M 元，请问小明有多少种不同的买书方案？

输入

第一行有两个整数 N 和 M 。（1 ≤ N ≤ 100,1 ≤ M ≤ 10000）。

第二行有 N 个整数 Ai（1 ≤ A i ≤ 1000）。

输出

输出一个整数，代表买书的方案数。（请注意：本题的计算结果可能会超过 int）

输入

4 4

1 1 2 2

输出

3

```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;
		for(int j = m;j >= v;j--)		//01 背包
		{
			dp[j] = dp[j] + dp[j-v];
		}
	}
	cout<<dp[m];
	return 0;
}
```

/*
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)
*/

1904. 数字的组合

给定 N 个正整数 $A_1, A_2, \cdots, A_N$，从中选出若干个数，使它们的和为 M ，求有多少种选择方案。注意：选择不同位置的，但值相同的数，认为是不同的方案。比如：有 3 个数 1 1 1，要组合出 2，那么有 3 个方案，分别是选第 1、2 个数，选第 1、3 个数，选第 2、3 个数。

输入：第一行包含两个整数 N 和 M 。

第二行包含 N 个整数，表示 $A_1, A_2, \cdots, A_N$。

数据范围 $1 \leqslant N \leqslant 100, 1 \leqslant M \leqslant 10000, 1 \leqslant A_i \leqslant 1000$。

输出：包含一个整数，表示可选方案数（本题的计算结果一定在 int 范围内）。

输入

4 4

1 1 2 2

输出

3

```
#include <bits/stdc++.h>//2-1904    数字的组合    javacn
using namespace std;
int n,m,v;
int f[10010];
int main()
{
	cin>>n>>m;
	f[0] = 1;    //边界
	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

1885. 钱币兑换

在一个国家仅有 1 分，2 分，3 分硬币，将钱 N 分 (N<32768) 兑换成硬币有很多种兑法。请你编程序计算出共有多少种兑法。

输入

输入一个正整数 N，N 小于 32768。

输出

输出兑换的方法数。（本题数据的计算结果在 int 范围内）

输入

2934

输出

718831

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   二维数组解法
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

		for(int j = 1;j <= n;j++)		//j从i开始循环，防止背包容量不够
		{
			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;
}
```

```cpp
#include <bits/stdc++.h>//1-1885-2   钱币兑换   javacn   一维数组解法
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;

	for(int i = 1;i <= 3;i++)      // 循环 3 种物品，第 i 种物品的值是 i
	{
		for(int j = i;j <= n;j++)
		{
			f[j] = f[j] + f[j-i];
		}
	}

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

1903. 自然数的拆分方案总数

给定一个自然数 N ，要求把 N 拆分成若干个正整数相加的形式，参与加法运算的数可以重复。注意：拆分方案不考虑顺序，也就是 3=1+2 和 3=2+1 算作相同的方案；至少拆分成 2 个数的和。求拆分的方案数 mod2147483648 的结果。

输入：一个自然数 N。（1 ≤ N ≤ 4000）

输出：输入一个整数，表示结果。

输入 :7

输出 :14

```
#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++)
	{
		for(int j = i;j <= n;j++)		//完全背包
		{
			//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];
}
```

2072. 公交乘车

A 城市有一条非常特别的街道，该街道在每个公里的节点上都有一个公交车站，乘客可以在任意的公交站点上车，在任意的公交站点下车。乘客根据每次乘坐公交的公里数进行付费，比如，下表就是乘客乘坐不同的公里数要付的费用。（请注意：不一定公里数越高，费用越高，这也是这条街道特别的地方）

公里数	1	2	3	4	5	6	7	8	9	10
付费金额	12	21	31	40	49	58	69	79	90	101

一辆公交车单次行驶的公里数一定不超过 10 公里，一个乘客如果打算乘坐公交车完成 n 公里（1 ≤ n ≤ 100）的行程，他可以选择无限次的换车来完成行程。请问，他最少要花多少钱？

输入：第一行十个整数分别表示公交行走 1 到 10 公里的费用（≤ 500）。注意这些数并无实际的经济意义，即行驶 10 公里费用可能比行驶一公里少。

第二行一个整数 n 表示，旅客的总路程数。（1 ≤ n ≤ 100）

输出：仅一个整数表示最少费用。

输入

12 21 31 40 49 58 69 79 90 101

15

输出

147

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;
int f[110],w[20],v[20]; //w: 公里数（1~10）,v: 不同公里数的价格

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;


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

	f[0] = 0;	//边界

	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;
}
```

2074. 货币问题

某国家有 n 种不同面值的货币，第 i 种货币价值 a i 元。

请问：如果每种货币都提供任意多的数量的情况下，如果需要 m 元金额的货币，有多少种不同的方案？

输入：第一行两个整数 n，m（m ≤ 5000，n ≤ 100)；以下 n 行，每行一个整数，第 i+1 行为第 i 种货币的面值。

输出：一个整数，为方案数（方案数≤ 10^{18}）。

输入

3 10

1

2

5

输出

10

```
#include <bits/stdc++.h>//4-2074     货币问题     javacn 完全背包求方案数
using namespace std;
long long f[5010];        // 完全背包
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

1892. 砝码称重

设有 1g、2g、3g、5g、10g、20g 的砝码各若干枚（其总重 ≤1000），求这些砝码能称出的不同重量的个数。

输入

读入 a1,a2,a3,a4,a5,a6 ，分别表示 1g 砝码有 a1 个，2g 砝码有 a2 个，…20g 砝码有 a6 个，（0 ≤每种砝码数量≤ 200）。

输出

整数 N（N 表示用这些砝码能称出的不同重量的个数，但不包括一个砝码也不用的情况）。

输入

1 1 0 0 0 0

输出

3

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  解法一：一维数组求解
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;
		for(int j = 1;j <= c;j++)		// 直接存入背包
		{
			k++;
			w[k] = a[i];
			maxn = maxn + a[i];		// 砝码重量求和
		}
	}
	f[0] = 1;
	for(int i = 1;i <= k;i++)	// 背包方案数求解
	{
		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;
}
```

```cpp
#include <bits/stdc++.h>//1-1892-2  砝码称重   javacn  解法二：二维数组求解
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;
}
```

2077. 奖品采购

为了迎接班级的元旦晚会，班主任王老师特批了 m 元请你帮忙采购物品，并给你一张清单，注明了王老师调研小卖部中出售的 n 个待选物品的价格、价值（价值越高，同学们越喜欢）、以及最多能买到的数量。

请你编程计算出，m 元最多能够采购到的物品的最大价值是多少？注意，m 元不一定都要花完。

输入：第 1 行两个数 n（n ≤ 500），m（m<6000），其中 n 代表清单中待选物品总数，m 表示王老师的拨款金额。

接下来 n 行，每行 3 个数，p、v、s 分别表示第 i 种物品的价格、价值和能够买的最大数量（买 0 件到 s 件均可），其中 p ≤ 100, v ≤ 100, s ≤ 10 。

输出：输出一个整数，表示能够采购到的最大价值。

输入

5 1000

80 20 4

40 50 9

30 50 7

40 30 6

20 20 1

输出

1040

```
#include<iostream>//2-2077    奖品采购    javacn    多重背包
#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;
}
```