树形与换根 DP

自底向上子树汇总、树形背包与二次扫描换根

HGNU-ACM 实验室算法组

配套：三大核心模型（基础双状态 + 树形分组背包 + 二次扫描换根）与 3 道真题全拆解





本章目录

树形 DP 基础：自底向上汇总与双状态

树形背包：带依赖的容量分配

换根 DP：二次扫描求全树每个点的答案

承接第 2 章《区间与环形 DP》的“小块合并成大块”思想，本章把合并舞台搬到树上：先学自底向上汇总，再学树形背包，最后学换根 DP。

本章目录（续）

三大真题精讲：从题意到代码

常见错误、检查清单与练习安排

第 4 节逐页精讲《周年纪念晚会》《二叉苹果树》《旅游规划》三道真题，第 5 节总结避坑清单与第 4 章预告。

承上启下：从“区间合并”搬到“树上合并”

- 第 2 章《区间与环形 DP》中，我们先把小区间 $[i,k]$、$[k+1,j]$ 算好，再拼成大区间 $[i,j]$。
- 树也一样：把每棵子树（某节点和它底下所有后代）看成一个小块，先算好所有孩子的子树，再在父节点处拼起来。



本页要点

线性和区间有左右顺序；树没有左右下标，唯一的天然顺序就是从叶子到树根！

先想一个小例子：4 个人的公司发奖金

生活场景

小公司只有 4 个人：1 号是老板，手下有 2 号和 3 号两位主管；2 号手下还有 4 号员工。每人来开会能带来欢乐度 $[2,5,4,3]$。规定直属上下级不能同时参加，总欢乐度最大是多少？

- 纸上试几种选法：选老板 1，则 $2,3$ 都不能来，再选 4，得 $2+3=5$。
- 不选老板 1，则 3 可以来（得 4）；2 这边若选 2 就得 5，不选 2 选 4 只得 3。
- 合并起来：不选 1、选 2 和 3，总欢乐度 $=5+4=9$，比选老板大得多！

本页要点

写代码前先手算小例子：我们发现“某人来不来”只跟他自己和他的直属下属有关。

为什么不能暴力枚举所有方案？

- 如果有 n 个人，每个人只有“来”或“不来”2 种选择，暴力穷举一共有 2^n 种名单。
- 检查每种名单是否有上下级冲突还要 $O(n)$ 时间，总操作次数大约 $n \cdot 2^n$。

节点数 n	暴力操作数 $n \cdot 2^n$	1 秒（约 10^8 次）能否跑完
10	$\approx 1.0 \times 10^4$	轻松通过
20	$\approx 2.1 \times 10^7$	勉强通过
6000	$> 10^{1800}$	宇宙毁灭也算不完（必 **TLE**）

本页要点

当 $n = 6000$ 时，指数级暴力必超时；把重复的子树决定记在数组里，就能降到 $O(n)$！

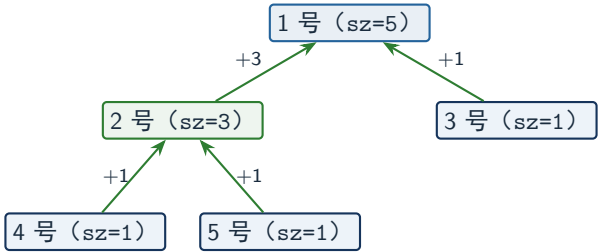
新术语大白话与建树准备

- 子树：把节点 u 和它下面的所有子孙圈起来，就是一棵以 u 为顶的小树。
- 后序遍历：先递归算完所有儿子 v 的答案，回溯到父亲 u 时再把儿子结果加起来。
- 双状态 `dp[u][0/1]`：对点 u 分别记两个答案——`dp[u][0]` 表示 u 不选时的子树最优解，`dp[u][1]` 表示 u 必选时的子树最优解。
- 两种存树方式：
  1. 有向上司关系：用 `ch[k].push_back(l)` 存孩子，找 `!has_fa[r]` 作唯一树根。
  2. 无向边：两边都连 `adj[u]`，DFS 时传参 `fa` 并写 `if (v == fa) continue;` 防止回头死循环！

本页要点

写树形 DP 前先看清：边是有向还是无向？树根是固定 1 号还是需要自己找？

热身图解：自底向上汇总子树大小 sz[u]

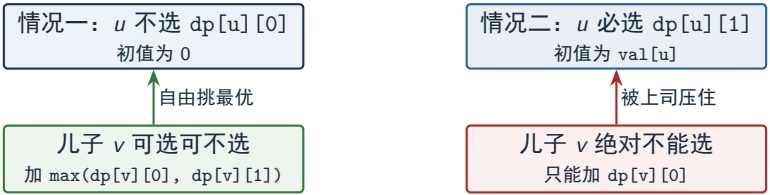


- 每个节点刚进来时，先把自己算成 1 个点：sz[u] = 1。
- 儿子 v 递归回来后，把儿子的汇总值直接加进父亲：sz[u] += sz[v]。

本页要点

所有树形 DP 的骨架都和求 sz[u] 一模一样：自己初始化 → 递归儿子 → 把儿子拼回自己！

状态转移图解：dp[u][0] 与 dp[u][1] 怎么拼？



- 各个儿子 v 之间只通过父亲 u 相连，彼此独立，所以可以直接累加每个儿子子树的贡献。
- 如果点权是负数且允许不选任何点，max(dp[root][0], dp[root][1]) 至少会保底拿到 0。

本页要点

把每个节点看作一个“只做局部决定”的开关：选 u 就锁死全部儿子为 0，不选 u 就放开所有儿子。

手算跟踪表：5 个节点的后序汇总全过程

设点权为 $w=[0,2,5,4,3,1]$，树结构同前图：1 连 $2,3$；2 连 $4,5$。

后序访问 u	刚进 u 时初值	合并的儿子与汇报值	更新后 sz[u]	更新后 dp[u][0/1]
4（叶）	sz=1，[0，3]	无儿子	1	[0，3]
5（叶）	sz=1，[0，1]	无儿子	1	[0，1]
2	sz=1，[0，5]	儿子 $4:[0,3]$，儿子 $5:[0,1]$	$1+1+1=3$	[3+1=4，5+0+0=5]
3（叶）	sz=1，[0，4]	无儿子	1	[0，4]
1（根）	sz=1，[0，2]	儿子 $2:[4,5]$，儿子 $3:[0,4]$	$1+3+1=5$	[5+4=9，2+4+0=6]
终止输出	整树算完	取 $\max(\texttt{dp[1][0]},\texttt{dp[1][1]})$	5	$\max(9,6)=9$

本页要点

最后一行终止步取根节点的两个状态最大值，得到 9（对应选 2 和 3）。

为什么是对的？树形 DP 的子树不变量

用三个问题检查“这样递归合并为什么一定不漏也不错”：

论证阶段	在树上成立的理由
初始化	叶子 u 没有儿子，`dp[u][0]=0`，`dp[u][1]=val[u]` 就是该单点子树唯一正确的两种最优解
保持	处理完前 k 个儿子时，这些儿子之间互不相邻；乘上第 $k+1$ 个儿子最优解，依然满足整棵子树合法
终止	根节点 *root* 把所有直属儿子合并完毕后，`dp[root][0]` 和 `dp[root][1]` 覆盖整棵树全部合法方案

- 每个节点只被访问一次，每条树边只被走过一次，因此总时间复杂度是严格的 $O(n)$。

本页要点

只要子问题之间只通过当前父节点发生约束，子树最优解就能无后效性地直接相加。

模板代码：基础子树汇总与双状态 dp[u][0/1]

```
void dfs(int u, int fa) {
    sz[u] = 1;                              // 自己先占 1 个节点
    dp[u][0] = 0;                           // u 不选，当前子树基础收益为 0
    dp[u][1] = val[u];                      // u 必选，先拿走自己的点权
    for (int v : adj[u]) {
        if (v == fa) continue;              // 无向树必须拦截回头边！
        dfs(v, u);                          // 先递归把儿子 v 的整棵子树算完
        sz[u] += sz[v];                     // 汇总子树大小
        dp[u][0] += max(dp[v][0], dp[v][1]); // u 不选：儿子选不选挑大的
        dp[u][1] += dp[v][0];               // u 选了：儿子只能不选
    }
}
```

为什么初值这样写

dp[u][0]=0 与 dp[u][1]=val[u] 对应叶子基准；若放进循环里就成了每个儿子都加一次 val[u]！

上机验证与反事实追问

在本地补上输入建树与 `dfs(1, 0)`，用下面 3 组数据实测对比：

测试类型	输入（n、点权、树边）	预期输出 `max(dp[1][0], dp[1][1])`
普通 5 点树	$n=5,\ w=[2,5,4,3,1]$，边 1-2, 1-3, 2-4, 2-5	9（选 2, 3）
最小 1 点树	$n=1,\ w=[-10]$，无边	0（选 `dp[1][0]=0`）
全负数 3 点	$n=3,\ w=[-5,-2,-8]$，边 1-2, 2-3	0（谁都不选最优）

反事实追问（写代码前先想清）

1. 如果删掉 `if (v == fa) continue;`，运行后会出什么错？（沿着无向边反复在 $1 \leftrightarrow 2$ 间绕圈，最终栈溢出 **RE**）
2. 如果把 `dp[u][1] += dp[v][0]` 错写成 `dp[u][1] += max(dp[v][0], dp[v][1])`，上面第 1 组会多算出什么错解？（把 1 和 2 同时选了，输出 $2+5+4=11$ **WA**）

基础树形 DP 常见易错对照

易错点	错误写法	正确做法与评测后果
写死 1 号为根	有向上司关系直接 `dfs(1)`	大老板未必是 1 号！应找 `!has_fa[r]` 作根，否则漏树枝 **WA**
无向树忘判父节点	遍历 `adj[u]` 不检查 `v == fa`	顺着无向边走回父亲，引发无限递归栈溢出 **RE**
`val[u]` 写进循环	循环里每次加 `val[u]`	自己被算重多次，应放在遍历儿子之前初始化 **WA**
长链栈溢出	$n = 2 \times 10^5$ 单链直接深搜	Windows 默认栈小易爆栈 **RE**，大数据可改 BFS 序逆序扫

本页要点

树形 DP 第一原则：认准唯一根、不走回头边、只把节点自身初始化一次！

生活引入：树形依赖下的“选 Q 条边”

生活场景

一棵苹果树根在 1 号位置：1 连着 2（长了 5 个苹果）和 3（长了 10 个苹果），2 下面又连着 4（长了 20 个苹果）。如果只允许保留 $Q = 2$ 根树枝，最多留住多少苹果？

- 注意规则：想留 2-4 这根树枝，必须先留住上面的 1-2，否则剪掉 1-2 时整枝就掉了！
- 方案 A：留 1-2 和 1-3，共 $5 + 10 = 15$ 个苹果。
- 方案 B：留 1-2 和 2-4，共 $5 + 20 = 25$ 个苹果（最优！）。

本页要点

不能直接拿最大的两根边 $10 + 20 = 30$，因为 2-4 的父边 1-2 没被留住就悬空了！

为什么不能直接套普通 0-1 背包？

- 普通 0-1 背包里，每件物品互相独立，可以随便挑；但树形问题里选儿子前必须先选父亲。
- 如果想暴力穷举所有“从根出发连通、恰好包含 Q 条边”的子树，在 $N = 100, Q = 50$ 时合法子图数量是指数级（远超 10^{15} 种），必 **TLE**！
- 换个角度：对于节点 u 和它的某个儿子 v，我们到底要决定什么？
  1. 不管子树 v 具体怎么挑，我们只关心总共分给子树 v 多少容量（比如 $0, 1, 2, \ldots$ 条边）。
  2. 在同一次合并里，只能挑一个分配额 k——这正是第 1 章学过的分组背包！

本页要点

把“一棵儿子子树”看作“一组互斥的物品”：给它 0 格、1 格、... 容量，每组最多选 1 个！

状态定义与大白话解释

状态怎么设？

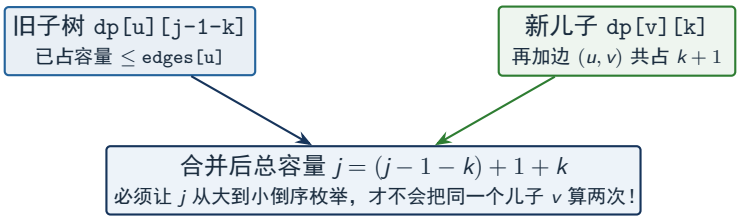
设 `dp[u][j]` 表示：在以 u 为根的子树中，连同必须保留的节点 u 一起，总共再保留 j 条子树树枝（或选 j 个点）时能得到的最大收益。

- 依赖关系怎么保证？只有先算过连接边 (u, v)（占用 1 条边），才允许把剩余 k 条边的份额分给 v 的子树 `dp[v][k]`。
- 已经处理完的儿子怎么拼？把之前所有儿子合起来的结果存在当前 `dp[u][...]` 里，再来一个新儿子 v 就做一次卷积更新：
  $\texttt{dp[u][j]} = \max\left(\texttt{dp[u][j]},\ \texttt{dp[u][j - 1 - k]} + \texttt{dp[v][k]} + w(u, v)\right)$
- 式子里的 1 是边 (u, v) 自己占的名额，k 是继续分给 v 内部的名额，剩下的 $j - 1 - k$ 留给 u 的其他旧儿子！

本页要点

旧儿子们的成果都已装进 `dp[u]`，新儿子 v 来时只需做一次“旧成果 + 新儿子”的容量划分。

独占图解：树形分组背包的容量拼接



- 和 0-1 背包一维优化完全同理：如果 j 从小到大更新，新算出的 `dp[u][j-1-k]` 会被后面的 j 再次使用，等于把同一棵子树 v 选了两遍！

本页要点

每来一个儿子 v，总容量 j 必须倒序枚举；而分给 v 的容量 k 在内层正序枚举即可。

手算跟踪表：4 个节点、保留 $Q = 2$ 条边

树结构：1-2(5)、1-3(10)、2-4(20)，初值全 0。从根 1 出发：

合并步骤	合并前 dp[u]	枚举 j（倒序）与 k	更新算式	合并后 dp[u]
处理 2 连 4(20)	$[0,0,0]$	$j=1,k=0$	$0+0+20=20$	$[0,20,0]$
1 合并儿子 2(5)	$[0,0,0]$	$j=2,k=1$	$0+20+5=25$	$[0,0,25]$
同上（续）	$[0,0,25]$	$j=1,k=0$	$0+0+5=5$	$[0,5,25]$
1 合并儿子 3(10)	$[0,5,25]$	$j=2,k=0$	$\max(25,5+0+10)=25$	$[0,5,25]$
同上（续）	$[0,5,25]$	$j=1,k=0$	$\max(5,0+0+10)=10$	$[0,10,25]$
终止输出	根 1 全合完	直接取 dp[1][2]	最终答案	25

本页要点

倒序时先算 $j=2$ 再算 $j=1$，确保儿子 3 在算 $j=2$ 时用的是儿子 2 留下的旧值 dp[1][1]=5！

正确性与复杂度：为什么上下界优化后不是 $O(n^3)$？

阶段 / 视角	关键结论
初始化	未合并任何儿子前，`dp[u][0]=0`（保留 0 条边），其余容量若求“至多”设为 0
保持	因为 j 倒序，`dp[u][j-1-k]` 严格来自“旧儿子们”，不会混入当前儿子 v
终止	根节点合并完全部儿子后，`dp[1][Q]` 即恰好选至多 Q 条连通边的最大值

- 为什么是 $O(n^2)$？每次只让 j 枚举到 `min(Q, edges[u] + edges[v] + 1)`，且 k 只枚举到 `min(edges[v], j-1)`。
- 换个眼光看：循环内部每执行一次，等价于选了旧集合里的 1 个点和儿子 v 里的 1 个点配对——树上任意两点只会在它们的最近公共祖先处相遇一次，所以总配对数不超过 $\binom{n}{2} = O(n^2)$！

本页要点

加上子树当前已合并大小 `edges[u]` 上界限制后，树形背包总复杂度从 $O(nQ^2)$ 降到 $O(n^2)$！

模板代码：上下界优化的树形分组背包

```
void dfs(int u, int fa) {
    for (auto [v, w] : adj[u]) if (v != fa) {  // 拦截父节点并先算儿子
        dfs(v, u);
        int lim = min(Q, edges[u] + edges[v] + 1);
        for (int j = lim; j >= 1; --j) {         // 总容量 j 必须倒序!
            int upk = min(edges[v], j - 1);
            for (int k = 0; k <= upk; ++k)       // 留 1 条给边 (u,v), k 条给子树 v
                dp[u][j] = max(dp[u][j], dp[u][j - 1 - k] + dp[v][k] + w);
        }
        edges[u] += edges[v] + 1;                // 合并完当前儿子后再更新边数!
    }
}
```

本页要点

`edges[u]` 必须写在循环末尾，否则找不到“旧树到底有几条边”。

上机验证与反事实追问

补全读入并用 `dfs(1, 0)` 后测试下面 3 组数据：

测试类型	输入（N, Q 及带权树边）	预期输出 `dp[1][Q]`
普通 4 点树	$N=4, Q=2$，边 1-2(5), 1-3(10), 2-4(20)	25（选 1-2 与 2-4）
最小边界 $Q=1$	$N=3, Q=1$，边 1-2(7), 1-3(9)	9（只能留 1-3）
单链极限	$N=4, Q=2$，链 1-2(3), 2-3(8), 3-4(100)	11（无法跨过 2-3 拿 100）

反事实追问

1. 如果把 `for (int j = lim; j >= 1; --j)` 误写为正序 `for (int j = 1; j <= lim; ++j)`，第 2 组或 1-2(5), $Q=2$ 时会怎样？（同一根 1-2 会被重复加上两次，算出不存在的 10 **WA**！）
2. 如果在单链测试中拿掉 `dp[u][j - 1 - k]` 里的 `-1`，程序会错成什么？（误以为不留父边就能拿儿子，直接错选出 100！）

树形背包的泛化：选边 vs 选点、“至多” vs “恰好”

题目变形	状态与转移微调	初始化关键区别
树上选点（如选课）	节点 u 必占 1 个名额：`dp[u][1]=val[u]`，j 从 lim 倒序到 1（至少留 1 给 u 自己）	若允许森林（多门课无先修），设虚拟总根 0 号，容限放大到 $M+1$
“恰好”选 Q 个	状态定义改为“恰好选 j 个”，初始不可达全设极小值 `-INF`	只有 `dp[u][0]=0`（或 `dp[u][1]=val[u]`）合法，防止空态乱转移
“至多”选 Q 个	所有可达边权非负时，全部 `dp[u][j]` 初值设为 0 即可	少选的空位价值为 0，最终直接读 `dp[root][Q]`

本页要点

树形背包只有两个开关要核对：是按选点还是选边扣 1？初始化用 0 还是 `-INF`？

生活引入：如果“每个路口”都想当一次总站呢？

问题升级

前两节的树都有固定树根。现在城市有 4 个路口连成树：1-2、1-3、2-4。市长想问：如果把总站分别建在 $1,2,3,4$ 号路口，总站到其他所有路口的距离之和分别是多少？

- 建在 1 号：到 $2,3$ 各 1 步，到 4 是 2 步，总和 $=1+1+2=4$。
- 建在 2 号：到 $1,4$ 各 1 步，到 3 是 2 步，总和 $=1+1+2=4$。
- 建在 3 号：到 1 为 1 步、到 2 为 2 步、到 4 为 3 步，总和 $=1+2+3=6$。

本页要点

现在要的不是“一个根的答案”，而是全部 n 个点轮流当根的 n 个答案！

为什么不能每个点都重跑一遍树形 DP？

- 最直观的暴力：枚举根 $r = 1, 2, \ldots, n$，每次以 r 为根做一遍 $O(n)$ 树形 DP。
- 当 $n = 2 \times 10^5$ 时，总时间复杂度是 $O(n^2) = (2 \times 10^5)^2 = 4 \times 10^{10}$ 次操作！
- 在 1 秒约 10^8 次运算的评测机上，暴力换根至少要跑 400 秒，必然 **TLE**。
- 观察相邻两个点 u 和 v 当根时的差别：整棵树其实只改了一根边 (u, v) 的方向，绝大多数子树的内部答案根本没变！

本页要点

换根 DP 的核心动机：不重复算同一片子树，只用 $O(1)$ 时间把根从父亲 u 挪到儿子 v！

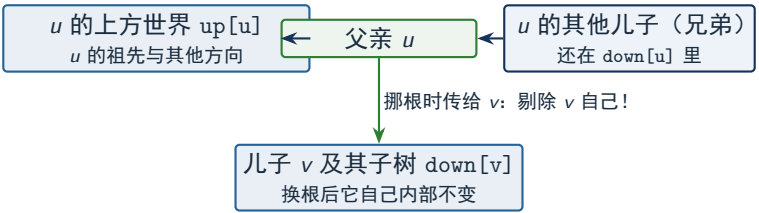
核心想法：二次扫描（两遍 DFS）大白话

- 二次扫描法就是把树走两遍，总计只需要 $O(n)$ 时间：
  1. 第一遍（自底向上）：随便定 1 号（或 0 号）当临时根，用普通树形 DP 算出每个点往自己子树下面看的贡献（记为 down 信息，例如子树大小 `sz[u]`、向下最长链 `d1[u]`）。
  2. 第二遍（自顶向下）：从临时根出发，把根从父亲 u 挪给儿子 v 时，利用父亲 u 的全树信息，剔除 v 自己那块子树，就得到了 v 往父亲外面看的贡献（记为 `up[v]`）。
- 于是对任何点 v，以它为根的全局答案 $=$ 内部（`down[v]`） $+$ 外部（`up[v]`）！

本页要点

一句话口诀：第一遍“先子后父”算子树里面，第二遍“先父后子”把外面的世界告诉儿子！

独占图解：二次扫描时，根从 u 挪给 v 发生了什么？



- 当新根挪到 v 时，整棵树被切成两半：v 的原子树内部（蓝色 down[v]），以及“除 v 外的所有点”（包含 up[u] 和 u 的其他兄弟）。

本页要点

只要从父亲 u 的全树视野里扣掉 v 自己，剩下的全部节点刚好拼成 up[v]！

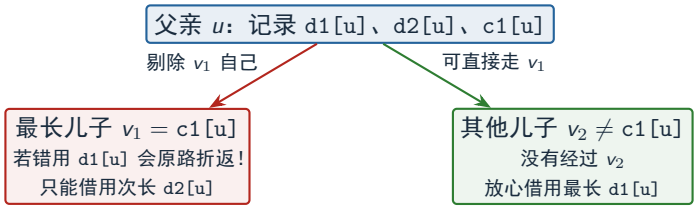
第一类换根模型：距离和（可加减直接扣）

- 回到前面“找总站”问题：设 ans[u] 表示以 u 为根时全树到 u 的距离和，第一遍先算出 sz[u] 和 ans[root]。
- 现在根从父亲 u 走 1 步挪到儿子 v：
  1. v 子树里的 sz[v] 个点，离新根变近了 1 步，总距离减少 sz[v]。
  2. 子树外面的 $n - \mathrm{sz}[v]$ 个点，离新根变远了 1 步，总距离增加 $n - \mathrm{sz}[v]$。
- 因此换根公式一行搞定：$\mathrm{ans}[v] = \mathrm{ans}[u] - \mathrm{sz}[v] + (n - \mathrm{sz}[v])$。

本页要点

遇到“求和、计数”类换根题，可以直接从父亲总和里加减贡献，实现 $O(1)$ 挪根！

独占图解：第二类换根——最值不能直接减，必须留次长支！



- max 操作不能像加法那样做减法！如果 v 正好在父亲向下的最长支 c1[u] 上，换根给 v 时必须改走次长向下支 d2[u]，再比较父亲向上的 up[u]：

$$\texttt{up[v]} = \max\Big(\texttt{up[u]},\ (v == \texttt{c1[u]}\ ?\ \texttt{d2[u]} : \texttt{d1[u]})\Big) + 1$$

本页要点

这是换根 DP 最著名的坑：算最值时记住最长 d1、次长 d2、最长出处 c1 三件套！

手算跟踪表：5 节点树的两遍扫描（最值换根）

树边：0-1, 0-2, 1-3, 1-4，以 0 为临时根，求每个点向下最长 d1、次长 d2 与向上 up：

阶段与节点	经过儿子计算（或来自父亲）	d1[u]，d2[u]	c1[u]	向上最长 up[u]
第一遍：3, 4（叶）	无儿子，向下链长均为 0	0, 0	-1	待第二遍计算
第一遍：1	3 献 1，4 献 1	1, 1	3	待第二遍计算
第一遍：2（叶）	无儿子	0, 0	-1	待第二遍计算
第一遍：0（根）	1 献 $1+1=2$，2 献 1	2, 1	1	根上方无点：up[0]=0
第二遍：1（父 0）	$1 ==$ c1[0]，借用 d2[0] $=1$	1, 1	3	$\max(0,1)+1=2$
第二遍：2（父 0）	$2 \neq$ c1[0]，借用 d1[0] $=2$	0, 0	-1	$\max(0,2)+1=3$
第二遍：3, 4（叶终止）	3 用 d2[1]=1，4 用 d1[1]=1	0, 0	-1	均得 $\max(2,1)+1=3$

本页要点

算完后可核对：3 号往 1 再往 0 再往 2 距离正好是 3，与表中 up[3]=3 完全一致！

为什么这样换根既不重也不漏？

阶段	严谨保证
初始化	临时根 0 没有上方节点，up[0]=0；所有节点的 d1，d2 已在第一遍算准确
保持	当父 u 的 up[u] 已包含“u 子树外部全貌”，那么 $\max(\texttt{up[u]}, \text{不含}v\text{的兄弟支}) + 1$ 正好涵盖 v 外部全树
终止	所有点都更新完 up 后，过点 u 的三条出路仅由 d1[u]，d2[u]，up[u] 组成，绝无折返

- 注意“不同子树分支”的关键：求 d1[u]，d2[u] 时只能记录来自不同直属儿子的最长与次长，不能让同一个儿子把两条边同时占掉！

本页要点

边 (u, v) 把树一刀切两半，down[v] 管里头，up[v] 管外头，合在一起就是全树！

模板代码：两遍扫描最值换根（第一遍 + 第二遍）

```
void dfs1(int u, int fa) {                    // 第一遍：自底向上算 d1, d2, c1
    for (int v : adj[u]) if (v != fa) {
        dfs1(v, u);
        int len = d1[v] + 1;
        if (len > d1[u]) { d2[u] = d1[u]; d1[u] = len; c1[u] = v; }
        else if (len > d2[u]) d2[u] = len;
    }
}
void dfs2(int u, int fa) {                    // 第二遍：自顶向下把 up 推给儿子
    for (int v : adj[u]) if (v != fa) {
        int sib = (c1[u] == v ? d2[u] : d1[u]); // 撞上最长支就借次长支！
        up[v] = max(up[u], sib) + 1;
        dfs2(v, u);
    }
}
```

进阶稳妥写法：用 BFS 序代替深递归防爆栈

当题目给定 $n \le 2 \times 10^5$ 且树可能退化成一条长链时，Windows 默认栈空间容易在深递归时崩溃（**RE**）。用 BFS 序数组 `order[]` 可以零递归完成两遍扫描：

```
vector<int> fa(n, -1), order = {0};
for (int i = 0; i < (int)order.size(); ++i) { // 1. 正向 BFS 得到父子关系与拓扑序
    int u = order[i];
    for (int v : adj[u]) if (v != fa[u]) { fa[v] = u; order.push_back(v); }
}
// 2. 第一遍自底向上：按 order 倒着扫 i = n-1 .. 0
// 3. 第二遍自顶向下：按 order 正着扫 u : order
```

为什么这么写稳？

`order` 保证父亲永远排在儿子前面：倒着走就是“先子后父”，正着走就是“先父后子”，彻底告别爆栈！

上机验证、反事实追问与易错对照

用前面 5 点树与下面两组边界数据验证模板输出 $\{d1[u], d2[u], up[u]\}$ 中的最大值：

测试类型	输入（n 与树边）	各点最远距离 max(d1[u], up[u])
普通 5 点树	$n=5$，边 0-1, 0-2, 1-3, 1-4	$0:2,\ 1:2,\ 2:3,\ 3:3,\ 4:3$
最小 2 点树	$n=2$，边 0-1	$0:1,\ 1:1$
4 点单链	$n=4$，边 0-1, 1-2, 2-3	$0:3,\ 1:2,\ 2:2,\ 3:3$

反事实追问与易错

1. 若把 `sib = (c1[u] == v ? d2[u] : d1[u])` 错写为直接用 `d1[u]`，单链测试里 1 号传给 2 号时发生什么？（把 2 自己贡献的长度再加 1 传回给 2，变成自己绕自己一圈 **WA**！）
2. 若把两遍扫描顺序写反（先正序算 up 再逆序算 d1），由于 d1，d2 还全都是 0，所有 up[v] 全都算错！

例题总览与难度分层

点击题名跳转到精讲第一页；建议先独立手算样例再看代码：

1. 例 1 | 周年纪念晚会（一本通 1582） [基础]
   考点：有向“下属-上司”树找根 + 树上最大独立集双状态 `dp[u][0/1]`。
2. 例 2 | 二叉苹果树（一本通 1575） [提高]
   考点：无向带权树转边容量 + 树形分组背包倒序合并。
3. 例 3 | 旅游规划（一本通 1581） [提高]
   考点：非递归 BFS 序两遍换根 DP + 直径节点等价条件。

本页要点

三道例题恰好对应本章三大核心模型：基础双状态、树形背包、二次扫描换根。

例 1 | 周年纪念晚会 [基础]：先读懂要求

大白话题意

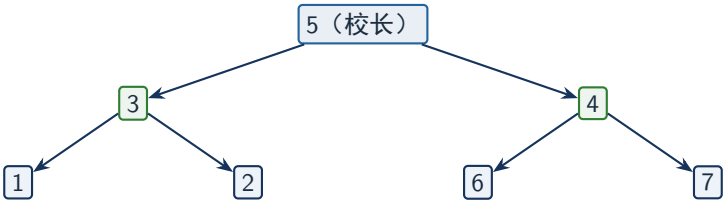
大学有 N 位职员（$1 \le N \le 6000$），每位职员有一个欢乐度 p_i（$-128 \le p_i \le 127$，可能为负数！）。大家按“直接上司”连成一棵树。如果请了某个职员，他的直接上司就不能来。求来参会人员的最大总欢乐度。

- 输入：第 1 行 N；接着 N 行为 $p_1 \dots p_N$；接着若干行 L K 表示 K 是 L 的上司，直到 0 0 结束。
- 处理：找出唯一没有上司的总校长作树根，从下往上做独立集 DP。
- 输出：一行一个整数，最大欢乐度之和。

原题样例

$N = 7$，欢乐度全为 1，边为 $(1,3),(2,3),(6,4),(7,4),(4,5),(3,5)$，以 0 0 结束 $\Longrightarrow$ 输出 5。

例 1 | 怎么想到的与树上建模



- 看到“K 是 L 的直接上司”，用 `has_fa[L]=true` 标记下属，最后没有父亲的点就是总校长 5。
- 树上任意边两端的两个人不能同时去，直接套用第 1 节树上最大独立集双状态：`dp[u][0]`（不来）与 `dp[u][1]`（来）。

本页要点

看到“有向父子树 + 上下级不能同选”，立刻用 `has_fa` 找根再跑 `dp[u][0/1]`！

例 1 | 样例手推跟踪表

按后序（先叶子再上司）追踪原题样例树，每人欢乐度均为 1：

当前处理节点 u	下属汇报情况	`dp[u][0]`（u 不来）	`dp[u][1]`（u 来）
$1, 2$（叶子）	无下属	0	1
3	下属 $1:[0,1]$，$2:[0,1]$	$\max(0,1)+\max(0,1)=2$	$1+0+0=1$
$6, 7$（叶子）	无下属	0	1
4	下属 $6:[0,1]$，$7:[0,1]$	$\max(0,1)+\max(0,1)=2$	$1+0+0=1$
5（总根）	下属 $3:[2,1]$，$4:[2,1]$	$\max(2,1)+\max(2,1)=4$	$1+2+2=5$
终止输出	全部汇总完毕	$\max(4,5)=5$	选 $5,1,2,6,7$ 共 5 人

本页要点

手推结果精确等于样例输出 5：校长 5 和最底层的 4 名普通员工一起参加！

例 1 | 核心步骤与相对模板的变化

按顺序计算	纸上得到	核对问题
读入欢乐度与上下级	`ch[k]` 存下属，`has_fa[l]=true`	是否注意是 K 当上司、L 当下属？
找总根 `root`	扫 $1 \ldots N$ 找到 `!has_fa[root]`	是否用 `while (l || k)` 停在 `0 0`？
回溯合并 `dp[u][0/1]`	根处 `dp[5][0]=4`，`dp[5][1]=5`	负点权时 `dp[u][0]` 会兜底至少为 0 吗？

相对基础模板改了哪里

1. 边是有向“下属 $\leftarrow$ 上司”，只需从 `ch[k]` 单向往下走，不需要传入 `fa`；
2. 树根不一定叫 1 号，必须用 `has_fa[]` 扫出唯一总根再出发。

例 1 | 核心代码（1/2）：读入关系并找出总根

```cpp
int n; if (!(cin >> n)) return 0;
vector<int> happy(n + 1);
for (int i = 1; i <= n; ++i) cin >> happy[i];
vector<vector<int>> ch(n + 1);
vector<bool> has_fa(n + 1, false);
int l, k;
while (cin >> l >> k && (l || k)) {    // 读到 0 0 结束输入
    ch[k].push_back(l);                 // k 是 l 的直接上司
    has_fa[l] = true;                   // 标记 l 有上司，不可能是总根
}
int root = 1;
while (root <= n && has_fa[root]) ++root; // 唯一没有上司的就是校长
```

本页要点

注意输入格式是 L K（先下属后上司），且要读到 0 0 才退出读边循环！

例 1 | 核心代码（2/2）：后序树形 DP 与输出

```
vector<array<long long, 2>> dp(n + 1, {0, 0});
auto dfs = [&](auto &&self, int u) -> void {
    dp[u][1] = happy[u];                    // u 参加，先把自己的欢乐度算上
    for (int v : ch[u]) {
        self(self, v);                      // 先递归算好下属 v 的子树
        dp[u][0] += max(dp[v][0], dp[v][1]); // u 不来：下属来不来取更大
        dp[u][1] += dp[v][0];               // u 来了：下属绝不能来
    }
};
dfs(dfs, root);
cout << max(dp[root][0], dp[root][1]) << '\n';
```

本页要点

完整可运行源码见 code/anniversary_party.cpp；注意 dp[u][0] 始终不低于 0。

例 1 | 易错点与 3 组自测边界

- 常见错误 **1**：直接写死 `dfs(dfs, 1)` ——本题样例根就是 5 号，写 1 会直接输出 1（**WA**）。
- 常见错误 **2**：把读入方向写反成 `ch[l].push_back(k)` ——整棵树父子颠倒（**WA**）。

自测类型	输入数据	预期输出
最小单人	`1 / -5 / 0 0`	0（谁都不邀请最开心）
非 1 为根	`3 / 10 1 10 / 1 2 / 3 2 / 0 0`	20（根是 2，不选 2 改选 1, 3）
负权夹正	`3 / -10 8 -10 / 2 1 / 3 2 / 0 0`	8（只选中间的 2 号）

例 2 | 二叉苹果树 [提高]：先读懂要求

大白话题意

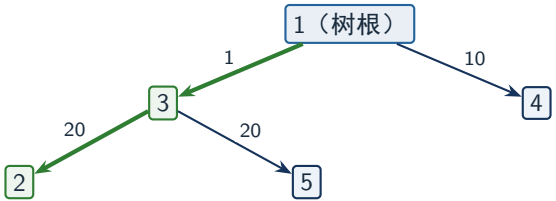
有一棵 N 个节点的苹果树（$1 \le Q \le N \le 100$，树根固定在 1 号）。除了叶子，每个分叉都有两个儿子。每根树枝上长着一定数量的苹果。现在要把树剪成只剩 Q 根树枝（仍挂在根 1 上），求最多能留住多少个苹果？

- 输入：第 1 行 N, Q；接下来 $N-1$ 行每行 u, v, w 表示树枝连接 u, v 且有 w 个苹果。
- 处理：从根 1 出发做树形分组背包，把“保留 Q 根树枝”分摊给各个子树。
- 输出：一行一个整数，最大苹果总数。

原题样例

$N = 5, Q = 2$，四根树枝为 $(1, 3, 1),\ (1, 4, 10),\ (2, 3, 20),\ (3, 5, 20) \Longrightarrow$ 输出 21。

例 2 | 怎么想到的与树形背包建模



- 要留住 2-3 这根 20 个苹果的树枝，必须顺带连向树根的 1-3（虽只有 1 个苹果）。
- “树上选 Q 条连通根的边”直接对应树形分组背包（选边模型）：设 `dp[u][j]` 为子树 u 内共保留 j 条边时的最大苹果数。

本页要点

树枝的苹果长在“边”上，因此往儿子 v 分配 k 条边时，边 (u, v) 本身要先占 1 条！

例 2 | 样例手推跟踪表

按后序先算好 3 的子树（连接 2 和 5，边权均 20），再回到树根 1：

当前合并步骤	合并前 dp[u][0..2]	关键倒序枚举 (j,k)	合并后 dp[u][0..2]
3 合并儿子 2(20)	$[0,0,0]$	$j=1,k=0$: $0+0+20$	$[0,20,0]$
3 合并儿子 5(20)	$[0,20,0]$	$j=2,k=0$: $20+0+20$	$[0,20,40]$
1 合并儿子 3(1)	$[0,0,0]$	$j=2,k=1$: $0+20+1$；$j=1,k=0$: 1	$[0,1,21]$
1 合并儿子 4(10)	$[0,1,21]$	$j=2,k=0$: $\max(21,1+0+10)=21$	$[0,10,21]$
终止输出	根 1 算完所有边	直接取 dp[1][2]	21

本页要点

最后一行 1 合并 4 时，虽然 $1+10=11$ 变大了 1 条边的情况，但 $Q=2$ 仍是 21 更优！

例 2 | 核心步骤与相对模板的变化

按顺序计算	纸上得到	核对问题
无向带权建图	`adj[u]` 存 `{v, w}`	输入无方向，两边都要存，DFS 必须判 `v != fa`
子树内部先递归	`dp[3] = [0, 20, 40]`	`edges[u]` 是否在合并完该儿子后才加？
枚举 (j,k) 更新	`dp[1][2] = 21`	j 必须倒序到 1；分给 v 的 k 至多到 $j-1$

相对基础模板改了哪里

完全贴合第 2 节“选边模型”树形分组背包模板：边权挂在儿子边 (u,v) 上，分配名额时减 1（即 `dp[u][j - 1 - k]`），上界用 `edges[u] + edges[v] + 1` 控制。

例 2 | 核心代码（1/2）：读入无向带权树与初始化

```
int n, q; if (!(cin >> n >> q)) return 0;
vector<vector<pair<int, int>>> adj(n + 1);
for (int i = 0, u, v, w; i < n - 1; ++i) {
    cin >> u >> v >> w;
    adj[u].push_back({v, w});            // 树枝是无向边，两边都建
    adj[v].push_back({u, w});
}
vector<vector<int>> dp(n + 1, vector<int>(q + 1, 0));
vector<int> edges(n + 1, 0);             // edges[u] 记子树 u 当前总边数
```

本页要点

虽然题意说是二叉树，用通用邻接表 adj 存完全通用且不怕输入顺序！

例 2 | 核心代码（2/2）：树形背包倒序合并与输出

```
auto dfs = [&](auto &&self, int u, int fa) -> void {
    for (auto [v, w] : adj[u]) if (v != fa) {
        self(self, v, u);
        int lim = min(q, edges[u] + edges[v] + 1);
        for (int j = lim; j >= 1; --j) {        // 倒序枚举总保留边数
            int upk = min(edges[v], j - 1);
            for (int k = 0; k <= upk; ++k)      // (u,v) 占 1 条，v 分 k 条
                dp[u][j] = max(dp[u][j], dp[u][j - 1 - k] + dp[v][k] + w);
        }
        edges[u] += edges[v] + 1;
    }
};
dfs(dfs, 1, 0); cout << dp[1][q] << '\n';
```

本页要点

完整源码见 code/binary_apple_tree.cpp；根固定为 1，直接输出 dp[1][q]。

例 2 | 易错点与 3 组自测边界

- 常见错误 **1**：内层 j 从 1 到 `lim` 正序枚举——同一儿子树枝会被累加多次（**WA**）。
- 常见错误 **2**：忘了 `edges[u]` 应在循环结束后才加——上界算错漏解或越界（**WA/RE**）。

自测类型	输入数据	预期输出
最小树	`2 1 / 1 2 15`	15（只有 1 根边）
多于 0 苹果	`3 1 / 1 2 0 / 1 3 8`	8（选 1-3）
单侧深重边	`4 2 / 1 2 1 / 2 3 100 / 1 4 50`	101（选 1-2 和 2-3，不能只选 1-4 和 2-3）

例 3 | 旅游规划 [提高]：先读懂要求

大白话题意

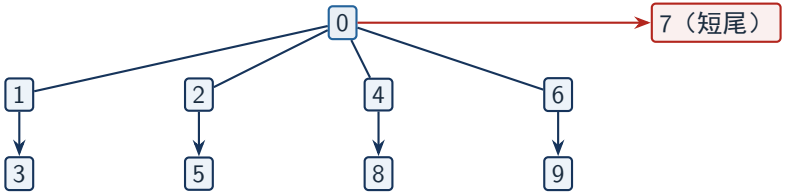
城市有 n 个路口（编号 $0 \sim n-1$，$1 \le n \le 2 \times 10^5$），由 $n-1$ 条街道连成一棵树。把长度最大的简单路径叫做“最长路径”（即树的直径，可能不止一条）。请从小到大输出落在至少一条最长路径上的所有路口编号。

- 输入：第 1 行 n；接下来 $n-1$ 行每行两个整数 u, v。
- 处理：算出每个点往所有方向的最远距离，判断经过该点的最长路径是否等于全树直径。
- 输出：若干行，按编号从小到大每行一个路口编号。

原题样例

$n = 10$，边为 $(0,1),(0,2),(0,4),(0,6),(0,7),(1,3),(2,5),(4,8),(6,9) \Longrightarrow$ 输出 $0,1,2,3,4,5,6,8,9$（唯独 7 号不在）。

例 3 | 怎么想到的与性质转化



- 树的直径长度是 4（如 3-1-0-2-5）。关键性质：点 u 在某条直径上 $\iff$ 经过 u 的两条不相交最长链之和等于直径！
- 对 u 而言，各邻接方向的最远伸展长度只有三个候选：`d1[u]`（向下最长）、`d2[u]`（向下次长）、`up[u]`（向上最长）。

本页要点

只要算出每个点的 `d1`，`d2`，`up`，取三者前两大相加（即 $a+b+c-\min\{a,b,c\}$）与直径一比就知道答案！

例 3 | 样例手推跟踪表

先算全树直径：0 号点 d1[0]=2，d2[0]=2，直径 diam $= 2 + 2 = 4$。挑关键点验证前两大之和：

节点 u	d1[u]	d2[u]	up[u]	前两大之和	等于 diam=4?
0（根）	2	2	0	$2+2=4$	是（输出 0）
1	1	0	3（经 0 到 5）	$1+3=4$	是（输出 1）
3（叶）	0	0	4（经 $1,0,2,5$）	$0+4=4$	是（输出 3）
7（叶）	0	0	3（经 $0,1,3$）	$0+3=3$	否（$3<4$，不输出 7）
终止输出	依次扫 $0\ldots9$	只有 7 被淘汰	——	$0,1,2,3,4,5,6,8,9$	与原题样例完全一致！

本页要点

为什么 7 号的 up[7] 是 3？因为 7 自己贡献 1，父亲 0 没经过 7 的最长向下支 d1[0]=2，所以 $2+1=3$！

例 3 | 核心步骤与相对模板的变化

按顺序计算	纸上得到	核对问题
BFS 建序 `order`	$0 \ldots 9$ 父子序列	节点编号从 0 开始，根为 0，`fa[0]=-1`
逆序算 `d1, d2, c1`	得 `diam = 4`	注意直径可能是 $d_1[u] + d_2[u]$ 在某子树内部达到
正序推 `up[v]`	算出各点 `up`	当 $v == c_1[u]$ 时必须借用 `d2[u]`！
三选二筛输出	按 $0 \ldots n-1$ 扫一遍	即使叶子 $d_1 = d_2 = 0$，只要 $up =$ `diam` 也算直径端点

相对基础模板改了哪里

1. 数据范围高达 $n = 2 \times 10^5$，直接用第 3 节推荐的非递归 **BFS** 序替代递归防止栈溢出；
2. 直径公式统一用 `d1[u] + d2[u]` 取全树最大值（因为直径最高点一定在某点向下两条不同分支拼成）。

例 3 | 核心代码（1/2）：BFS 建序与第一遍自底向上

```
vector<int> fa(n, -1), order = {0};
for (int i = 0; i < (int)order.size(); ++i) {
    int u = order[i];
    for (int v : adj[u]) if (v != fa[u]) {
        fa[v] = u; order.push_back(v);
    }
}
vector<int> d1(n, 0), d2(n, 0), c1(n, -1), up(n, 0);
for (int i = n - 1; i >= 0; --i) {      // 逆序 = 先子后父
    int u = order[i];
    for (int v : adj[u]) if (v != fa[u]) {
        int len = d1[v] + 1;
        if (len > d1[u]) { d2[u] = d1[u]; d1[u] = len; c1[u] = v; }
        else if (len > d2[u]) d2[u] = len;
    }
}
```

例 3 | 核心代码（2/2）：第二遍换根与输出直径点

```
for (int u : order) {                          // 正序 = 先父后子换根
    for (int v : adj[u]) if (v != fa[u]) {
        int sib = (c1[u] == v ? d2[u] : d1[u]); // 借用次长支防折返
        up[v] = max(up[u], sib) + 1;
    }
}
int diam = 0;
for (int u = 0; u < n; ++u) diam = max(diam, d1[u] + d2[u]);
for (int u = 0; u < n; ++u) {
    int a = d1[u], b = d2[u], c = up[u];
    if (a + b + c - min({a, b, c}) == diam) cout << u << '\n';
}
```

本页要点

完整可运行源码见 code/travel_plan.cpp；节点编号从 0 到 $n-1$ 自然升序输出。

例 3 | 易错点与 3 组自测边界

- 常见错误 **1**：用深递归在 $n = 2 \times 10^5$ 单链测试时栈溢出（**RE**），改用 BFS 序彻底解决。
- 常见错误 **2**：求 `diam` 时只看 `d1[0] + d2[0]` ——如果直径不经过 0 号，在更深子树里，必须遍历全部 u 取 $\max(\texttt{d1[u]+d2[u]})$（**WA**）。

自测类型	输入数据	预期输出
单点最小树	`1`（无边）	`0`（直径 0，唯一节点在直径上）
两点最小边	`2 / 0 1`	`0` 和 `1`
直径不经 0	`4 / 0 1 / 1 2 / 1 3`（星形中心在 1）	`0`，`1`，`2`，`3`（1 处 `d1+d2=2` 才是直径）

常见错误对照（一）：树形基础与树形背包

场景	常见错误写法	正确做法与原因
写死 1 号作根	有向“下属-上司”直接搜 1	老板未必是 1 号！找没有父亲的点作 root
无向树没防回头	扫邻居不写 `if (v == fa) continue;`	沿无向边回到父亲导致死循环 **RE**
树形背包正序枚举	`for (int j = 1; j <= lim; ++j)`	同一棵儿子子树会被重复叠加多次，必须倒序！
树形背包先改大小	循环前就写 `edges[u] += edges[v] + 1`	上界和循环内逻辑会混用新旧大小，务必写在循环后

本页要点

树形基础查“根与回头边”，树形背包查“倒序枚举与边数更新位置”。

常见错误对照（二）：二次扫描换根 DP

场景	常见错误写法	正确做法与原因
换根未踢自己分支	`up[v] = max(up[u], d1[u]) + 1`	若 v 就是 `c1[u]`，会原路折返成假路径！应借用 `d2[u]`
两遍扫描顺序写反	先推 `up` 再算 `d1`，`d2`	上方贡献依赖兄弟的向下长度，必先自底向上再自顶向下
直径只看总根	写 `diam = d1[0] + d2[0]`	最长路可能不经过 0 号！应遍历全部 u 取 $\max(d_1[u]+d_2[u])$
长链递归栈溢出	$n = 2 \times 10^5$ 写深层递归	Windows 栈空间小易 **RE**，用 BFS 序 `order[]` 非递归最稳

本页要点

换根两遍顺口溜：第一遍逆序先子后父算 `down`，第二遍正序先父后子算 `up`！

评测结果快速定位指南

提交后如果没 AC，先看评测机返回的状态再对症下药：

- **WA**（答案错误）：先手算一个 4 点小树核对；有向树查是否找错了根，树形背包查 `j` 是否倒序、是否扣掉了连接边 1，换根查是否忘了用 `d2[u]` 剔除 v 自己。
- **TLE**（超时）：检查树形背包有没有加上 `min(Q, edges[u]+edges[v]+1)` 上界优化；检查是不是每个点重跑了一遍 DFS 导致 $O(n^2)$ 而不是二次扫描 $O(n)$。
- **RE**（运行错误）：无向图是不是漏了 `v == fa` 导致无限递归？或者 $n \geq 10^5$ 单链树引发了递归栈溢出？
- **CE**（编译错误）：检查 C++17 结构化绑定 `auto [v, w]` 或泛型 lambda 写法是否漏开了 `-std=c++17`。

本页要点

用“4 点单链”和“4 点菊花图”各手推一遍，能提前抓出 90% 的树形与换根 BUG！

提交前检查清单（六必查）

1. 无向树遍历邻接表时，是否严格拦住了父节点 `if (v == fa) continue;`？
2. 有向父子树题面（如公司上下级），是否正确找出了没有父亲的唯一 `root`？
3. 树形背包中，外层总容量 j 是否从大到小倒序枚举，且 `sz/edges[u]` 写在儿子合并循环后？
4. 换根求最值时，是否同时存了最长 `d1`、次长 `d2` 与最长子编号 `c1`，并在 `c1[u]==v` 时切换 `d2[u]`？
5. 第二遍换根是否严格按照“自顶向下（父 $\rightarrow$ 子）”顺序更新 `up[v]`？
6. 数据规模达到 $n \geq 10^5$ 时，是否改用了 BFS 序 `order[]` 非递归扫描防爆栈？

最后一步

交题前用最小值 $n = 1$（无边）和单链 $n = 4$ 各跑一次本地验证，确认不越界、不漏根！

本章全部练习与完成标准

1. 树上独立集：周年纪念晚会（一本通 1582，`code/anniversary_party.cpp`，[基础]）；选做：战略游戏（最小点覆盖）。
2. 树形背包：二叉苹果树（一本通 1575，`code/binary_apple_tree.cpp`，[提高]）；选做：选课（有向森林加虚拟根 0）。
3. 换根 **DP**：旅游规划（一本通 1581，`code/travel_plan.cpp`，[提高]）。

完成标准

能合上课件独立写出 `dp[u][0/1]` 与树形背包倒序转移；能画图讲清为什么树形背包是 $O(n^2)$ 以及换根为什么要用 `d2[u]`；独立 AC 上述三道主力真题。

课后安排、下一章预告与本章一句话

- 复习：合上代码重新推一遍《旅游规划》里“过点 u 的三条边长取前两大”与《二叉苹果树》的倒序表。
- 练习：独立复写 code/ 下三份源码，并用题库里的边界数据验证正序/倒序与防回头写法。
- 下一章预告：第 4 章《状压与数位 DP》——当题目给你的不再是一棵树，而是一张 $N \times M$ 小棋盘（需要把一整行状态压缩进一个二进制整数）或一个高达 2×10^9 的大区间整数计数时，看状态压缩和四参数记忆化数位 DP 如何大显身手！

本章一句话

树形 DP 等孩子汇报完再汇总给父亲，换根 DP 让父亲反过来把外面的世界告诉孩子。

谢谢！