背包九讲：前六讲

1. 0-1 背包问题 （只选一次）
2. 完全背包问题 （无限次）
3. 多重背包问题 （有限次）
4. 混合背包问题
5. 二维费用背包问题
6. 分组背包问题 （组内互斥）
7. 背包问题求方案数
8. 最优背包方案
9. 有依赖的背包问题

1. 0-1 背包

二维动态规划

$f[i][j]$：前 i 件物品，总体积为 j 时最大价值是多少

↓

答案 $\max(f[n][0\sim v])$

$f[i][j]$：

① 不选第 i 件物品 $f[i][j]=f[i-1][j]$

② 选第 i 件物品 $f[i][j]=f[i-1][j-v[i]]$

∴ $f[i][j]=\max(①,②)$

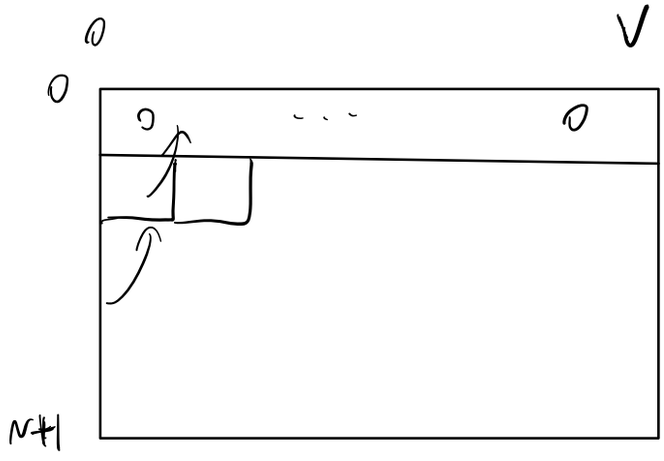
初始化 $f[0][0]=0$

伪码实现

```
N,V = [int(x) for x in input().split()]
v = []
w = []
for i in range(N):
    tmpv,tmpw = [int(x) for x in
input().split()]
    v.append(tmpv)
    w.append(tmpw)

dp = [[0]*(V+1) for _ in range(N+1)]
for i in range(1,N+1):
    for j in range(0,V+1):
        dp[i][j] = dp[i-1][j]
        if v[i-1]<=j:
            dp[i][j] = max(dp[i][j],dp[i-1]
[j-v[i-1]]+w[i-1])

print(max(max(dp)))
```



每一行只依赖于上一行，所以可以进行状态压缩

```
N,V = [int(x) for x in input().split()]
v = []
w = []
for i in range(N):
    tmpv,tmpw = [int(x) for x in
input().split()]
    v.append(tmpv)
    w.append(tmpw)

dp = [0]*(V+1)
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        dp[j] = max(dp[j],dp[j-v[i-1]]
+w[i-1])

print(dp[-1])
```

伪码优化：① 状态压缩

② 最后求最值

```
N,V = [int(x) for x in input().split()]
v = []
w = []
for i in range(N):
    tmpv,tmpw = [int(x) for x in
input().split()]
    v.append(tmpv)
    w.append(tmpw)

dp = [0]*(V+1)
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        dp[j] = max(dp[j],dp[j-v[i-1]]
+w[i-1])

print(dp[-1])
```

循环方式探究

为什么要倒着循环？

如果正着循环，此处对于第i行（二维dp中）后面的j会用到前面更新过的

而本质上在二维dp中，此处应该用到i-1行的，倒序保证后面不作更新

如何找恰好拼出体积m的最大价值？

↓

初始化时 { dp[0] = 0
dp[i] = -INF otherwise

↓

其他位置都要从前面转移

2. 完全背包问题

对物品使用次数无限制

转移（难点）

```
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        dp[j] = max(dp[j],dp[j-v[i-1]]
w[i-1])
```

只需将 i / j 两者顺序调换

```
N,V = [int(x) for x in input().split()]
v = []
w = []
for i in range(N):
    tmpv,tmpw = [int(x) for x in
input().split()]
    v.append(tmpv)
    w.append(tmpw)

dp = [0]*(V+1)
for j in range(1,V+1):
    dp[j] = dp[j-1]
    for i in range(0,N+1):
        if j>=v[i-1]:
            dp[j] = max(dp[j],dp[j-v[i-1]]
+w[i-1])

print(dp[-1])
```

```
N,V = [int(x) for x in input().split()]
dp = [0]*(V+1)
for i in range(N):
    tmpv,tmpw = [int(x) for x in
input().split()]
    for j in range(tmpv,V+1):
        dp[j] = max(dp[j],dp[j-tmpv]+tmpw)
print(dp[-1])
```

优化 ↑ 不断更新 → dp数组

与0-1背包的区别：

在更新 dp 时，j 的更新顺序

3. 多重背包问题：对物品使用次数限制

[未优化版本]

$O(N^3)$

```
N,V = [int(x) for x in input().split()]
v = []
w = []
s = []
for i in range(N):
    tmpv,tmpw,tmps = [int(x) for x in
input().split()]
    v.append(tmpv)
    w.append(tmpw)
    s.append(tmps)

dp = [0]*(V+1)
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        for k in range(0,s[i-1]+1):
            if j>=v[i-1]*k:
                dp[j] = max(dp[j],dp[j-v[i-1]*k]
+k*w[i-1])
```

→ 此处多加一层循环

多重背包 → 0-1背包？

拆分

较笨 → s拆s份

较聪明 → 二进制

拆分 假设限制最多10份

10可以拆分成

2^0 2^1 2^2 $10-\text{sum}$

Sum

→ 拆此优化时间复杂度

(已转换成0-1背包问题)

```
N,V = [int(x) for x in input().split()]
v = []
w = []
for i in range(N):
    tmpv,tmpw,tmps = [int(x) for x in
input().split()]
    k = 1
    while(tmps>k):
        tmps -= k
        v.append(tmpv*k)
        w.append(tmpw*k)
        k *= 2
    if tmps>0:
        v.append(tmps*tmpv)
        w.append(tmps*tmpw)

N = len(v)

dp = [0]*(V+1)
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        dp[j] = max(dp[j],dp[j-v[i-1]]+w[i-1])
```

运用以上拆分方法

成功降低时间复杂度

$N \times V \times \log(S)$

多重背包
- 常规 $O(N\times V\times S)$
- 二进制优化 $O(N\times V\times \log(S))$
- 单调队列 $O(N\times V)$

比较 ①, ②

$$\begin{cases} f(0) \\ f(V)-W \\ f(2V)-2W \end{cases}$$

留3个坑

单调队列 《男人八题》

单调队列

```
dp = [0]*(V+1)
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        for k in range(0,s[i-1]+1):
            if j>=v[i-1]*k:
                dp[j] = max(dp[j],dp[j-v[i-1]*k]
```

把所有体积归类

按模 ... 归为V类

V类两两相互独立

① $f[j]=f[j-v]+w,\ f[j-2*v]+2w,\ \cdots,\ f[j-k*v]+kw$

↗ 框向后物 $\Rightarrow$ 单调队列

② $f[j+v]=f[j]+w,\ f[j-v]+2w,\ \cdots,\ f[j-(k-1)*v]+kw$

4.混合背包问题

- 0,1背包 → 存
- 完全背包 → 存
- 多重背包 → 拆（二进制优化）

内层根据背包选择对应的循环就

```
from collections import namedtuple
Item = namedtuple("Item",["kind","v","w"])
N,V = [int(x) for x in input().split()]
item = []
for i in range(N):
    tmpv,tmpw,tmps = [int(x) for x in input().split()]
    if tmps == 0:
        item.append(Item(0,tmpv,tmpw))
    elif tmps < 0:
        item.append(Item(-1,tmpv,tmpw))
    else:
        k = 1
        while(tmps>k):
            tmps -= k
            item.append(Item(-1,k*tmpv,k*tmpw))
            k *= 2
        if tmps>0:
            item.append(Item(-1,tmps*tmpv,tmps*tmpw))



dp = [0]*(V+1)
for i in item:
    if i.kind == 0:
        for j in range(i.v,V+1):
            dp[j] = max(dp[j],dp[j-i.v]+i.w)
    elif i.kind == -1:
        for j in range(V,i.v-1,-1):
            dp[j] = max(dp[j],dp[j-i.v]+i.w)

print(dp[-1])
```

5. 二维费用背包问题

除容量/加重量限制

[数组f[ci][cj]]

```
N,V,M = [int(x) for x in input().split()]
v = []
m = []
w = []
for i in range(N):
    tmpv,tmpm,tmpw = [int(x) for x in
input().split()]
    v.append(tmpv)
    m.append(tmpm)
    w.append(tmpw)

dp = [[0]*(M+1) for _ in range(V+1)]
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        for k in range(M,m[i-1]-1,-1):
            dp[j][k] = max(dp[j][k],dp[j-v[i-1]][k-
m[i-1]]+w[i-1])

print(dp[-1][-1])
```

[相当于01背包的变种]

6. 分组背包问题

每组物品有 s 个，就会有 s+1 种决策

```
N,V = [int(x) for x in input().split()]
dp = [0]*(V+1)

for i in range(N):
    gs = int(input())
    w = []
    v = []
    for s in range(gs):
        tmpv,tmpw = [int(x) for x in input().split()]
        w.append(tmpw)
        v.append(tmpv)
    for j in range(V,0,-1):
        for k in range(gs):
            if j>=v[k]:
                dp[j] = max(dp[j],dp[j-v[k]]+w[k])

print(dp[-1])
```

对每一个体积

循环判断是否应该替换

最内层遍历，保证每组只取一个