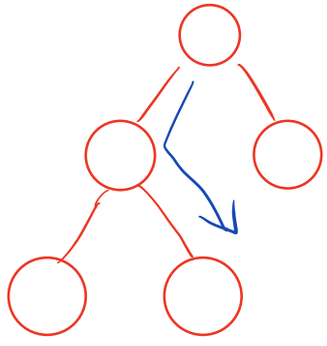
背包九讲（后三讲）

7、有依赖的背包问题



物品 → 体积 → 决策

有依赖的背包问题：

- 每个结点：分组背包中的一个组
- 子结点的每种选择：组内的一个物品

f[i][j] 选择以i为子树的物品，体积不超过v时所获得的最大价值

详解：

① 输入部分

root：根结点

g[fa]：fa的子结点编号数组

② dfs部分

dfs(root) → 递归更新f数组

③ 最终答案

f[root][V]

背包问题 + 树形DP

状态表示 f[i][j]

选结点i，体积为j下，最大收益（以i为根的整个子树）

做法：① 先算出所有子节点的不同体积下的最大价值

② 不同体积对应不同物品组，不同物品组只能选一个

→ 分组背包问题（物品→体积→决策）

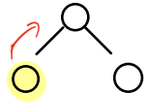
```
int dfs(int x)
{
    for(int i=v[x];i<=m;i++) f[x][i]=w[x];//点x必须选，所以初始化f[x][v[x] ~ m]= w[x]
    for(int i=0;i<g[x].size();i++)
    {
        int y=g[x][i];
        dfs(y);
        for(int j=m;j>=v[x];j--)//j的范围为v[x]~m，小于v[x]无法选择以x为子树的物品
        {
            for(int k=0;k<=j-v[x];k++)//分给子树y的空间不能大于j-v[x],不然都无法选根物品x
            {
                f[x][j]=max(f[x][j],f[x][j-k]+f[y][k]);
            }
        }
    }
}
```

```cpp
int dfs(int x)
{
    for(int i=v[x];i<=m;i++) f[x][i]=w[x];//点x必须选，所以初始化f[x][v[x] ~ m]= w[x]
    for(int i=0;i<g[x].size();i++)
    {
        int y=g[x][i];
        dfs(y);
        for(int j=m;j>=v[x];j--)//j的范围为v[x]~m，小于v[x]无法选择以x为子树的物品
        {
            for(int k=0;k<=j-v[x];k++)//分给子树y的空间不能大于j-v[x],不然都无法选根物品x
            {
                f[x][j]=max(f[x][j],f[x][j-k]+f[y][k]);
            }
        }
    }
}
```


推演：

① 遍历到叶子结点 → 不会进入 [redacted]，则每个都初始化为自身价值

② 遍历到倒数第二层



倒序遍历体积：

正序遍历该体积下，分配给子树的体积

都要预留给根结点体积大小

```cpp
f[x][j]=max(f[x][j],f[x][j-k]+f[y][k]);
```

选择在当前体积下，是应该维持现状，还是加入子结点

初始化

[代码存档]

```
from collections import defaultdict
N,V = [int(x) for x in input().split()]
v = []
w = []
p = defaultdict(list)
# f 节点i的子树在体积为j下的最大价值
f = [[0]*(V+1) for _ in range(N)]
def dfs(s):
    # 初始化，所有体积>=v的对应价值为自身（因为该节点一定入选，根据f的定义）
    for j in range(v[s],V+1):
        f[s][j] = w[s]
    # 遍历所有的叶子结点
    for n in p[s]:
        dfs(n)
        # 遍历所有的体积,此时的体积要大于等于v[s],否则不会有意义（即为0）
        for j in range(V,v[s]-1,-1):
            # 遍历所有决策，对于每一个体积下，分配多少给叶子几点
            for k in range(0,j-v[s]+1):
                f[s][j] = max(f[s][j],f[s][j-k]+f[n][k])

for i in range(N):
    tmpv,tmpw,tmpp = [int(x) for x in input().split()]
    v.append(tmpv)
    w.append(tmpw)
    if tmpp != -1:
        p[tmpp-1].append(i)
    else:
        root = i
dfs(root)
print(f[root][V])
```

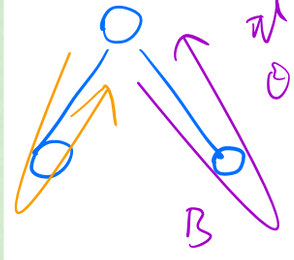
遍历到第二个结点回来后：

对于所有体积：

① 是维持之前的分配方

or

② 还是加入新的叶子？



是否加入B的问题

8. 背包问题求方案数

在0-1背包基础上开一个新的数组：

表示体积为j的情况下方案数为多少

思路：添加一个数组记录体积为j对应的最优解的方案数

```
N,V = [int(x) for x in input().split()]
v = []
w = []
INF = 1e9
for i in range(N):
    tmpv,tmpw = [int(x) for x in input().split()]
    v.append(tmpv)
    w.append(tmpw)
M = 1e9+7
dp = [0 if i==0 else -INF for i in range(V+1)]
g = [1]*(V+1)
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        t = max(dp[j],dp[j-v[i-1]]+w[i-1])
        s = 0
        if dp[j] == t:
            s += g[j]
        if dp[j-v[i-1]] + w[i-1] == t:
            s += g[j-v[i-1]]
        if s > M:
            s -= M
        dp[j] = t
        g[j] = s

tmpmax = -INF
for d in dp:
    if d > tmpmax:
        tmpmax = d
result = 0
for idx,d in enumerate(dp):
    if d == tmpmax:
        result += g[idx]
        if result > M:
            result -= M

print(int(result))
```

9. 背包问题求方案

只要在基础上作一点修改

每次判断，只保留最小字典序

```
from collections import defaultdict
N,V = [int(x) for x in input().split()]
v = []
w = []
INF = 1e9
for i in range(N):
    tmpv,tmpw = [int(x) for x in input().split()]
    v.append(tmpv)
    w.append(tmpw)
dp = [0 if i==0 else -INF for i in range(V+1)]
# 定义一个数组，保存体积为j时候，所有可能的list
g = defaultdict(list)
g[0] = []
for i in range(1,N+1):
    for j in range(V,v[i-1]-1,-1):
        t = max(dp[j],dp[j-v[i-1]]+w[i-1])
        s = [N+1,]
        if dp[j] == t:
            s = min(s,g[j])
        if dp[j-v[i-1]] + w[i-1] == t:
            s = min(s,g[j-v[i-1]]+[i,])
        dp[j] = t
        g[j] = s

tmpmax = -INF
for d in dp:
    if d > tmpmax:
        tmpmax = d
result = [N+1,]
for idx,d in enumerate(dp):
    if d == tmpmax:
        result = min(result,g[idx])

for i in result:
    print(i,end=" ")
```