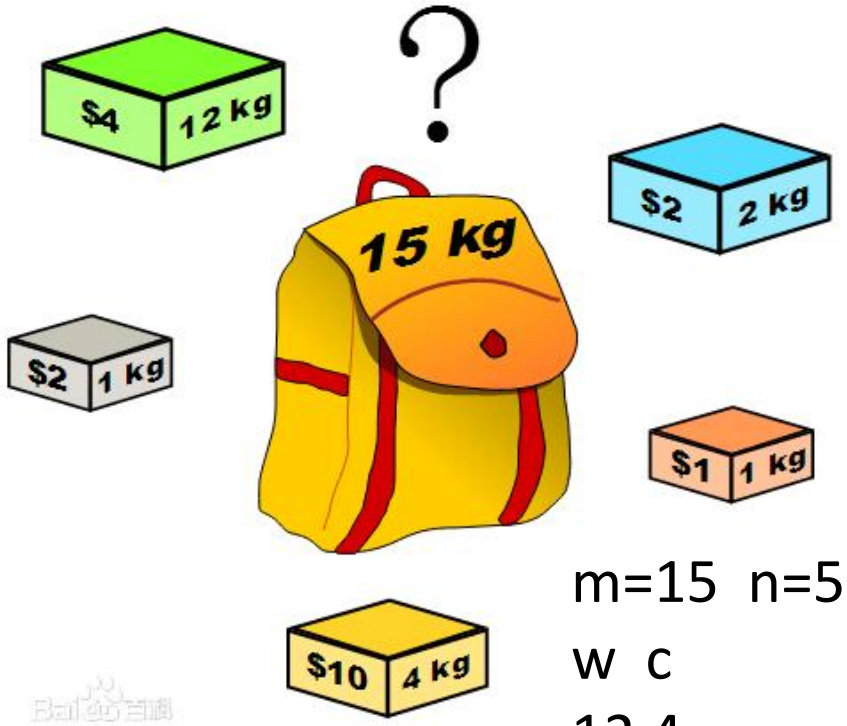
动态规划-背包问题

- 0-1背包、完全背包、多重背包

一. 0-1背包



m=15 n=5
w c
12 4
2 2
1 2
1 1
4 10

有一个背包，最大载重量为m（或体积V）。

有n种货物：重量为 w[i](<1000)（或体积）；

价值为 c[i](<1000).

今从 n 种物品中选取若干件放入背包，使其重量的和不超过m，而所选货物的价值的和为最大。

n<=1000,m<1000.(有的n<=100,m<10000)

求最大价值。

每件物品不放（0）或者放（1）（每种物品就1件）

方法1：暴力求解

每种物品都有选和不选两种决策：

递归枚举：

```
void dfs(int i,int j,int s){
//对物品i进行决策，j剩余空间，s获得价值
      if(i==n+1){
            ans=max(ans,s);
            return;
      }
      dfs(i+1,j,s);//不选物品i
      if(j>=w[i])dfs(i+1,j-w[i],s+c[i]);//能装的下就选物品i
}
dfs(1,m,0);//从第一件物品开始枚举是放还是不放。
```

时间：$O(2^n)$

二进制枚举：

```
int k=1<<n;
for(int i=0;i<k;i++){
    int W=0,C=0;
    for(int j=0;j<n;j++)
        if(i&(1<<j)){
            W+=w[j+1];
            C+=c[j+1];
        }
    if(W<=m)ans=max(ans,C);
}
cout<<ans<<endl;
```

枚举的时间：：时间$O(2^n)$

适合n<=20

算法2：设计表格

n=4 m=10

编号	1	2	3	4
容量w	4	3	5	7
价值c	15	7	20	25

物品0--4

			体积										
序号	w	c	0	1	2	3	4	5	6	7	8	9	10
1	4	15	0										
2	3	7	0										
3	5	20	0										
4	7	25	0										

物品个数4，背包容量是10，为什么考虑:
1,2,3个物品，体积1,2..的情况呢？

算法2:

设f[i][j]:从1到i件物品中选若取干件放到容量为j的背包中，获得的最大价值。目标是：f[n][m]

n=4 m=10

编号	1	2	3	4
容量w	4	3	5	7
价值c	15	7	20	25

物品0--4

			体积										
序号	w	c	0	1	2	3	4	5	6	7	8	9	10
1	4	15	0	0	0	0	15	15	15	15	15	15	15
2	3	7	0	0	0	7	15	15	15	22	22	22	22
3	5	20	0	0	0	7	15	20	20	22	27	35	35
4	7	25	0	0	0	7	15	20	20	25	27	35	35

用f[i][j]表示在第 1 到第i件物品中选择若干件到载重量为j的背包中所能获得的最大价值.

1) f[i-1][j]:不放第i件物品获得的价值。

2) f[i-1][j-w[i]]+c[i]:

放第i件的价值。条件: j>=w[i]

方程1: f[i][j]=max{ f[i-1][j] ,

f[i-1][j-w[i]]+c[i]:(j>=w[i]) }

(1<=i<=n,1<=j<=m)

目标: f[n,m];

非常重要，基本上所有跟背包相关的问题的方程都是由它衍生出来的！

设f[i][j]:从1到i件物品中选若取干件放到容量为j的背包中，获得的最大价值。目标是：f[n][m]

f	j=0	...	j-w[i]	...	j	...	m
i=0	0	...	0	...	0	...	0
i=1							
...	...						
i-1	0		f[i-1][j-w[i]]	+c[i]	f[i-1][j]		
i					f[i][j]		
...	...						
n	0						f[n][m]

实现1:

```
for(int i=1;i<=n;i++)
      for(int j=0;j<=m;j++){
            f[i][j]=f[i-1][j];
            if(j>=w[i])f[i][j]=max(f[i][j],f[i-1][j-w[i]]+c[i]);
      }
```

j的循环顺序无关紧要，因为依靠上一行

方程2:

f[i][j]=max{ f[i-1][j-k*w[i]]+k*c[i]}

(k=0..1:不选与选;j>=k*w[i])

实现2:

```
for(int i=1;i<=n;i++)  //阶段
      for(int j=0;j<=m;j++)  //状态
            for(int k=0;k<=1;k++) //决策
            if(j>=k*w[i])f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*c[i]);
```

空间优化:滚动数组实现:

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

f[i][j]只与前一行有关系，所以可以滚动数组（随时更新，无需保留以前的）：
floyed也是用的滚动数组

f	j=0	...	j-w[i]	...	j	...	m
i=0	0	...	0	...	0	...	0
i=1							
...	...						
i-1	0		f[i-1][j-w[i]]	+c[i]	f[i-1][j]		
i					f[i][j]		
...	...						
n	0						f[n][m]

方法1：两个一维数组

f[1][j]:i是奇数；f[0][j]:i是偶数

```
int f[2][201];
int n,m;
int main(){
    cin>>m>>n;
    for(int i=1;i<=n;i++)cin>>w[i]>>c[i];
    for(int i=1;i<=n;i++)
        for(int j=0;j<=m;j++){
            f[1&i][j]=f[1&(i-1)][j];
            if(j>=w[i])f[1&i][j]=max(f[1&i][j],f[1&(i-1)][j-w[i]]+c[i]);
        }
    cout<<f[1&n][m]<<endl;
    return 0;
}
```

方法2：一维数组（但注意更新顺序：倒着更新）

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

f[i][j]只与前一行有关系，所以可以滚动数组（随时更新，无需保留以前的）：floyed也是用的滚动数组

j的枚举顺序必须从大到小，理解原因：

右边的f[j]是原来的f[i-1][j],左边的f[j]是更新后的是f[i][j]

方程3：f[j]=max{f[j],f[j-w[i]]+c[i]}

实现3：

```
for(int i=1;i<=n;i++)
        for(int j=m;j>=w[i];j--)
                f[j]=max(f[j],f[j-w[i]]+c[i]);
cout<<f[m]<<endl;
```

			体积										
序号	w	c	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	4	15	0	0	0	0	15	15	15	15	?		
2	3	7	0										
3	5	20	0										
4	7	25	0										

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

如果j从小到大，则当i=1时：

f[4]=15,f[5]=15,f[6]=15,f[7]=15

f[8]=max(f[8],f[4]+15)=max(0,15+15)=30,显然是错的,物品1装了2次

因为f[4]已经是15了，已经装了一次了，如果倒着求，在求 f[8]时，f[4]还这一轮没更新，就不会错了。正向求，i会被装多次。f[12]=45,装了3次。（正好有了后面的完全背包问题）

0-1背包的实现：

```
for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++){
        f[i][j]=f[i-1][j];
        if(j>=w[i])f[i][j]=max(f[i][j],f[i-1][j-w[i]]+c[i]);
    }

for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++)
        for(int k=0;k<=1;k++)
        if(j>=k*w[i])f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*c[i]);

for(int i=1;i<=n;i++)
    for(int j=m;j>=w[i];j--)
        f[j]=max(f[j],f[j-w[i]]+c[i]);
```

0-1背包训练题目：

P1048	yt1290	采药	01背包
P2871	yt1294	Charm Bracelet	01背包
P1049	yt1295	装箱问题	01背包
	yt1291	数字组合	01背包求方案数

yt1291：数字组合方案数

有n个正整数，找出其中和为t(t也是正整数)的可能的组合方式。如：
n=5,5个数分别为1,2,3,4,5，t=5；
那么可能的组合有5=1+4和5=2+3和5=5三种组合方式。
输入】
输入的第一行是两个正整数n和t，用空格隔开，其中1≤n≤20,表示正整数的个数，t为要求的和(1≤t≤1000)；
接下来的一行是n个正整数，用空格隔开。
【输出】
和为t的不同的组合方式的数目。
【输入样例】
5 5
1 2 3 4 5
【输出样例】
3

背包问题的方案总数

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背包中的物品;

转移方程由:

f[i][v]=max{f[i-1][v],f[i-1][v-w[i]]+c[i](v>w[i]}

变为:

f[i][v]=f[i-1][v]+f[i-1][v-w[i]] （v>=w[i]）

初始条件f[0][0]=1。

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。

```
cin>>n>>t;
for(int i=1;i<=n;i++)cin>>a[i];
f[0][0]=1;
for(int i=1;i<=n;i++)
    for(int j=0;j<=t;j++){
        f[i][j]=f[i-1][j];
        if(j>=a[i])f[i][j]+=f[i-1][j-a[i]];
    }
cout<<f[n][t]<<endl;
```

或者：

```
cin>>n>>t;
for(int i=1;i<=n;i++)cin>>a[i];
f[0]=1;
for(int i=1;i<=n;i++)
    for(int j=t;j>=a[i];j--)
        f[j]=f[j]+f[j-a[i]];
cout<<f[t]<<endl;
return 0;
```

二. 完全背包

有n种物品和一个容量为m的背包，每种物品都有无限件可用。

第i种物品的费用是w[i]，价值是c[i]。

求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

yt1268 【例9.12】完全背包问题

【题目描述】
设有n种物品，每种物品有一个重量及一个价值。但每种物品的数量是无限的，同时有一个背包，最大载重量为M，今从n种物品中选取若干件(同一种物品可以多次选取)，使其重量的和小于等于M，而价值的和为最大。
【输入】
第一行：两个整数，M(背包容量，M≤200)和N(物品数量，N≤30)；
第2..N+1行：每行二个整数Wi,Ci，表示每个物品的重量和价值。
【输出】
仅一行，一个数，表示最大总价值。
【输入样例】
10 4
2 1
3 3
4 5
7 9
【输出样例】
max=12

基本思路：

这个问题非常类似于01背包问题，所不同的是每种物品有无限件。也就是从每种物品的角度考虑，与它相关的策略已并非取或不取两种，而是有取0件、取1件、取2件……最多v/w[i]件。

如果仍然按照解01背包时的思路，令f[i][v]表示前i种物品恰放入一个容量为v的背包的最大权值。仍然可以按照每种物品不同的策略写出状态转移方程：

f[i][v]=max{f[i-1][v-k*w[i]]+k*c[i]|0<=k<= v/w[i]}。

将01背包问题的基本思路加以改进，可以推及其它类型的背包问题。

0-1背包的实现：

```
for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++){
        f[i][j]=f[i-1][j];
        if(j>=w[i])f[i][j]=max(f[i][j],f[i-1][j-w[i]]+c[i]);
    }

for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++)
        for(int k=0;k<=1;k++)
        if(j>=k*w[i])f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*c[i]);

for(int i=1;i<=n;i++)
    for(int j=m;j>=w[i];j--)
        f[j]=max(f[j],f[j-w[i]]+c[i]);
```

完全背包的实现1:

方程1: f[i][j]=max{ f[i-1][j] ,

f[i-1][j-w[i]]+c[i]:(j>=w[i]) }

0-1背包

完全背包

方程1: f[i][j]=max{ f[i-1][j] ,

f[i][j-w[i]]+c[i]:(j>=w[i]) }

0-1背包

f	j=0	. . .	j-w[i]	. . .	j	. . .	m
i=0	0	. . .	0	. . .	0	. . .	0
. . .	. . .						
i-1	0		f[i-1][j-w[i]]	+c[i]	f[i-1][j]		
i					f[i][j]		
. . .	. . .						
n	0						f[n][m]

完全背包

f	j=0	. . .	j-w[i]	. . .	j	. . .	m
i=0	0	. . .	0	. . .	0	. . .	0
. . .	. . .						
i-1	0				f[i-1][j]		
i			f[i][j-w[i]]	+c[i] →	f[i][j]		
. . .	. . .						
n	0						f[n][m]

完全背包的实现1：

```
for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++){
        f[i][j]=f[i-1][j];
        if(j>=w[i])f[i][j]=max(f[i][j],f[i-1][j-w[i]]+c[i]);
    }
```

0-1背包

完全背包

```
for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++){
        f[i][j]=f[i-1][j];
        if(j>=w[i])f[i][j]=max(f[i][j],f[i][j-w[i]]+c[i]);
    }
```

完全背包的实现2:

f[i][j]=max{ f[i-1][j-k*w[i]]+k*c[i]}

(k=0..1:不选与选;j>=k*w[i])

0-1背包

完全背包

f[i][j]=max{ f[i-1][j-k*w[i]]+k*c[i]}

(k=0..j/w[i]; j>=k*w[i])

完全背包的实现2：

```
for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++)
        for(int k=0;k<=1;k++)
        if(j>=k*w[i])f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*c[i]);
```

0-1背包

完全背包

```
for(int i=1;i<=n;i++)
    for(int v=0;v<=m;v++)
        for(int k=0;k<=v/w[i];k++)
            f[i][v]=max(f[i][v],f[i-1][v-k*w[i]]+k*c[i]);
```

完全背包的实现3：

```
for(int i=1;i<=n;i++)
    for(int j=m;j>=w[i];j--)
        f[j]=max(f[j],f[j-w[i]]+c[i]);
```

0-1背包

完全背包

```
for(int i=1;i<=n;i++)
    for(int j=w[i];j<=m;j++)
        f[j]=max(f[j],f[j-w[i]]+c[i]);
```

完全背包求方案数量：

P1616 疯狂的采药

yt1273：【例9.17】货币系统

yt1293：买书

三. 多重背包

有n种物品和一个容量为v的背包。

第i种物品：

体积是w[i]；

价值是c[i]；

有s[i]件可用（当然可以一件也不用）；

求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

【例9.13】庆功会 yt1269

【题目描述】

为了庆贺班级在校运动会上取得全校第一名成绩，班主任决定开一场庆功会，为此拨款购买奖品犒劳运动员。期望拨款金额能购买最大价值的奖品，可以补充他们的精力和体力。

【输入】

第一行二个数n(n≤500)，v(v≤6000)，其中n代表希望购买的奖品的种数，v表示拨款金额。

接下来n行，每行3个数，w、c、s，分别表示第i种奖品的价格、价值（价格与价值是不同的概念）和能购买的最大数量（买0件到s件均可），其中w≤100，c≤1000，s≤10。

【输出】一行：一个数，表示此次购买能获得的最大的价值（注意！不是价格）。

【输入样例】

```
5 1000
80 20 4
40 50 9
30 50 7
40 30 6
20 20 1
```

【输出样例】


```
1040
```

0-1背包的实现：

```
for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++){
        f[i][j]=f[i-1][j];
        if(j>=w[i])f[i][j]=max(f[i][j],f[i-1][j-w[i]]+c[i]);
    }

for(int i=1;i<=n;i++)
    for(int j=0;j<=m;j++)
        for(int k=0;k<=1;k++)
        if(j>=k*w[i])f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*c[i]);

for(int i=1;i<=n;i++)
    for(int j=m;j>=w[i];j--)
        f[j]=max(f[j],f[j-w[i]]+c[i]);
```

方法：采用类似完全背包（2）的方法

第i种物品有s[i]+1种策略：取0件，取1件……取s[i]件。

令f[i][j]表示前i种物品恰放入一个容量为j的背包的最大权值，

则：f[i][j]=max{f[i-1][j-k*w[i]]+ k*c[i]|0<=k<=s[i]}。

时间：O(V*∑s[i])。

```
for(int i=1;i<=n;i++)
    for(int j=0;j<=v;j++)
        for(int k=0;k<=s[i];k++)
            if(j>=k*w[i])f[i][j]=max(f[i][j],f[i-1][j-k*w[i]]+k*c[i]);
```

时间： O(v*∑(s[i[))

滚动数组实现：

```
for(int i=1;i<=n;i++)
    for(int j=v;j>0;j--)
        for(int k=0;k<=s[i];k++)
            if(j>=k*w[i])f[j]=max(f[j],f[j-k*w[i]]+k*c[i]);
```

背包问题有很多变种，学会转换