树上背包及其优化详解

星夜
理解万岁

关注她

> 本文大致介绍了树上背包的 $\mathrm{O}(nm^2)$ 暴力dp方法和若干 $\mathrm{O}(nm)$ 优化方法。

问题描述

给定一棵 n 个节点的有根树，编号为 1 到 n， 1 号点为根节点且每个节点 i 有价值 w_i ，体积 v_i 。背包容量为 m ，在满足如果 u 选了，则其所有祖先 v 都要选的限制下，使选择的物品组合总价值和最大。

下文中的 V 均表示**以当前点为根的子树最多可用的容量**， now 均表示当前点的编号且邻接表均使用链式前向星实现。

$\mathrm{O}(nm^2)$ 方法

我们定义泛化物品为**没有固定的费用（体积）和价值，价值与费用（体积）成函数关系的物品。**泛化物品可以用一个一维数组 f 来描述，其中 f_i 意义为**在体积为 i 时泛化物品 f 的价值。**

显然将两个泛化物品 f_1, f_2 在价值尽可能大的意义下合并为一个泛化物品 f 的操作也就是

$$f_i = \max\{f_{1,j}, f_{2,i-j}\} \quad 0 \le j \le i \le m$$

其时间复杂度为 $\mathrm{O}(m^2)$ 。

显然对于这个问题我们可以将每个子树看作泛化物品，不难发现一棵子树的泛化物品就是这颗子树所有的以它的儿子为根的子树和它的根节点合并的结果。

于是我们令 $f_{i,k}$ 表示**以 i 为根的子树里，选出体积不超过 k 的物品组合的最大价值**，也就是**以 i 为根的子树成为的泛化物品。**由 $f_{i,j}$ 定义， f_i 初值即为它的根节点未与它子树合并时的权值，也就是 $f_{i,j} = w_i \quad j \ge v_i$ 。

状态转移方程为：

$$f_{i,j} = \max_{v \in \text{son of i}} \{f_{i,j}, f_{i,j-k-v_i} + f_{v,k} + w_i\} \quad j \in [v_i, m], k \in [0, j - v_i]$$

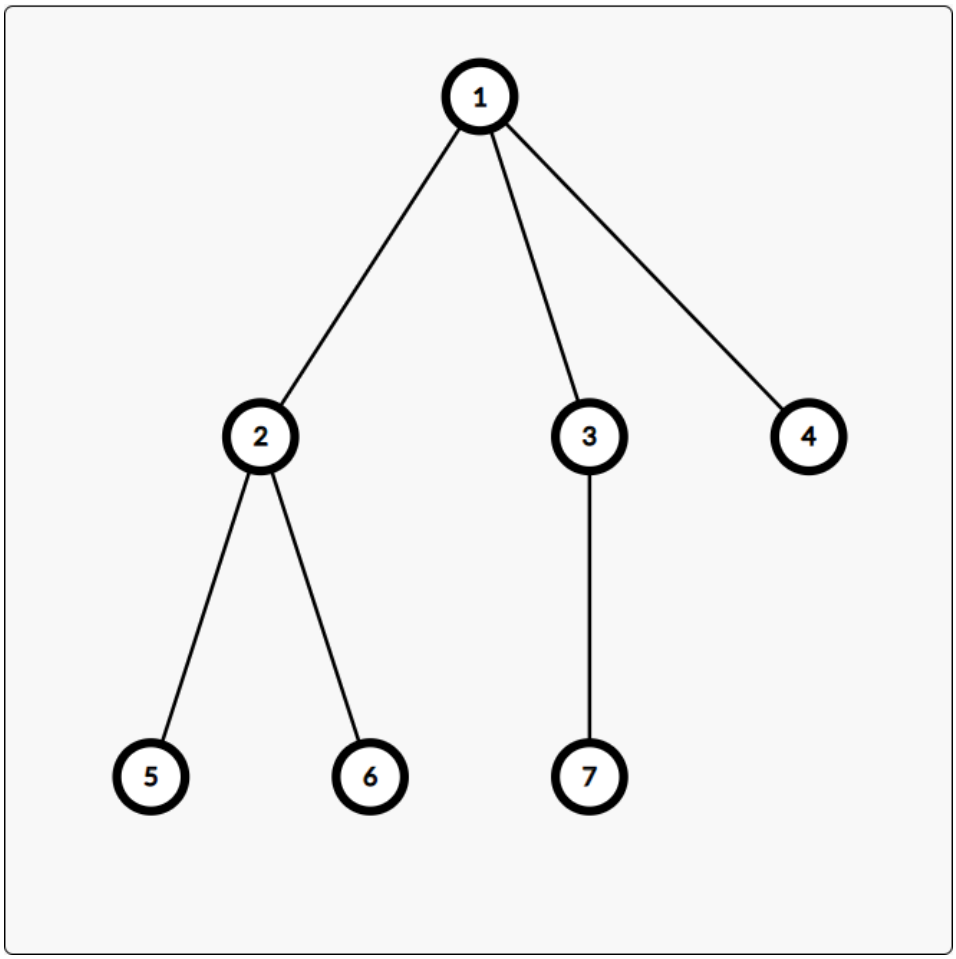
核心代码：

```
for (int i = Head[now]; i; i = G[i].Next)
{
    if (G[i].v == father)
        continue;
    dfs(G[i].v, now, V - v[now]);
    for (int j = m; j >= v[now]; --j)
        for (int k = 0; k <= j - v[now]; ++k)
            f[now][j] = max(f[now][j], f[now][j - k - v[now]] + f[G[i].v][k] +
}
```

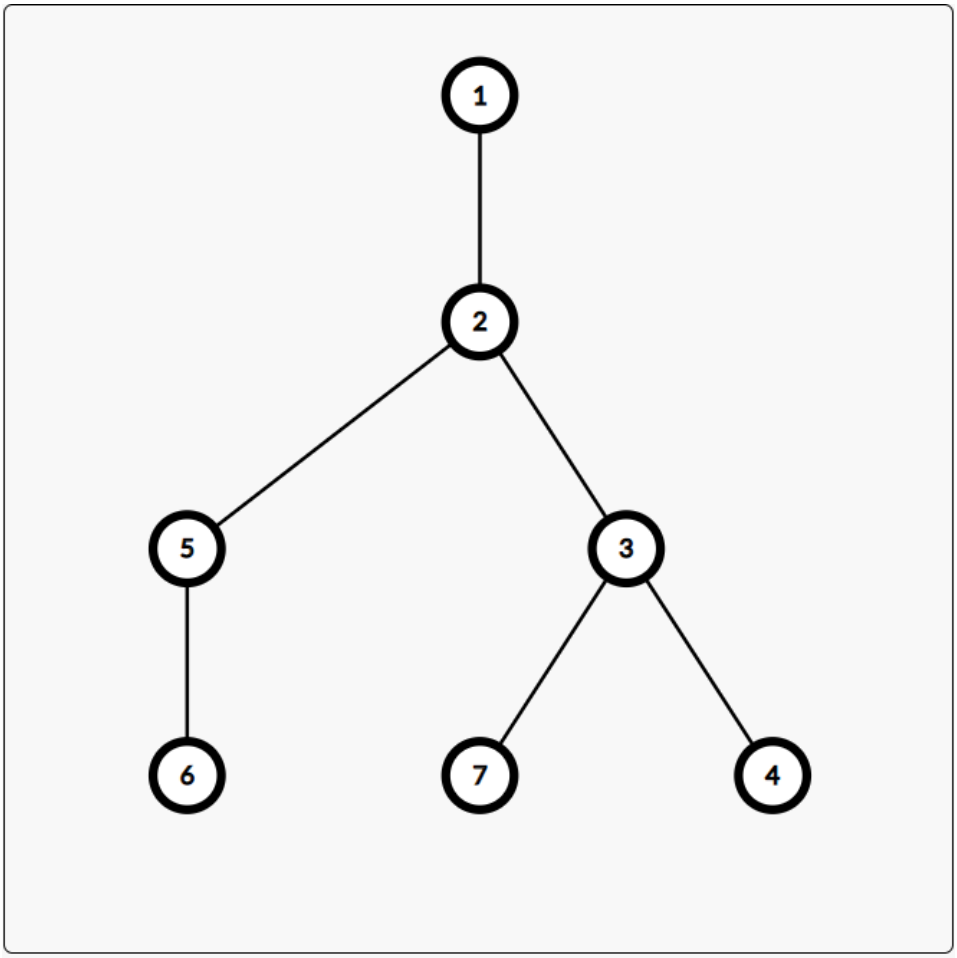
```
for (int i = Head[now]; i; i = G[i].Next) { if (G[i].v == father) continue;
dfs(G[i].v, now, V - v[now]); for (int j = m; j >= v[now]; --j) for (int k = 0;
k <= j - v[now]; ++k) f[now][j] = max(f[now][j], f[now][j - k - v[now]] +
f[G[i].v][k] + w[now]); }
```

技巧：多叉转二叉

如果我们新建一棵二叉树，左儿子对应给定树上对应节点右侧第一个兄弟节点，让这棵树上每一个节点对应给定树上的节点，右儿子对应给定树上对应节点的第一个子节点，这样我们就将多叉树转为了二叉树。



原树



二叉树

考虑二叉树上的动态规划，此时只需枚举在儿子和兄弟间的空间分配即可（注意不选儿子时无需需要讨论当前点本身选不选）：

$$f_{i,j}=\max\left(\max_{0\le k\le j-v_i}\left\{f_{son_l,k}+f_{son_r,j-k-v_i}+w_i\right\},f_{son_l,j}\right)$$

核心代码：

```
dfs(l_son[x], V);                       //l_son为兄弟节点
f[x][V] = max(f[x][V], f[l_son[x]][V]); //只有儿子选的情况
for (int i = v[x]; i <= V; ++i)         //枚举空间分配
{
    dfs(r_son[x], V - i); //r_son为子节点
    dfs(l_son[x], i - v[x]);
    f[x][V] = max(f[x][V], f[l_son[x]][i] + f[r_son[x]][V - i] + w[x]);
}
```

优化一[1]

注意到对于节点 i 的每一个儿子，我们都有选或者不选两种操作。

对于节点 i 我们可以指定一个遍历顺序，按照这个顺序遍历其所有儿子。若对于 i 的某个儿子 j，若已经处理完毕了在 j 前所有 i 的儿子和 j 的所有儿子（因为是深搜），即已知 $f_{i,j,k}$ 表示**在 j 之前（不包括 j）的所有的 i 的儿子中，以这些儿子为根的所有子树构成的泛化物品在消耗体积为 k 时的最大价值**。已知 $f'_{i,j,k}$ 表示**在 j 之前（包括j）的所有的 i 的儿子中，以这些儿子为根的所有子树构成的泛化物品在消耗体积为 k 且必选 j 时的最大价值。**

设 j 之前被遍历到的上一个 i 的儿子为 j'，则：

$$f_{i,j,k} = \max\{f_{i,j',k}, f'_{i,j,k}\}$$

注意到由于已经处理完 j 的所有儿子，故 f' 已知。

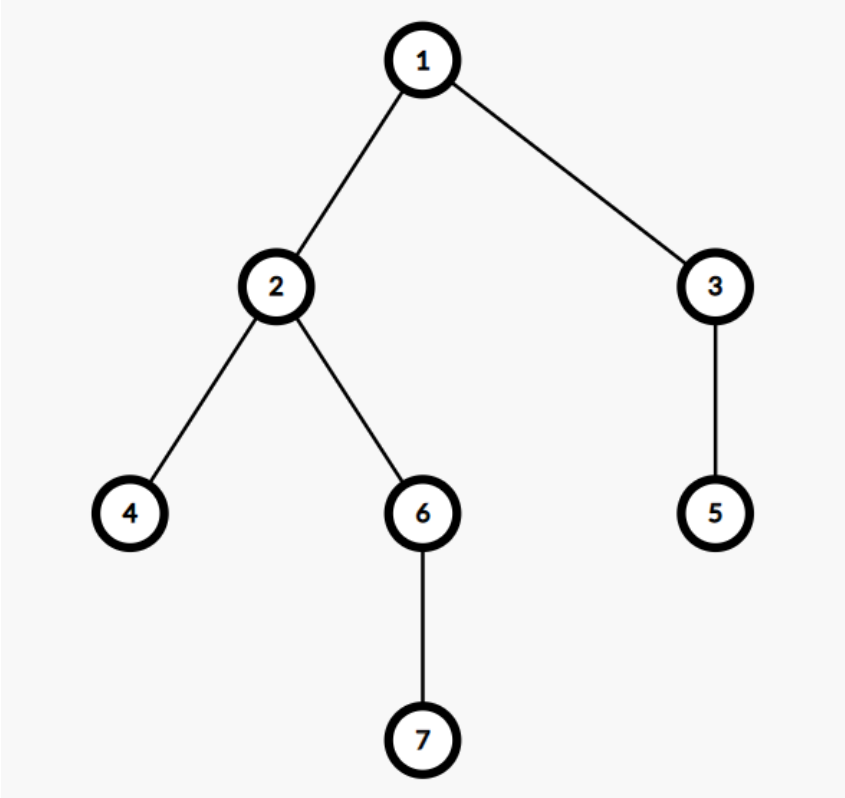
显然滚动数组可以压掉一维。

核心代码：

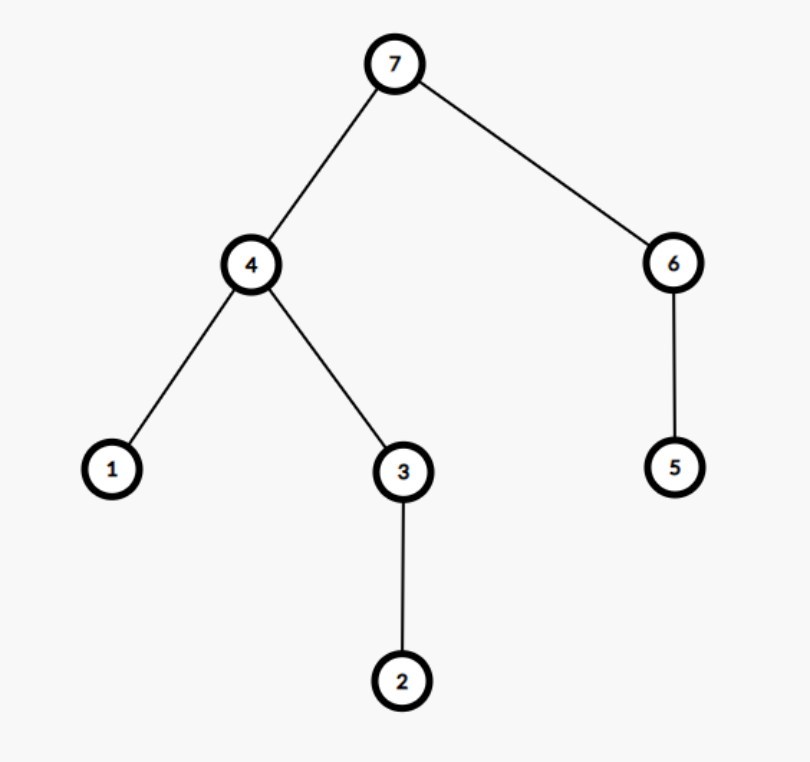
```
for (int i = 0; i <= V; ++i) //赋初值
    f[now][i] += w[now];      //注意这里是+=，不能直接赋值
for (int i = Head[now]; i; i = G[i].Next) //遍历i的儿子
{
    if (G[i].v == father)
        continue;
    for (int j = 0; j <= V; ++j) //f[G[i].v]实际上是f'[i][j]
        f[G[i].v][j] = f[now][j];
    dfs(G[i].v, V - v[G[i].v]);           //在选子树时当前点必选
    for (int j = v[G[i].v]; j <= V; ++j) //转移
        f[now][j] = max(f[now][j], f[G[i].v][j - v[G[i].v]]);
}
```

优化二

我们可以考虑将树按照 dfs 序重新编号，例如下图：



编号前



编号后

由于 dfs 序中每一个节点的子节点编号都必定比当前节点小，每一个节点的父节点编号都必定比当前节点大。所以如果将重新编号后的树上节点按编号**从小到大依次**加入一颗新树，那么显然每一个节点加入时都必然为**当前状况下所在子树的根节点**。

所以 $f_{i,j}$ 表示**将第 i 个节点加入新树时，背包容量为 j 时能取得的最大价值。**

如果那么节点 i 被取到，节点 i 的子节点才能被取到，所以说问题就转为求**对于前 $i-1$ 个点，当前背包容量为 j时能取到的最大价值。**

如果节点 i 不被取到，那么节点 i 的子节点也都不会被取到，而因为 dfs 序中一颗子树上的点编号按照 dfs 序遍历**一定是一段连续的数列**，以节点 i 为根的子树中所有节点的编号一定是从 $i-size_i+1$ 到 i，所以对于将前 $i-1$ 个点中**不在**以 i 为根的子树内的点加入新树可以获得的**最大价值**即为前 $i-size_i$ 个点在**背包容量不变**的情况下加入新树可以获得的**最大价值**，也就是 $f_{i-size_i,j}$ 。

状态转移方程

$$f_{i,j}=\max\{f_{i-1,j-v_i}+w_i,f_{i-size_i,j}\}$$

核心代码：

```
for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= V; ++j)
    {
        f[i][j] = f[i - Size[To[i]]][j]; //To[i]表示新树上编号为i的点对应的原树上的点
        if (j >= v[To[i]])
            f[i][j] = max(f[i][j], f[i - 1][j - v[To[i]]] + w[To[i]]);
    }
```

```
for (int i = 1; i <= n; ++i) for (int j = 1; j <= V; ++j) { f[i][j] = f[i - Size[To[i]]][j]; //To[i]表示新树上编号为i的点对应的原树上的点的编号 if (j >= v[To[i]]) f[i][j] = max(f[i][j], f[i - 1][j - v[To[i]]] + w[To[i]]); }
```

优化三

对于原本的 $\mathrm{O}(nm^2)$ 做法，我们可以考虑优化循环的上下界。

令 $\mathbf{size}(x)$ 表示**以 x 为根的子树中所有元素所占空间之和**。很明显当在以节点 i 为根的子树上进行选择时，选择物品的最大容量不会超过 $\mathbf{size}(i)$ 。在枚举以子节点 j 为根的子树全部能选的容量时，显然在子树中选择物品的最大容量不会超过 $\mathbf{size}(j)$ 。

可以证明此时时间复杂度为均摊 $\mathrm{O}(nm)$ 。倘若运用前文所述多叉转二叉技巧则复杂度的一个证明立即由不等式

$$\begin{aligned}&\mathbf{size}(x)^2\\\geq&(\mathbf{size}(\mathbf{son}_l(x))+\mathbf{size}(\mathbf{son}_l(x))+1)^2\\\geq&\mathbf{size}^2(\mathbf{son}_l(x))+\mathbf{size}^2(\mathbf{son}_l(x))+\mathbf{size}(\mathbf{son}_l(x))\,\mathbf{size}(\mathbf{son}_r(x))\end{aligned}$$

立即给出。

核心代码：

```
Size[now] = v[now]; //初始化
for (int i = Head[now]; i; i = G[i].Next)
{
    if (G[i].v == father)
        continue;
    dfs(G[i].v, now, V - v[now]);
    Size[now] += Size[G[i].v];
    for (int j = v[now]; j <= min(V, Size[now]); ++j)
        for (int k = 0; k <= min(j - v[now], Size[G[i].v]); ++k)
            f[now][j] = max(f[now][j], f[now][j - v[now] - k] + f[G[i].v][k] +
}
```

Size[now] = v[now]; //初始化 for (int i = Head[now]; i; i = G[i].Next) { if (G[i].v == father) continue; dfs(G[i].v, now, V - v[now]); Size[now] += Size[G[i].v]; for (int j = v[now]; j <= min(V, Size[now]); ++j) for (int k = 0; k <= min(j - v[now], Size[G[i].v]); ++k) f[now][j] = max(f[now][j], f[now][j - v[now] - k] + f[G[i].v][k] + w[now]); }

参考

1. ^ https://max.book118.com/html/2018/0220/154009827.shtm

编辑于 2022-09-05 22:50

OI（信息学奥林匹克） 动态规划算法 背包问题

17 条评论 默认 最新

知乎用户2cTKoS

2020-11-29 回复 4

AH20

优化三的代码部分，j的循环方向是不是错了

11-15 回复 1

群里人过来蹭一蹭

2020-11-29 回复 1

星夜 作者

UPD 2021.9.23: 修改了大量错误并优化了文本

2021-09-23 · 热评 回复 赞

wwlw

请问优化二的dfs序的图是不是有问题啊

2021-09-09 回复 赞

星夜 作者

已改正。

2021-09-10 回复 赞

星夜 作者

UPD 2021.2.27: 修改了一些描述性错误

2021-02-27 · 热评 回复 赞

星夜 作者

upd 12.10：已更新优化四以及改正此前一些小错误。

2020-12-10 · 热评 回复 赞

知乎用户2cTKoS

加油!

2020-12-10 回复 1

铁肩二世

蹭一蹭

2020-12-08 回复 赞

Expeditioner

多叉转二叉那里图左右儿子颠倒了。

10-10 回复 赞

星夜 作者

UPD 2022.9.5: 改正了一些严重错误并增加了少许证明

09-05 · 热评 回复 赞

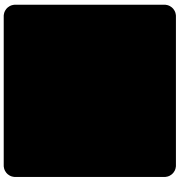
亲切的问候

优化2，请问这个和没有优化的版本有区别吗，都是对于每个节点的空间从1-V枚举，当然越到树的下一层，可用的空间越少。没有优化的版本也是dfs(G[i].v, now, V - v[now]),这里两者没有区别。而对于dp[i][j],两个版本不都是要再进行一次O(m)的循环更新吗？优化2这个版本，dp[i][j]，j要枚举，更新dp[i][j]的值的时候也要枚举。为什么它是O(nm)的呢

08-30 回复 赞

星夜 作者
是的，优化二是假的😂
09-05 回复 赞

tgs9311
👍👍👍



07-21 回复 赞

旭旭宝宝
合并两个泛化物品那里是不是应该改成f[i] = max{ f1[j] + f2[i - j] }?
05-27 回复 赞

星夜 作者
这和我写的也没啥区别吧……
05-27 回复 赞