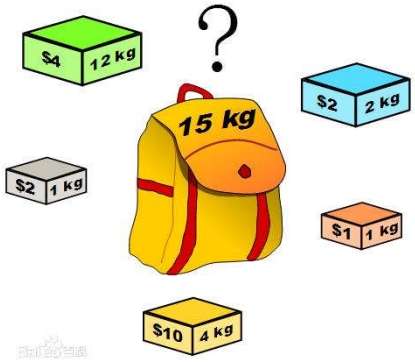
背包动规 1

背包问题是经典的动态规划问题，我们先从 01 背包讲起。

01 背包

有一个背包，最多可以承受 P 的重量，有 n 个物品，每个物品有两个属性，重量和价格，对于第 i 个物品的重量是 w[i]，价格是 v[i]，问怎样放置物品，才能在背包的承重范围内使得背包内的总价格最大。



思路一：要让背包内的价格尽量大，能不能每次挑价格最高的物品放进背包，直到背包放不下物品为止。如上图，背包容量为 15，5 件物品，按照价格排序分别为（10，4），（4，12），（2，1），（2，2），（1，1），那么物品放置情况如下：

	物品 1	物品 2	物品 3	物品 4	物品 5
价格	10	4	2	2	1
重量	4	12	1	2	1
背包价格	10	10	12	14	15
背包重量	4	放不下	5	7	8

这个思路对吗？

显然是有漏洞的。如下反例：

	物品 1	物品 2	物品 3	物品 4	物品 5
价格	6	4	2	2	1
重量	15	12	1	2	1
背包价格	6	6	6	6	6
背包重量	15	放不下	放不下	放不下	放不下

上面的例子背包的价格至少可以达到 8，而按照贪心原则只能达到 6，显然这个方式是不行的。

如果换成按背包重量从小到达排序或者按照性价比（v[i]/w[i]）从大到小排序进行贪心选取，能不能解决这个问题呢？

显然也是不行的。为什么呢？

容易证明，设当前背包放置了 x 件物品，背包剩余容量为 t，那么可能存在一种置换方案，

使得将 x 件物品中的某一件物品 i 替换为未选中的物品 j 和 k，使得 t+w[i]>w[j]+w[k]且 v[i]<v[j]+v[k]。所以，背包问题，本质是枚举所有放置方案的问题。

思路二：所谓 01 背包，即每个物品只能放或者不放进背包，可以用 0 表示不放，用 1 表示放。要解决问题，我们可以利用二进制枚举出所有的放置方案，如 4 件物品的放置方案为：

物品 4	物品 3	物品 2	物品 1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
……			
1	1	1	0
1	1	1	1

代码实现如下：

```
#include<iostream>
using namespace std;
int n,p,w[101],v[101],maxv;
int main(){
    cin>>p>>n;
    for(int i=1;i<=n;i++){
        cin>>w[i]>>v[i];
    }
    int num=1<<n;
    for(int i=0;i<num;i++){ //枚举每一种方案
        int t=i;
        int sw=0;
        int sv=0;
        for(int j=1;j<=n;j++){
            if(t&1){ //当前位为 1
                sw+=w[j]; //累加重量
                sv+=v[j]; //累加价格
            }
            t>>=1;  //去掉当前位
        }
        if(sw<=p&&sv>=maxv)maxv=sv; //统计最大价格
    }
    cout<<maxv;
}
```

那么至此为止，问题解决了吗？显然没有，因为每件物品要枚举放和不放，n 件物品，需要枚举的方案就有 2^n 种，时间复杂度太高了。这里面有哪些可以优化的地方吗？

思路三：以上的枚举方案存在大量的重复子问题。以下表为例，我们首先求出了物品 1 和物

品 2 的放置情况，那么对于第三件物品，如果不放，只需在前两件中取价格最大的就可以，如果放，也可以直接在前两件的统计情况上搭配最优选择就可以了，这样，对于每个物品的放置，只需统计两次就可以了。

物品 3	物品 2	物品 1
	0	0
	0	1
	1	0
	1	1
0		
1		

为了更详细描述这个问题，我们用动态规划的方式来解决。

以物品放置为阶段，我们定义变量 f[i][j]表示前 i 件物品用了背包 j 的容量所获得的最大价值。对于前 i 前 j 的定义，我们的思考点从第 i 第 j 出发（参考二维动态规划讲义）。
考虑当前第 i 件物品：
如果第 i 件物品放置，能够获得 v[i]的价格，代价是占用了 w[i]的容量，那么剩下的容量为 j-w[i]，也就是说对于剩下的 i-1 个物品，只能有 j-w[i]的容量进行物品放置，所以，如果放置第 i 个物品，则问题转化为 f[i][j]=f[i-1][j-w[i]]+v[i]。
如果第 i 个物品不放置，则问题转化为前 i-1 个物品有 j 的容量可以放置物品，即 f[i][j]=f[i-1][j]。对于两种方案，怎么决策呢？当然是选择收益最大的。

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

边界：观察以上方程，当 i 为 1 时，需要访问到 f[i-1]即 f[0]的情况，没有物品选择，收益当然为 0，所以 f[0][i]=0。再观察 j，当 j 为 w[i]的时候，需要访问到 j-w[i]的情况，即 f[i-1][0]的情况，当然，背包容量为 0，收益也为 0，所以 f[i][0]=0。综上，边界为 f[i][0]=f[0][i]=0。

代码实现：

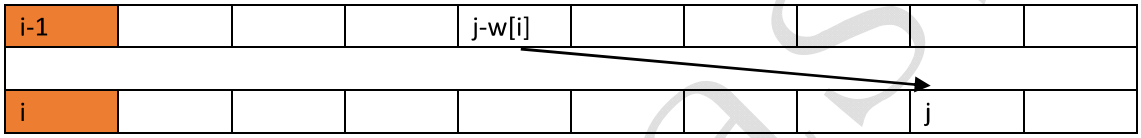
```
#include<iostream>
using namespace std;
int n,p,w[101],v[101],f[101][1001];
int main(){
    cin>>p>>n;
    for(int i=1;i<=n;i++){
        cin>>w[i]>>v[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=0;j<=p;j++){
            f[i][j]=f[i-1][j];//不选
            if(j>=w[i]){//只有背包容量大于等于物品容量时才可以选
```

```
                f[i][j]=max(f[i][j],f[i-1][j-w[i]]+v[i]);
            }
        }
    }
    cout<<f[n][p];
}
```

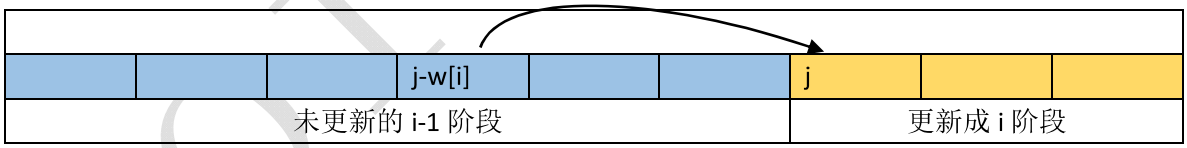
时间复杂度：O(n*p)

思路四：我们从思路二到思路三，时间复杂度从 O(2^n)优化到 O(n*p)，我们还可以继续优化吗？在时间复杂度上我们很难再继续优化了，但是在空间上，我们还可以继续优化。

我们观察状态转移方程，f[i]只与 f[i-1]的状态有关，与 i-1 之前的状态无关，这种情况，我们可以使用滚动数组来存储 i 行和 i-1 行的状态。如用 a 数组表示 i-1 行的状态，b 数组表示 i 行状态，通过 i-1 行求出 i 行，然后将表示 i 行状态的数组 b 重新赋值给 a 数组，这时 a 数组表示的第 i 行的状态，从 a 数组出发，又可以更新 i+1 行的状态。



继续观察动态转移方程，对于当前状态 f[i][j]，在列上，求 j 只与 j 和 j-w[i]有关，j-w[i]小于 j，也就是说，求 f[i][j]，只与上一行左上的状态有关，和上一行右边的状态无关。在这里，我们可以进一步将上面的两个数组，压缩成一个数组，每次从 i-1 行状态更新为 i 行状态时，倒序来求。进入状态 f[j]，那么这个 f[j]相当于 f[i-1][j]，表示不取的情况，f[j-w[i]]相当于 f[i-1][j-w[i]]，那么 f[j]=max(f[j],f[j-w[i]]+v[i])，即完成了 f[j]的更新，这时的 f[j]，对应的就是第 i 阶段的状态，而 j 之前的状态，则还是 i-1 阶段的状态，下次继续更新 j-1 的状态，倒序进行，直到 j-w[i]=0 为止。



代码如下：

```
#include<iostream>
using namespace std;
int n,p,w[101],v[101],f[1001];
int main(){
    cin>>p>>n;
    for(int i=1;i<=n;i++){
        cin>>w[i]>>v[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=p;j>=w[i];j--){  //注意，这里一定是倒序处理
```

oiClass

```
            f[j]=max(f[j],f[j-w[i]]+v[i]);
        }
    }
    cout<<f[p];
}
```

在 01 背包的基础上，还有很多背包问题的变形，如满背包问题，二维背包问题，多重背包和完全背包等。

满背包问题

给定一个背包容量为 p，有 n 个物品，每个物品的体积为 v[i]，问能否将背包放满。

此问题与 01 背包的差异在于物品只有一个属性，我们可以自己增加一个属性，设物品的价值也是 v[i]，那么问题可以转化成 01 背包，即 f[i][j]表示前 i 个物品用背包容量 j 所能获得的最大价值，状态转移方程为 f[i][j]=max(f[i-1][j],f[i-1][j-v[i]]+v[i])，如果 f[n][p]==p 即背包装满了。

此题我们还有另一种实现方式，我们设 f[i][j]表示前 i 个物品能否装满 j 的容量，没错，这里 f[i][j]是一个布尔类型，那么 f[i][j]=f[i-1][j]||f[i-1][j-v[i]]。边界 f[0][0]=true。

代码实现如下：

```
#include<iostream>
using namespace std;
int n,v[101],p;
bool f[10001];
int main(){
    cin>>p>>n;
    for(int i=1;i<=n;i++)cin>>v[i];
    f[0]=true;
    for(int i=1;i<=n;i++){
        for(int j=p;j>=v[i];j--){
            f[j]|=f[j-v[i]];
        }
    }
    if(f[p]==true)cout<<"yes";
    else cout<<"no";
}
```

此方法也用来求一系列组合方案问题。

二维背包问题

给定一个背包，能够装上 p 的重量 g 的体积，有 n 个物品，每个物品有三个属性，重量 w[i]，体积 v[i]和价值 c[i]，从 n 个物品中挑物品装进背包，问背包最终的最大价值是多少。

这个问题，与 01 背包不同的地方在于，背包增加了一个限制条件，为了描述这个限制条件，我们把状态变量多增加一维，设 f[i][j][k]表示前 i 件物品使用了 j 的重量 k 的体积所能获得的最大价值，那么状态转移方程为 f[i][j][k]=max(f[i-1][j][k],f[i-1][j-w[i]][k-v[i]]+c[i])，与 01 背包的解法类似。因为三维的数组对空间要求较大，一般使用状态压缩的方法压缩成二维数组。

代码见《NASA 的食物计划》一题。

采药

【题目描述】
辰辰是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”
如果你是辰辰，你能完成这个任务吗？
【输入】
第一行有两个整数 T（1 <= T <= 1000）和 M（1 <= M <= 100），用一个空格隔开，T 代表总共能够用来采药的时间，M 代表山洞里的草药的数目。接下来的 M 行每行包括两个在 1 到 100 之间（包括 1 和 100）的整数，分别表示采摘某株草药的时间和这株草药的价值。
【输出】
一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。
【样例输入】
70 3
71 100
69 1
1 2
【样例输出】
3
【分析】
01 背包模板题。
【代码】

```
#include<iostream>
using namespace std;
int n,p,w[101],v[101],f[101][1001];
int main(){
    cin>>p>>n;
    for(int i=1;i<=n;i++){
        cin>>w[i]>>v[i];
    }
```

oiClass

```
    for(int i=1;i<=n;i++){
        for(int j=0;j<=p;j++){
            f[i][j]=f[i-1][j];//不选
            if(j>=w[i]){//可选
                f[i][j]=max(f[i][j],f[i-1][j-w[i]]+v[i]);
            }
        }
    }
    cout<<f[n][p];
}
```

【状态压缩版】

```
#include<iostream>
using namespace std;
int n,p,w[101],v[101],f[1001];
int main(){
    cin>>p>>n;
    for(int i=1;i<=n;i++){
        cin>>w[i]>>v[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=p;j>=w[i];j--){
            f[j]=max(f[j],f[j-w[i]]+v[i]);
        }
    }
    cout<<f[p];
}
```

开心的金明

【题目描述】

金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间他自己专用的很宽敞的房间。更让他高兴的是，妈妈昨天对他说：“你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过 N 元钱就行”。今天一早金明就开始做预算，但是他想买的东西太多了，肯定会超过妈妈限定的 N 元。于是，他把每件物品规定了一个重要度，分为 5 等：用整数 1~5 表示，第 5 等最重要。他还从因特网上查到了每件物品的价格（都是整数元）。他希望在不超过 N 元（可以等于 N 元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第 j 件物品的价格为 v[j]，重要度为 w[j]，共选中了 k 件物品，编号依次为 j1，j2，……，jk，则所求的总和为：

v[j1]*w[j1]+v[j2]*w[j2]+ …+v[jk]*w[jk]。（其中*为乘号）

请你帮助金明设计一个满足要求的购物单。

【输入】

输入文件 happy.in 的第 1 行，为两个正整数，用一个空格隔开：

N m
（其中 N（<30000）表示总钱数，m（<25）为希望购买物品的个数。）
从第 2 行到第 m+1 行，第 j 行给出了编号为 j-1 的物品的基本数据，每行有 2 个非负整数
v p
（其中 v 表示该物品的价格(v<=10000)，p 表示该物品的重要度(1~5)）
【输出】
输出文件 happy.out 只有一个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值（<100000000）。
【样例输入】
1000 5
800 2
400 5
300 5
400 3
200 2
【样例输出】
3900
【分析】
01 背包模板题
【代码】

```
#include<iostream>
using namespace std;
int M,n,v[26],p[26],f[30001];
int main(){
    cin>>M>>n;
    for(int i=1;i<=n;i++)cin>>v[i]>>p[i];
    for(int i=1;i<=n;i++){
        for(int j=M;j>=v[i];j--){
            f[j]=max(f[j],f[j-v[i]]+v[i]*p[i]);
        }
    }
    cout<<f[M];
}
```

最大约数和

【题目描述】
一个数的约数和是指这个数所有的约数（不含它本身）之和。
例如：8 的约数和为 1+2+4=7；
9 的约数和为 1+3=4；
现要求选取若干个不同的正整数，使得这些整数之和不超过 S 时，求其约数和之和最大是多少？
【输入】

输入一个正整数 S (S<=1000)。
【输出】
输出最大的约数和之和。
【样例输入】
11
【样例输出】
9
【样例说明】
取数字 4 和 6，可以得到最大值(1+2)+(1+2+3)=9。
【分析】
01 背包问题可以被包装成各种含有限制条件的选择问题，此题即为一个例子。在本题中，应该分析出 s 为背包容量，正整数本身为物品重量，正整数的约束和为物品价值。预处理出正整数的价值和，套用 01 背包模板即可求解。
【代码】

```
#include<iostream>
using namespace std;
int f[1001],a[1001],s;
int getsum(int x){
    int sum=0;
    for(int i=1;i<x;i++){
        if(x%i==0)sum+=i;
    }
    return sum;
}
int main(){
    cin>>s;
    int n=s;
    for(int i=1;i<=n;i++)a[i]=getsum(i);  //a[i]存储 i 的约数和
    for(int i=1;i<=n;i++){
        for(int j=s;j>=i;j--){
            f[j]=max(f[j],f[j-i]+a[i]);
        }
    }
    cout<<f[s];
}
```

买稻草

【题目描述】
农民 john 面临一个很可怕的事实，因为防范失措他存储的所有稻草给澳大利亚蟑螂吃光了，他将面临没有稻草喂养奶牛的局面。在奶牛断粮之前，john 拉着他的马车到农民 Don 的农场中买一些稻草给奶牛过冬。已知 john 的马车可以装的下 C(1 <= C <=50,000)立方的稻草。农民 Don 有 H(1 <= H <= 5,000)捆体积不同的稻草可供购买，每一捆稻草有它自己的体积(1 <=

V_i <= C)。面对这些稻草 john 认真的计算如何充分利用马车的空间购买尽量多的稻草给他的奶牛过冬。
现在给定马车的最大容积 C 和每一捆稻草的体积 Vi，john 如何在不超过马车最大容积的情况下买到最大体积的稻草？他不可以把一捆稻草分开来买。
【输入】
第一行两个整数，分别为 C 和 H
第 2..H+1 行:每一行一个整数代表第 i 捆稻草的体积 Vi
【输出】
一个整数，为 john 能买到的稻草的体积。
【样例输入】
7 3
2
6
5
【样例输出】
7
【分析】
满背包问题。注意剪枝优化，如果背包放满了，就可以直接退出。
【代码】

```
#include<iostream>
using namespace std;
int f[50001],v,h,x;
int main(){
    cin>>v>>h;
    for(int i=1;i<=h;i++){
        cin>>x;
        for(int j=v;j>=x;j--){
            f[j]=max(f[j],f[j-x]+x);
            if(f[j]==v){  //优化，背包放满了就直接退出
                cout<<v<<endl;
                return 0;
            }
        }
    }
    cout<<f[v];
}
```

NASA 的食物计划

【题目描述】
NASA(美国航空航天局)因为航天飞机的隔热瓦等其他安全技术问题一直大伤脑筋,因此在各方压力下终止了航天飞机的历史,但是此类事情会不会在以后发生，谁也无法保证,在遇到这类航天问题时,解决方法也许只能让航天员出仓维修,但是多次的维修会消耗航天员大量的能

量,因此 NASA 便想设计一种食品方案,让体积和承重有限的条件下多装载一些高卡路里的食物。
航天飞机的体积有限,当然如果载过重的物品,燃料会浪费很多钱,每件食品都有各自的体积、质量以及所含卡路里,在告诉你体积和质量的最大值的情况下,请输出能达到的食品方案所含卡路里的最大值,当然每个食品只能使用一次。
【输入】
第一行 两个数 体积最大值(<400)和质量最大值(<400)
第二行 一个数 食品总数 N(<50).
第三行一第 3+N 行
每行三个数 体积(<400) 质量(<400) 所含卡路里(<500)
【输出】
一个数 所能达到的最大卡路里(int 范围内)
【样例输入】
320 350
4
160 40 120
80 110 240
220 70 310
40 400 220
【样例输出】
550
【分析】
二维背包模板题。
【代码】

```
#include<iostream>
using namespace std;
int V,M,n,v[51],m[51],c[51],f[401][401];
int main(){
    cin>>V>>M>>n;
    for(int i=1;i<=n;i++){
        cin>>v[i]>>m[i]>>c[i];
    }
    for(int i=1;i<=n;i++){
        for(int j=V;j>=v[i];j--){
            for(int k=M;k>=m[i];k--){
                f[j][k]=max(f[j][k],f[j-v[i]][k-m[i]]+c[i]);
            }
        }
    }
    cout<<f[V][M];
}
```

金明的预算方案

【题目描述】
金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间金明自己专用的很宽敞的房间。更让他高兴的是，妈妈昨天对他说："你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过 N 元钱就行"。今天一早，金明就开始做预算了，他把想买的物品分为两类：主件与附件，附件是从属于某个主件的，下表就是一些主件与附件的例子：

主件	附件
电脑	打印机，扫描仪
书柜	图书
书桌	台灯，文具
工作椅	无

如果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。附件不再有从属于自己的附件。金明想买的东西很多，肯定会超过妈妈限定的 N 元。于是，他把每件物品规定了一个重要度，分为 5 等：用整数 1~5 表示，第 5 等最重要。他还从因特网上查到了每件物品的价格（都是 10 元的整数倍）。他希望在不超过 N 元（可以等于 N 元）的前提下，使每件物品的价格与重要度的乘积的总和最大。
设第 j 件物品的价格为 v[j]，重要度为 w[j]，共选中了 k 件物品，编号依次为 j1，j2，……，jk，则所求的总和为：v[j1]*w[j1]+v[j2]*w[j2]+ …+v[jk]*w[jk]。（其中*为乘号）
请你帮助金明设计一个满足要求的购物单。
【输入】
第 1 行，为两个正整数，用一个空格隔开：N m（其中 N（<32000）表示总钱数，m（<60）为希望购买物品的个数。）
从第 2 行到第 m+1 行，第 j 行给出了编号为 j-1 的物品的基本数据，每行有 3 个非负整数 v p q（其中 v 表示该物品的价格（v<10000），p 表示该物品的重要度（1~5），q 表示该物品是主件还是附件。如果 q=0，表示该物品为主件，如果 q>0，表示该物品为附件，q 是所属主件的编号）
【输出】
只有一个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值（<200000）。
【样例输入】
1000 5
800 2 0
400 5 1
300 5 1
400 3 0
500 2 0
【样例输出】
2200
【分析】
本题是带限制条件的 01 背包问题。首先，预处理好所有主件和附件，枚举每一件主件，对于该主件，可以不选，也可以只选主件，也可以主件加附件一，也可以主件加附件二，也可以主件加附件一和附件二，对这五种方案选最大的即可。
【代码】

oiClass

```cpp
#include<iostream>
using namespace std;
struct data{
    int v,w;
}a[61][3];
int M,m,n,v,p,q,t[61],f[61][32001],nxt[61];
int main(){
    cin>>M>>m;
    for(int i=1;i<=m;i++){
        cin>>v>>p>>q;
        if(q==0){  //主件
            a[++n][0]=(data){v,p};
            t[n]++;
            nxt[i]=n;
        }else{  //附件放在主件后面
            a[nxt[q]][t[nxt[q]]++]=(data){v,p};
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=M;j++){
            f[i][j]=f[i-1][j];
            if(j-a[i][0].v>=0)f[i][j]=max(f[i][j],f[i-1][j-a[i][0].v]+a[i][0].v*a[i][0].w);
            if(j-a[i][0].v-a[i][1].v>=0)f[i][j]=max(f[i][j],f[i-1][j-a[i][0].v-a[i][1].v]+a[i][0].v*a[i][0].w+a[i][1].v*a[i][1].w);
            if(j-a[i][0].v-a[i][1].v-a[i][2].v>=0)f[i][j]=max(f[i][j],f[i-1][j-a[i][0].v-a[i][1].v-a[i][2].v]+a[i][0].v*a[i][0].w+a[i][1].v*a[i][1].w+a[i][2].w*a[i][2].v);
        }
    }
    cout<<f[n][M];
}
```