有依赖性的背包问题

问题简介

> 物品之间存在某种依赖关系，要选择了物品i，就必须选择物品j。将j作为i的父节点。一般所有物品的依赖体系，是一个森林。一样，问，m体积背包下所有方案的最大价值。

分析

- 问题:~~怎么做？ 完全没有思路好吧。~~
  - 定义 $d_{i,j}$ 表示体积为j，在i节点状态下，价值的最大值。
  - 对于一个父节点的选择。针对某一个父节点，有非常多种的策略。他们是独立的。我们针对这一个独立性。得到每一个组的所有方案了。就可以转化为一个分组背包问题。
  - 在众多方案中，有鸽笼原理，在有限的体积下，有非常多不必要的枚举，以及相同花费，价值较小的方案都应该塞掉。
  - 对于一个规模最小的，就是只有一个主见对应的附件都是只能作为附件（不能作为子件）的子树中。我们用01背包问题仅凭$O(n \times V)$的复杂计算出某一花费下的最大价值方案。0~~v就是对所有的合法方案的max。
    $f_{i,j}$就是选择i为基础，i顶点下，j花费以下所有方案最大价值。
  - 往上推更高一级的节点，确保更低节点的各种花费的最大值已经计算整理完成。计算该节点的各种花费下限制下的的max值。为了求取该节点相关的所有合法选举方案中，整理各种花费下的最优方案。该问题就转变为了分组背包问题。
  - 如果只是简单的一颗树，那么，记父节点为k，$f_{k,m}$就是$ans.$
  - 如果是一个森林，直接对每一颗树的祖宗整理出来。做一次分组背包即可。

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 111;
int n, m, root;
int f[maxn][maxn];
int v[maxn], w[maxn], p[maxn]; //分别表示体积花费。

vector<int> g[maxn];

//对于一颗树来说，一是用vector的stl的结构来吧这颗储存下来。
//自定义一个邻接表来存储这一颗树先尝试自己熟悉的方法。
//是否有办法将每一个节点的计算都统一为
//这一个父节点是必须要选的产生了什么影响？和普通的分组背包以及01背包有什么差别？
void dfs(int u) //表明当前在处理什么节点
{
    for (int i = 0; i < g[u].size(); i++)
    {
        int t = g[u][i];
        dfs(t);
        for (int j = m - w[u]; j >= 0; j--)
            for (int k = 0; k <= j; k++)
```

```
                f[u][j] = max(f[u][j], f[u][j - k] + f[t][k]);
    }
    for (int i = m; i >= w[u]; i--)
        f[u][i] = f[u][i - w[u]] + v[u];
    for (int i = 0; i < w[u]; i++)
        f[u][i] = 0;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> n >> m;
    for (int i = 1; i <= n; ++i)
    {
        cin >> w[i] >> v[i] >> p[i];
        if (p[i] == -1)
            root = i;
        else //只需要指向下下面的点就行。对一个点处理不需要计算某个点。
            g[p[i]]
                .push_back(i);
    } //这里初次的构造借结束
    dfs(root);
    cout << f[root][m] << '\n';
}
```

解决问题:

$first$证明，先将只作为附件的点，泛化为m种物品。后对同一主件的点做分组背包。和对同一主见的附件，直接做01背包。最终，是等效的。即两种方向计算出的$f_{u,0.....m}$相同。

- 假设现在各种体积下计算选择第一个组合的最优方案。
  - 条件1：$f_{u,0.....m} = 0.$
  - 对于01背包的方案：计算某一个指标函数（某一个状态对应子问题的解），$f_{u,j}$。

```
当j>=w[t]时。ps:w[t],为当前附件的花费
f[u][j]=max(f[u][j],f[u][j-w[t]]+v[t]);
当j<w[t]时
f[u][j]=f[u][j].
```

  - 对于分组背包问题：

```
f[u][j]=max(f[u][j],f[u][j]+v[t][0],f[u][j-1]+v[t][1].....f[u][j-k]+v[t][k])
假设k<=j.
```

  - 对附件的泛化步骤如下：

```
for (int i = m; i >= w[u]; i--)
    f[u][i] = f[u][i - w[u]] + v[u];
for (int i = 0; i < w[u]; i++)
    f[u][i] = 0;
    对于最底层节点的泛化基本如下
    0 0 0 0 0.......v[t]....v[t]...
    - 第一个是0，紧接若干0再紧接若干v[i].数目可以为0.
```

- 综上不同情形下的比较：

```
first：当j>=w[t]时
01：f[u][j]=v[t];
多重： max运算中有操作对象，有f[u][j-w[t]]+f[t][w[t]].此时k=w[t]
其中 f[t][w[t]]=v[t];
比较发现，当k上移不变。下移减少。所以解决最小规模的子问题后，得到的指标函数是相等的。且有f[u][k]>=f[u][k-1],k合法。
second: j<w[t]
都为0.
```

- 推广步骤，两种方案的$f_{u,j}$小规模指标函数计算结果都相同。且单调递减。

```
若j<w[u];
01背包：
f[u][j]=f[u][j];
多重：
f[u][j]=max(f[u][j],f[u][j-k]+f[t][k]),k<w[u];
此时f[t][k]=0。故f[u][j]=max(f[j][j],f[u][j-k])。
由单调性，f[u][j]>=f[u][j-k].
所以此时 f[u][j]=f[u][j];
若j>=w[u];
同时由于单调性。j>j-k。有
f[u][j]=max(f[u][j],f[u][j-w[t]])
对于多重背包：
只关注 f[t][w[t]];
f[u][j]=max(f[u][j],f[u][j-w[t]]+f[t][w[t]]);
令temp(k)=f[u][j-k]+f[t][k];
当：假设，b>w[t]；t范围合法；
则temp(b)-temp(w[t])=f[t][w[t]]-f[t][w[t]]。
由单调性temp(k)-temp(b)>=0;
temp(k)>=temp(b);
综上只有两种选择方案：是两个划分的最大值
f[t][0],f[t][w[i]]。
对于01解法
f[u][j]=max(f[u][j],f[u][j-w[i]]+v[i]);
对于多重
f[u][j]=max(f[u][j],f[u][j-0]+0,f[u][j-w[i]]+f[t][w[i]]);
显然最终结果相等。
综上两种处理得到的结果是一样的。
```