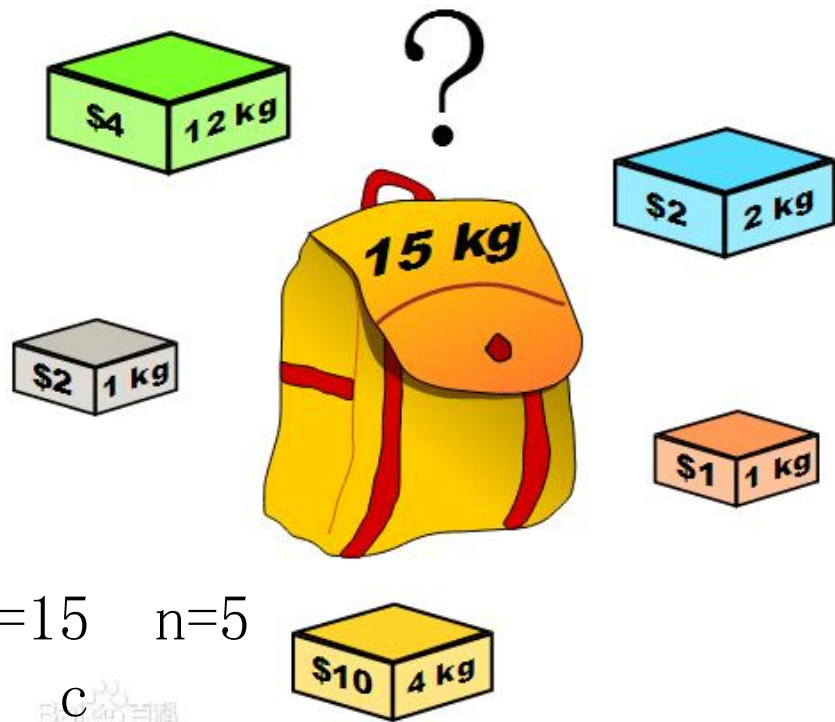


动态规划算法初步4

(4) 背包类模型

- 背包问题九讲

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。

$n \leq 1000, m < 1000$. (有的 $n \leq 100, m < 10000$)

求最大价值。

每件物品不放（0）或者放（1）（每种物品就1件）

输入:

n: 物品种类

m: 背包容积

w[i]: 体积

c[i]: 价值

输入样例1:

4 10

4 3 5 7

15 7 20 25

输出样例1:

35

输入样例2:

4 20

2 9 10 15

2 9 10 16

输出样例2:

19

价值大的?

单位价值大的?

NO

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`
- `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。`

- 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 $n \leq 20$

物品0-4

n=4 m=10

[illegible]

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 v	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
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	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: $j \geq w[i]$

$$\text{方程1: } f[i][j] = \max \left\{ \begin{array}{l} f[i-1][j] \quad , \\ f[i-1][j-w[i]]+c[i] : (j \geq w[i]) \end{array} \right\}$$

$$(1 \leq i \leq n, 1 \leq j \leq m)$$

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

主程序：

- 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; $j \geq k*w[i]$)

- 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]);`

空间优化:滚动数组实现:

- $f[i][j] = \max(f[i-1][j], f[i-1][j-w[i]] + c[i])$
- $f[i][j]$ 只与上一行有关系, 所以可以滚动数组 (随时更新, 无需保留以前的): floyd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 $f[i-1][j]$, 右边的 $f[j]$ 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	←									
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]还是0，就不会错了。

（正好有了后面的完全背包问题）

0-1背包训练题目：

1290	采药	01背包
1291	数字组合	01背包计数
1294	Charm Bracelet	01背包
1295	装箱问题	01背包

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。

- 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……等很多种。
- 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，像这样：
- $f[i][v] = \max \{ f[i-1][v-k*w[i]] + k*c[i] \mid 0 \leq k*w[i] \leq v \}$ 。
- 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。

1268 【例9.12】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]);
```

完全背包的实现1:

$$\text{方程1: } f[i][j] = \max \left\{ \begin{array}{l} f[i-1][j] \quad , \\ f[\textcolor{red}{i}-1][j-w[i]]+c[i] : (j \geq w[i]) \end{array} \right\}$$

0-1背包

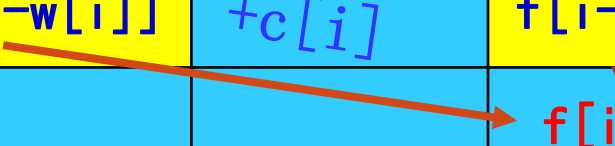


完全背包

$$\text{方程1: } f[i][j] = \max \left\{ \begin{array}{l} f[i-1][j] \quad , \\ f[\textcolor{red}{i}][j-w[i]]+c[i] : (j \geq w[i]) \end{array} \right\}$$

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

区别在哪里?

```
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; $j \geq k*w[i]$)

0-1背包



完全背包

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

($k=0..j/w[i]$; $j \geq k*w[i]$)

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

区别在哪里?

```
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

区别在哪里?

```
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]);
```

- 完全背包问题转化为01背包问题来解。
- 最简单的想法是，考虑到第 i 种物品最多选 $V/w[i]$ 件，于是可以把第 i 种物品转化为 $V/w[i]$ 件费用及价值均不变的物品，然后求解这个01背包问题。这样完全没有改进基本思路的时间复杂度，但这毕竟给了我们完全背包问题转化为01背包问题的思路：将一种物品拆成多件物品。
- 高效的转化方法是：把第 i 种物品拆成费用为 $w[i]*2^k$ 、价值为 $c[i]*2^k$ 的若干件物品，其中 k 满足 $w[i]*2^k < V$ 。这是二进制的思想，因为不管最优策略选几件第 i 种物品，总可以表示成若干个 2^k 件物品的和。这样把每种物品拆成 $O(\log(V/w[i]))+1$ 件物品，是一个很大的改进。后面多重背包也用到这种方法。

完全背包训练：

- 1273： 【例9.17】 货币系统
- 1293： 买书

三. 多重背包

- 有 n 种物品和一个容量为 V 的背包。
- 第 i 种物品最多有 $s[i]$ 件可用，每件费用是 $w[i]$ ，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。

【例9.13】庆功会 1269

- 【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】
- 第一行二个整数 n ($n \leq 500$)， v ($v \leq 6000$)，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 $w \leq 100$ ， $c \leq 1000$ ， $s \leq 10$ 。
- 【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

方法1：采用类似完全背包的方法（2）

- 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： $f[i][j]=\max\{f[i-1][j-k*w[i]]+k*c[i] \mid 0 \leq k \leq s[i]\}$ 。
复杂度是 $O(V*\sum s[i])$ 。


```
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]);
```

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]);
```

方法2：转换为0-1背包（滚动数组）：

第*i*件物品，看做*s[i]*件一样的物品*i*，变成了*s[1]+...+s[n]*件物品的0-1背包，每个背包取还是不取，和方法1有区别的。

```
for (int i=1; i<=n; i++)  
    for (int j=1; j<=s[i]; j--) //s[i] 个物品都是i的01背包  
        for (int k=v; k>=w[i]; k--)  
            f[k]=max(f[k], f[k-w[i]]+c[i]);
```

复杂度是 $O(V * \sum s[i])$

方法3：方法2的改进

- 二进制思想
- 如： $s[i]=13$, 没有必要增加13个物品，只需增加4件物品即可：
- 价格和价值分别乘系数：
- 1, 2, 4, 6
- $(1, 2, \dots, 2^{(k-1)}, s[i]-(2^k-1))$ 因为 $2^k-1=1+2+4+\dots+2^{(k-1)}$
- 1, 2, 4, 6能组出1到13的任意数，等价13件物品
- 时间复杂度： $O(V*\sum(\log(s[i])))$

```

for (int i=1; i<=n; i++) {
    int x, y, z, k=1;
    cin >> x >> y >> z;
    for (k=1; z>0; k=2*k) {
        int d=min(k, z);
        if (d>0) {
            w[++sn]=d*x;
            c[sn]=d*y;
        }
        z=z-k;
    }
}

```

```

}

```

```

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

```

建议：

- 0-1背包，完全背包，多重背包都采用一维的滚动数组实现。

0-1背包 `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 `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]);`

多重背包 `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]);`

 `for (int i=1; i<=n; i++)`
 `for (int j=1; j<=s[i]; j--) //s[i] 个物品都是i的01背包`
 `for (int k=v; k>=w[i]; k--)`
 `f[k]=max(f[k], f[k-w[i]]+c[i]);`

四. 混合背包问题

- 如果将0-1背包、完全背包、多重背包混合起来。
- 也就是说，有的物品只可以取一次（01背包），有的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）。
- 应该怎么求解呢？

1270: 【例9.14】混合背包

- 如果有3种背包，可以：
- 完全背包单独处理
- 0-1背包和多重背包都可以化为0-1背包处理。
- 结构：
- ```
for(int i=1;i<=n;i++) {
 • if完全背包{

 • }
 • else{01或多重背包

 • }
}
```

## 五. 二维费用背包

- 1292: 宠物小精灵之收服

- 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 $a[i]$ 和 $b[i]$ 。
- 两种代价可付出的最大值（两种背包容量）分别为 $V$ 和 $U$ 。
- 物品的价值为 $c[i]$ 。

- 算法：
- 费用加了一维，只需状态也加一维即可。设 $f[i][v][u]$ 表示前 $i$ 件物品付出两种代价分别为 $v$ 和 $u$ 时可获得的最大价值。
- 状态转移方程就是：
- $f[i][v][u] = \max \{f[i-1][v][u], f[i-1][v-a[i]][u-b[i]] + c[i]\}$ 。
- 如前述方法，可以只使用二维的数组：当每件物品只可以取一次时变量 $v$ 和 $u$ 采用逆序的循环，当物品有如完全背包问题时采用顺序的循环。当物品有如多重背包问题时拆分物品。

- 物品总个数的限制：
- 有时，“二维费用”的条件是以这样一种隐含的方式给出的：  
最多只能取M件物品。这事实上相当于每件物品多了一种“件数”的费用，每个物品的件数费用均为1，可以付出的最大件数费用为M。换句话说，设 $f[v][m]$ 表示付出费用v、最多选m件时可得到的最大价值，则根据物品的类型（01、完全、多重）用不同的方法循环更新，最后在 $f[0..V][0..M]$ 范围内寻找答案。
- 另外，如果要求“恰取M件物品”，则在 $f[0..V][M]$ 范围内寻找答案。

## 二维背包训练：

- 1271: 【例9.15】潜水员

## 六. 分组背包

- 1272: 【例9.16】分组背包

- 有N件物品和一个容量为V的背包。第i件物品的费用是 $w[i]$ ，价值是 $c[i]$ 。这些物品被划分为若干组，每组中的物品互相冲突，最多选一件。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。
- 算法
- 这个问题变成了每组物品有若干种策略：是选择本组的某一件，还是一件都不选。也就是说设 $f[k][v]$ 表示前k组物品花费费用v能取得的最大权值，则有 $f[k][v] = \max \{f[k-1][v], f[k-1][v-w[i]]+c[i] \mid \text{物品} i \text{属于第} k \text{组}\}$ 。
- 使用一维数组的伪代码如下：
- for 所有的组k
- for  $v=V..0$
- for 所有的i属于组k
- $f[v] = \max \{f[v], f[v-w[i]]+c[i]\}$
- 注意这里的三层循环的顺序，“for  $v=V..0$ ”这一层循环必须在“for 所有的i属于组k”之外。这样才能保证每一组内的物品最多只有一个会被添加到背包中。



## 七. 背包问题的方案总数

- 对于一个给定了背包容量、物品费用、物品间相互关系（分组、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：
$$f[i][v] = \max \{f[i-1][v], f[i-1][v-w[i]] + c[i] \mid v > w[i]\}$$
- 变为：
$$f[i][v] = f[i-1][v] + f[i-1][v-w[i]] \mid v > w[i]$$
- 初始条件 $f[0][0] = 1$ 。
- 事实上，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。

- 1291: 数字组合
- 1293: 买书
- 1273: 【例9.17】货币系统