注:续讲主要依据前面的背包问题,续讲部分的分析(无具体原理)较简洁，不讲可自行学习

混合背包问题

问题导入：有 N 种物品和一个容量是 V 的背包.物品一共有三类：

- 第一类物品只能用 1 次(0/1 背包)；
- 第二类物品可以用无限次(完全背包)；
- 第三类物品最多只能用 s_i 次(多重背包).

每种体积是 v_i，价值是 w_i，求解将哪些物品装入背包,可使物品体积总和不超过背包容量,且价值总和最大.输出最大价值.其中 $s_i=-1$ 表示第 i 种物品只能用 1 次；$s_i=0$ 表示第 i 种物品可以用无限次；$s_i>0$ 表示第 i 种物品可以使用 s_i 次.

分析：　代码链接

思路一： 0/1 背包就用 0/1 背包,完全背包就用完全背包,多重背包就用多重背包.

```
for(int i=1;i<=n;i++){
    if(s[i]==-1)//01 背包
        for(int j=V;j>=v[i];j--)
            dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
    else if(s[i]==0)//完全背包
        for(int j=v[i];j<=V;j++)
            dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
    else{//多重背包
        int vv[MAXN],ww[MAXN],cnt=0;
        for(int k=1;k<=s[i];k*=2){
            s[i]-=k;
            vv[++cnt]=k*v[i];
            ww[cnt]=k*w[i];
        }
        if(s[i]>0)
            vv[++cnt]=s[i]*v[i],ww[cnt]=s[i]*w[i];
        for(int k=1;k<=cnt;k++)
            for(int j=V;j>=vv[k];j--)
                dp[j]=max(dp[j],dp[j-vv[k]]+ww[k]);
    }
}
```

思路二：将 0/1 背包划分为特殊的多重背包(0/1 背包可看作最多只选 1 个的多重背包)，完全背包用完全背包.

```
for(int i=1;i<=n;i++){
    if(s[i]==0)//完全背包
        for(int j=v[i];j<=V;j++)
            dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
    else{//多重背包+01 背包
        if(s[i]==-1)
            s[i]=1;
        int vv[MAXN],ww[MAXN],cnt=0;
        for(int k=1;k<=s[i];k*=2){
            s[i]-=k;
            vv[++cnt]=k*v[i];
            ww[cnt]=k*w[i];
        }
        if(s[i]>0)
            vv[++cnt]=s[i]*v[i],ww[cnt]=s[i]*w[i];
        for(int k=1;k<=cnt;k++)
            for(int j=V;j>=vv[k];j--)
                dp[j]=max(dp[j],dp[j-vv[k]]+ww[k]);
    }
}
```

二维费用的背包问题

问题导入：有 N 件物品和一个**容量**是 V 的背包，背包能承受的最大**重量**是 M . 每件物品只能用一次. 体积是 v_i , 重量是 m_i , 价值是 w_i . 求解将哪些物品装入背包，可使物品总体积不超过背包容量，总重量不超过背包可承受的最大重量，且价值总和最大. 输出最大价值.

代码链接

分析：跟 0/1 背包很像，只不过在 0/1 背包的基础上增加了重量限制.

0/1 背包的状态转移方程为：$dp[i][j]=\max(dp[i-1][j],dp[i-1][j-v_i]+w_i)$

多了重量限制，则需要再开一维，变成三维

$$dp[i][j][k]=\max(dp[i-1][j][k],dp[i-1][j-v_i][k-m_i]+w_i)$$

同 0/1 背包，可压缩成二维，即 $dp[j][k]=\max(dp[j][k],dp[j-v_i][k-m_i]+w_i)$

```
for(int i=1;i<=n;i++)
    for(int j=V;j>=v[i];j--)
        for(int k=M;k>=m[i];k--)
            dp[j][k]=max(dp[j][k],dp[j-v[i]][k-m[i]]+w[i]);
```

背包问题的方案问题

问题 1 导入：有 N 件物品和一个容量是 V 的背包. 每件物品只能使用一次. 第 i 件物品的体积是 v_i，价值是 w_i. 求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总体积最大. 输出**最优选法的方案数**. 注意答案可能很大，输出答案模 10^9+7 的结果.

分析：代码链接

0/1 背包状态转移方程：$dp[i][j]=\max(dp[i-1][j],dp[i-1][j-v_i]+w_i)$

路径跟踪 $g[i][j]$ 表示：考虑前 i 个物品，当前已使用的体积恰好是 j 的，且价值为最大的方案.

路径跟踪 $g[i][j]$ 属性：方案数量 $count$.

状态转移过程：

✓ 如果 $dp[i][j]=dp[i-1][j]$ 且 $dp[i][j]=dp[i-1][j-v_i]+w_i$，则 $g[i][j]=g[i-1][j]+g[i-1][j-v_i]$；

✓ 如果 $dp[i][j]=dp[i-1][j]$ 且 $dp[i][j]\neq dp[i-1][j-v_i]+w_i$，则 $g[i][j]=g[i-1][j]$；

✓ 如果 $dp[i][j]\neq dp[i-1][j]$ 且 $dp[i][j]=dp[i-1][j-v_i]+w_i$，则 $g[i][j]=g[i-1][j-v_i]$

初始状态：$g[0][0]=1$.

朴素写法：——时间复杂度为 $O(nV)$，空间复杂度为 $O(nV)$.

```
g[0][0]=1;
for(int i=1;i<=n;i++)
    for(int j=0;j<=V;j++){
        if(dp[i][j]==dp[i-1][j])
            g[i][j]=(g[i][j]+g[i-1][j])%mod;
        if(j>=v[i]&&dp[i][j]==dp[i-1][j-v[i]]+w[i])
            g[i][j]=(g[i][j]+g[i-1][j-v[i]])%mod;
    }
```

```
    int res=0;
    for(int i=0;i<=V;i++)
        if(dp[n][i]==dp[n][V])
            res=(res+g[n][i])%mod;
```

朴素优化 ——时间复杂度为$O(nV)$，空间复杂度为$O(V)$.

可将g和dp写进一个循环里,再判断dp的转移路径的同时用g跟踪转移路径

然后再用 0/1 背包的朴素优化,消去一维.

```
    g[0]=1;
    for(int i=1;i<=n;i++)
        for(int j=V;j>=v[i];j--){
            int temp=max(dp[j],dp[j-v[i]]+w[i]),sum=0;
            if(temp==dp[j])
                sum=(sum+g[j])%mod;
            if(temp==dp[j-v[i]]+w[i])
                sum=(sum+g[j-v[i]])%mod;
            dp[j]=temp,g[j]=sum;
        }
    int  res=0;
    for(int i=0;i<=V;i++)
        if(dp[i]==dp[V])
            res=(res+g[i])%mod;
```

问题 2 导入：有N件物品和一个容量是V的背包. 每件物品只能使用一次. 第i件物品的体积是v_i, 价值是w_i. 求解将哪些物品装入背包,可使这些物品的总体积不超过背包容量,且总体积最大. 输出**字典序最小的方案**. 这里的字典序是指：所选物品的编号所构成的序列. 物品编号范围是$1 \cdots N$.

分析：　代码链接

假设存在一个包含第1个物品的最优解,为了确保字典序最小那么我们必然要选第一个. 那么问题就转化成从$2 \sim N$这些物品中找到最优解. 之前的$dp[i][j]$记录的都是**前**i个物品总容量为j的最优解,那么我们现在将$dp[i][j]$定义为**从第i个元素到最后一个元素**总容量为j的最优解。接下来考虑状态转移：$dp[i][j] = \max(dp[i+1][j], dp[i+1][j-v_i]+w_i)$.

两种情况,第一种是不选第i个物品,那么最优解等同于从第$i+1$个物品到最后一个元

素总容量为 j 的最优解；第二种是选了第 i 个物品，那么最优解等于当前物品的价值 w_i 加上从第 $i+1$ 个物品到最后一个元素总容量为 $j-v_i$ 的最优解.

计算完状态表示后，考虑如何的到最小字典序的解. 首先 $dp[1][V]$ 肯定是最大价值，那么便考虑能否选取第 1 个物品.

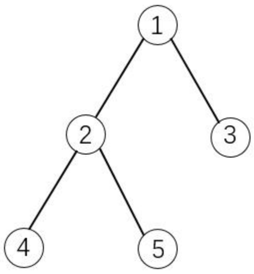
- ✓ 如果 $dp[1][V]=dp[2][V-v_1]+w_1$，说明选取了第 1 个物品可以得到最优解；
- ✓ 如果 $dp[1][V]=dp[2][V]$，说明不选取第一个物品才能得到最优解；
- ✓ 如果 $dp[1][V]=dp[2][V]=dp[2][V-v_1]+w_1$，说明选不选都可以得到最优解，但是为了考虑字典序最小，需要选取该物品.

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

int pos=V;
for(int i=1;i<=n;i++)
    if(pos>=v[i]&&dp[i][pos]==dp[i+1][pos-v[i]]+w[i]){
        printf("%d ",i);
        pos=pos-v[i];
    }
```

有依赖的背包问题（略且难）

问题导入：有 N 个物品和一个容量是 V 的背包. 物品之间具有**依赖关系**, 且依赖关系组成一棵树的形状. 如果选择一个物品, 则必然选择它的父节点. 如下图所示：



如果选择物品 5, 则必然选择物品 1 和 2. 因为 2 是 5 的父节点, 1 是 2 的父节点.

每件物品的编号是 i , 体积是 v_i , 价值是 w_i , 依赖的父节点编号是 p_i (如果 $p_i = -1$ 表示根节点). 物品的下标范围是 $1 \cdots N$. 求解将哪些物品装入背包, 可使物品总体积不超过背包容量, 且总价值最大. 输出最大价值.

分析：略

泛化物品(略)——可参考背包 9 讲