中国计算机学会 China Computer Federation

树 的 算 法 及 其 应 用

金靖

2021.05.11

树的定义

一个没有固定根结点的树称为**无根树**。

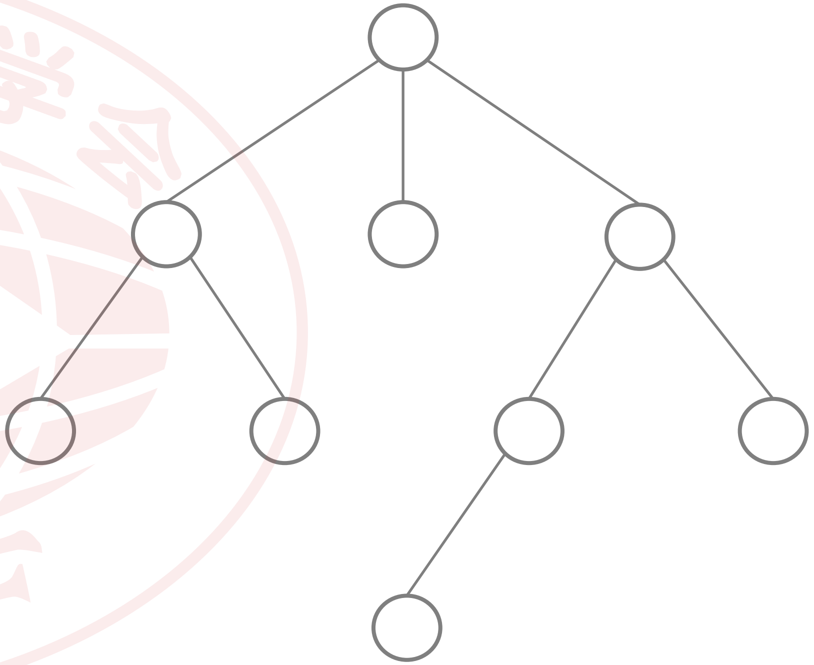
无根树有几种等价的形式化定义：

- 有 n 个结点，$n-1$ 条边的连通无向图
- 无向无环的连通图
- 任意两个结点之间有且仅有一条简单路径的无向图
- 任何边均为桥的连通图

在无根树的基础上，指定一个结点称为 **根**，则形成一棵 **有根树**。

有根树在很多时候仍以无向图表示，只是规定了结点之间的上下级关系。

树的存储：一般使用邻接表存图（链式前向星）的方式存储树结构。



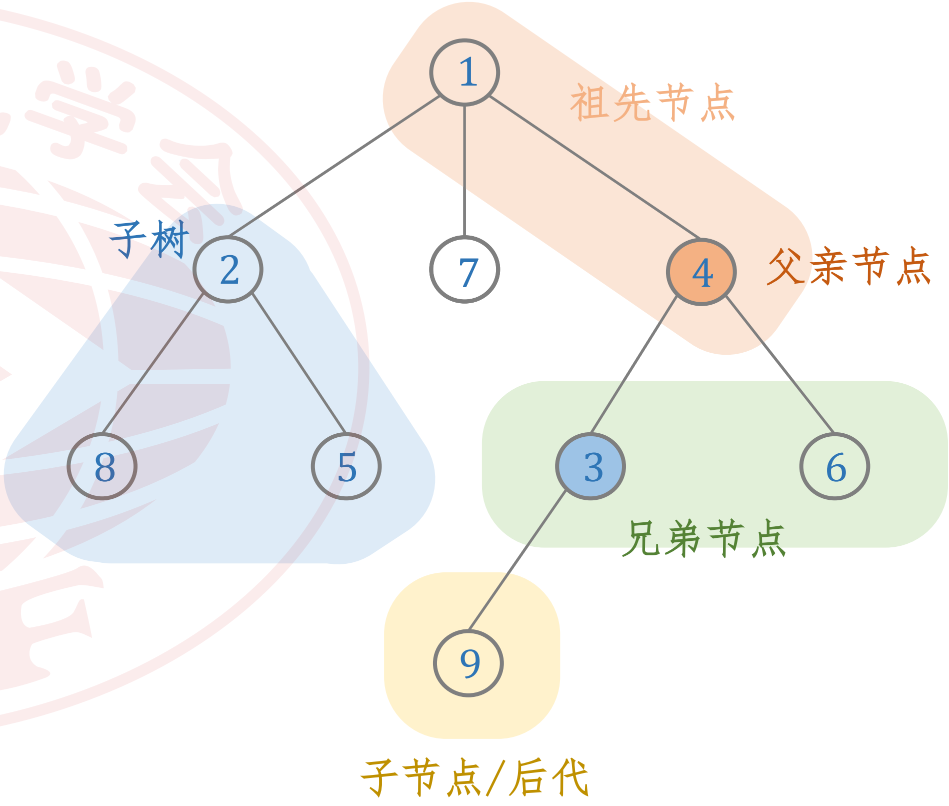
树的定义

适用于无根树和有根树：

- 森林：每个连通分量（连通块）都是树的图。按照定义，一棵树也是森林。
- 生成树：一个连通无向图的生成子图，同时要求是树。也即在图的边集中选择 $n-1$ 条，将所有顶点连通。

只适用于有根树：

- 父亲：对于除根以外的每个结点，定义为从该结点到根路径上的第二个结点。 根结点没有父结点。
- 祖先：一个结点到根结点的路径上，除了它本身外的结点。
- 子结点：如果 u 是 v 的父亲，那么 v 是 u 的子结点。
- 兄弟：同一个父亲的多个子结点互为兄弟。
- 后代：子结点和子结点的后代。
- 子树：删掉与父亲相连的边后，该结点所在的子图。

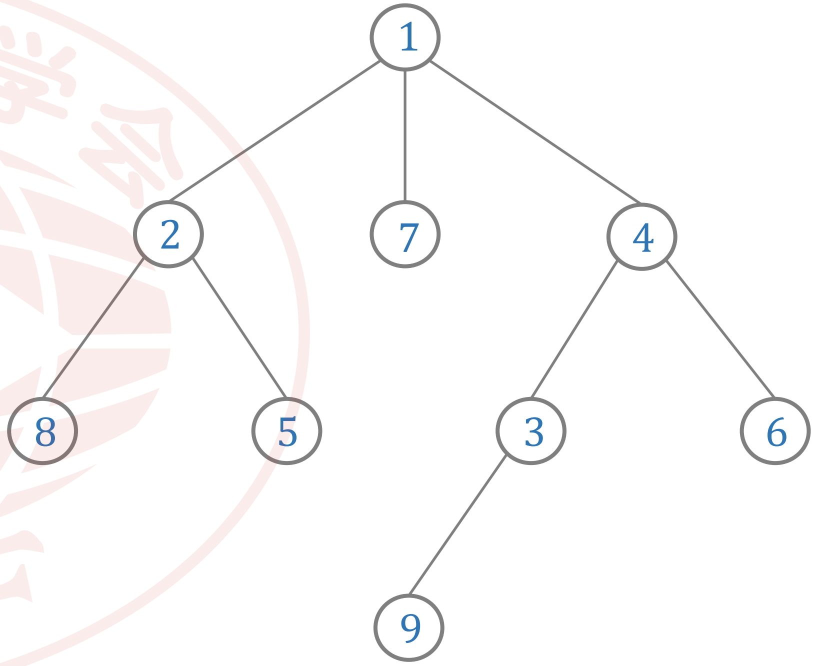


树的深度优先遍历

深度优先遍历（DFS）：在每个节点 x 上面对多条分支时，任意选一条访问，执行递归，直至回溯到 x 后，再考虑访问其他的边。

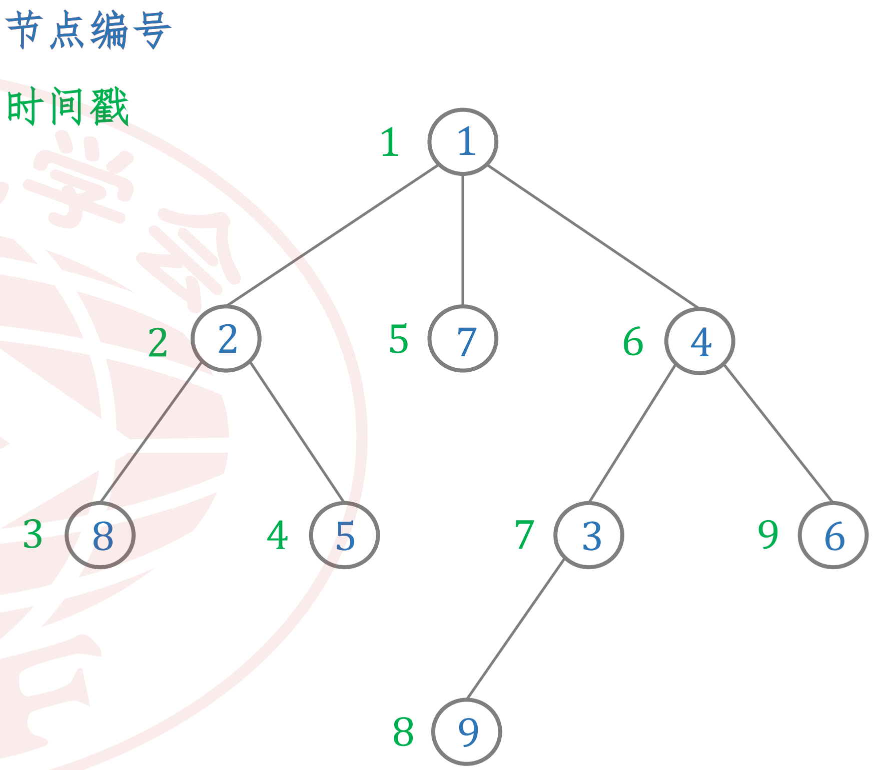
DFS 访问树中的每个点和每条边恰好 1 次，时间复杂度为 $O(N+M)$。

节点编号



时间戳

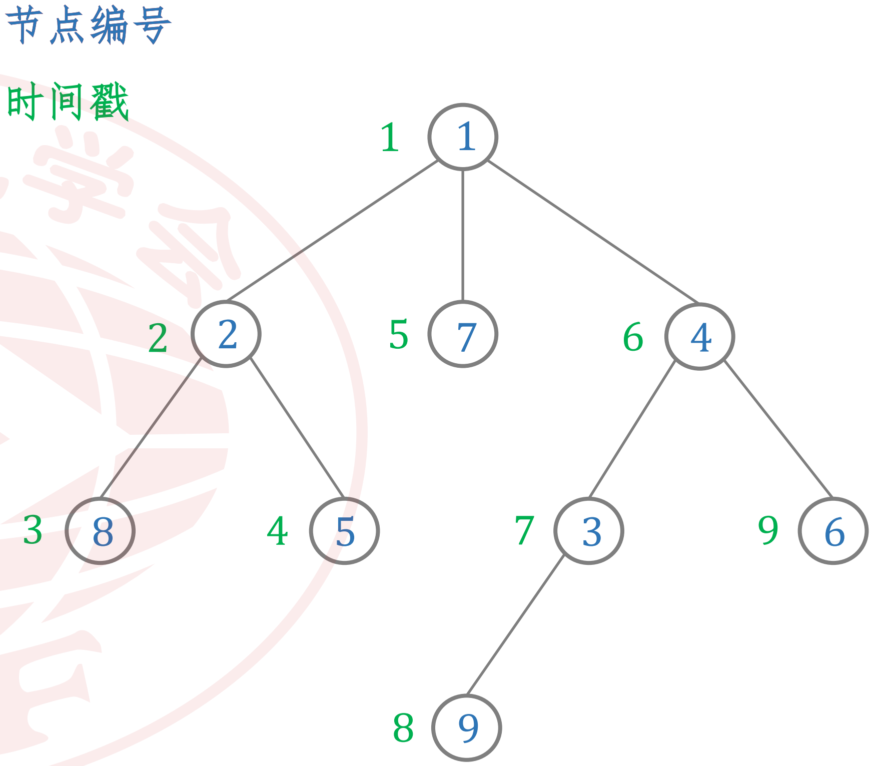
按照 DFS 遍历的过程，以**每个节点第一次被访问**（$vis[x]$ 被赋值为 1 时）的顺序，依次给予这 N 个节点 1~N 的整数标记，该标记被称之为 **时间戳**，记为 dfn。



树的 DFS 序

在对树进行 DFS 遍历时，对于每个节点，在**刚进入递归后以及即将回溯前**各记录一次该点的编号，最后产生的长度为 $2N$ 的节点序列，被称之为 **树的 DFS 序**。

在 DFS 序中，每个节点 x 的编号在序列中恰好出现 2 次。设这两次出现的位置为 $L[x]$ 和 $R[x]$，那么闭区间 $[L[x], R[x]]$ 就是以 x 为根的子树的 DFS 序。

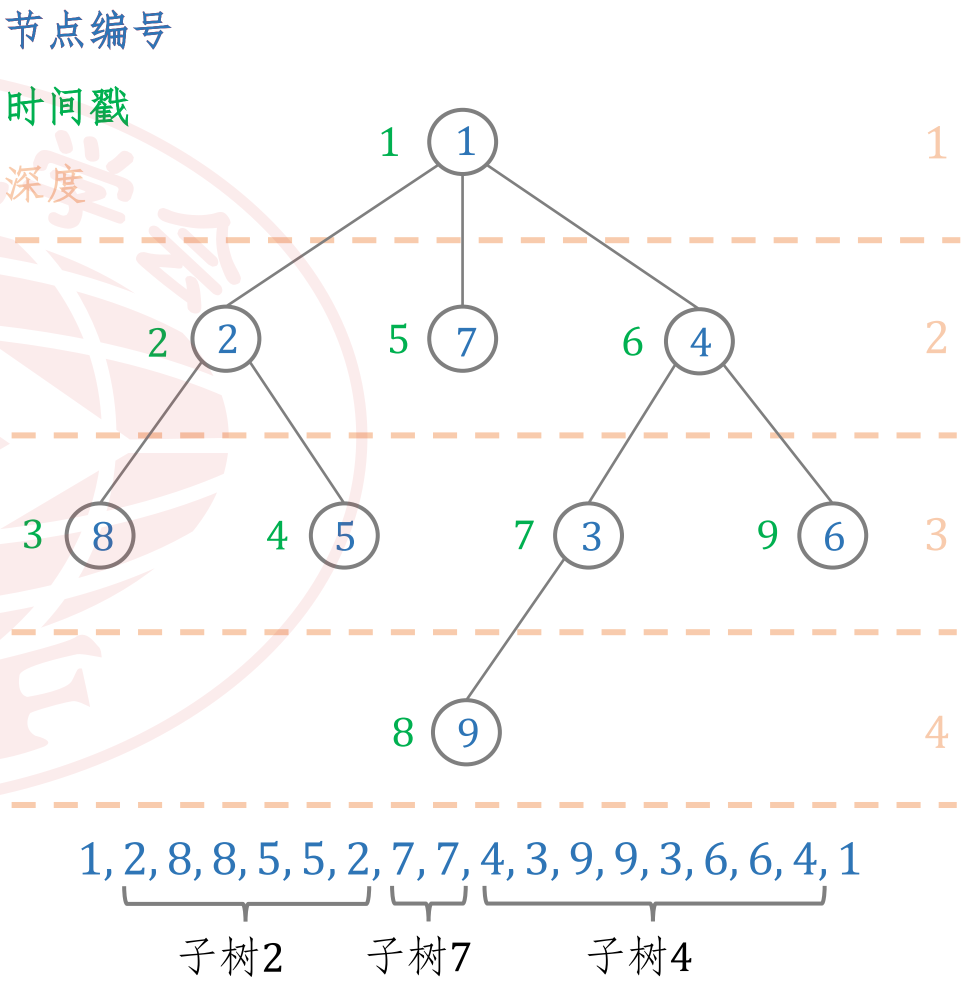


$$\underbrace{1,2,8,8,5,5,2,}_{\text{子树}2}\ \underbrace{7,7,}_{\text{子树}7}\ \underbrace{4,3,9,9,3,6,6,4,}_{\text{子树}4}1$$

树的深度

树中各节点的深度是一种自顶向下的统计信息。

已知根节点的深度为 1，若节点 x 的深度为 $d[x]$，则它的子节点 y 的深度就是 $d[y] = d[x] + 1$。

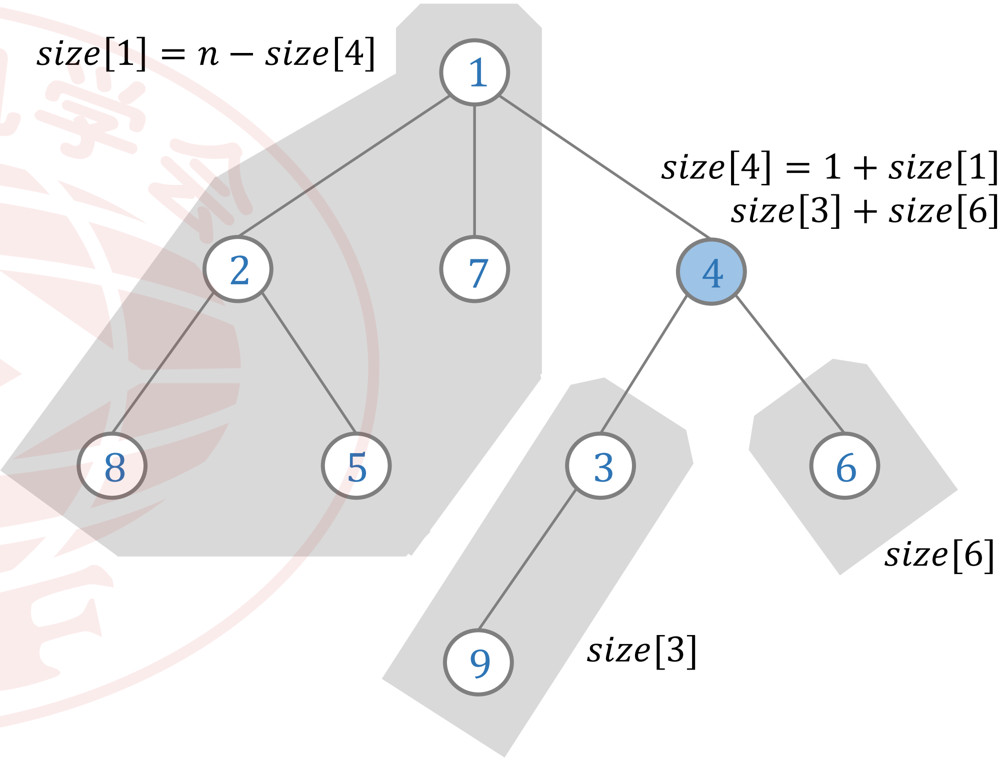


子树大小

以每个节点 x 为根的子树大小 $size[x]$，可以自底向上进行统计。

对于叶子节点，“以它为根的子树”大小为 1。

若节点 x 有 k 个子节点 $y_1 \sim y_k$，并且以 $y_1 \sim y_k$ 为根的子树大小分别是 $size[y_1], size[y_2], \cdots, size[y_k]$，则以 x 为根的子树的大小就是 $size[x] = 1 + size[y_1] + size[y_2] + \cdots + size[y_k]$。

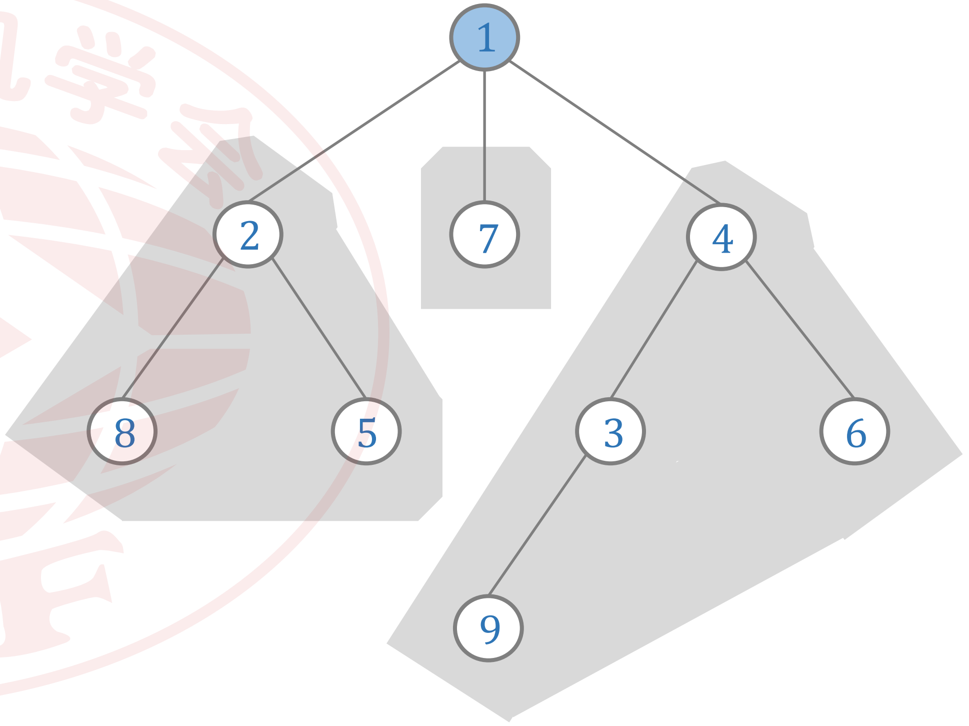


树的重心

如果在树中选择某个节点 x 并删除，这棵树将分为若干不相连的部分，每一部分都是一颗子树。

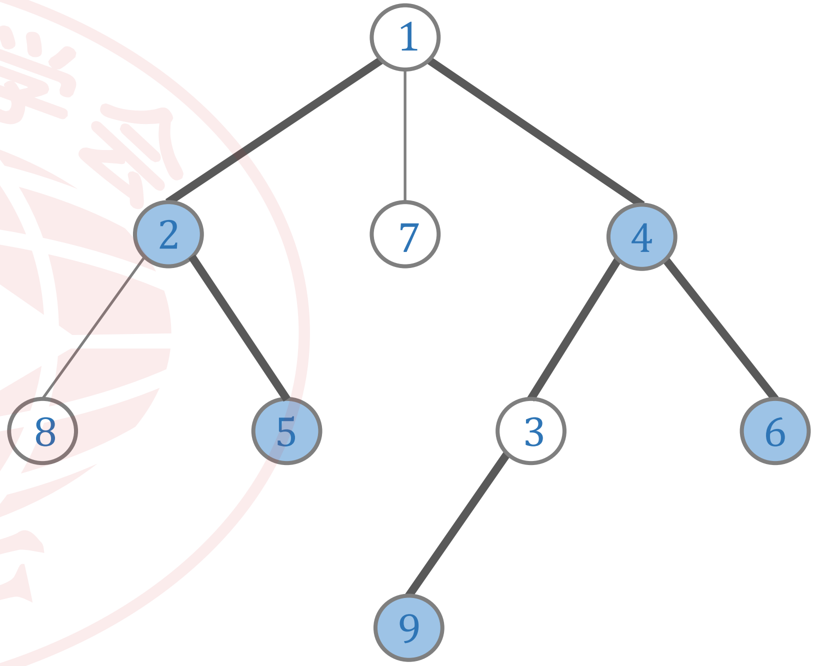
设 $\max\_\mathrm{part}(x)$ 表示在删除节点 x 后产生的子树中，最大的一颗子树的大小。

使 $\max\_\mathrm{part}$ 函数取到最小值的节点 p 就被称为整个**树的重心**。



完整题面请访问
LibreOJ 10132
AcWing 355

异象石



给定 N 个点 和 $N-1$ 条无向边构成的树。

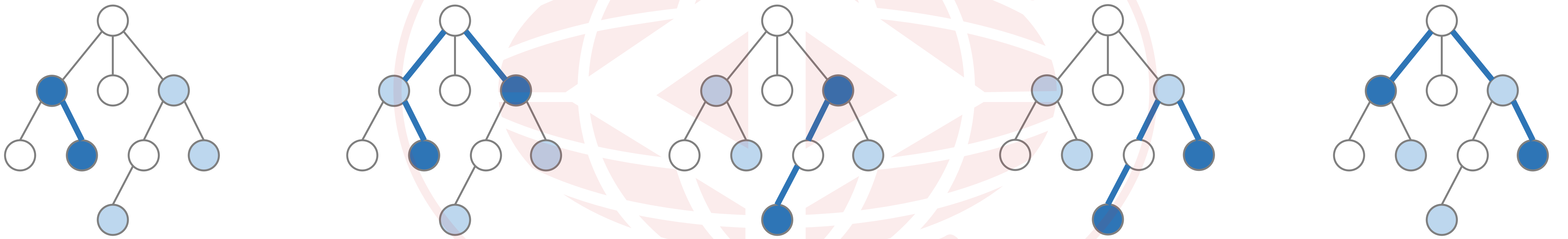
异象石是一种特殊的物体，起初树上的所有节点，均没有异象石。在接下来的 M 个时刻中 $(1 \le N, M \le 10^5)$，每个时刻会发生以下三种类型的事件之一：

1. 树上的某个点上出现了异象石（已经出现的不会再次出现）；
2. 树上某个点上的异象石被摧毁（不会摧毁没有异象石的点）；
3. 询问使用异象石所在的点连通的边集的总长度最小是多少。

完整题面请访问
LibreOJ 10132
AcWing 355

异象石

如果按照时间戳从小到大的顺序，把出现异象石的节点排成一圈（首尾相连），并且累加相邻两个节点之间的路径长度，最后得到的结果恰好是所求答案的两倍。下图中深蓝色节点表示**按照时间戳顺序**依次选定的两个节点，深蓝色边表示二者之间的路径。五幅图合起来，恰好把例图中加粗的边集覆盖了两次。

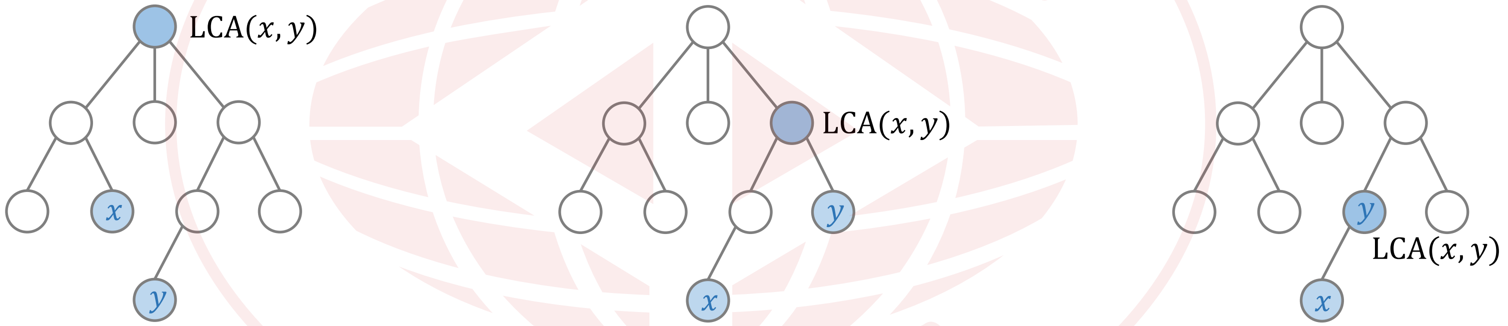


1. 按照时间戳递增的顺序，维护出现异象石的节点序列；
2. 用 ans 记录序列中相邻两个节点之间的路径长度之和（含首尾）；
3. 设树上 x,y 之间的路径长度为 $\text{path}(x,y)$；
4. 根据新增或摧毁异象石节点编号，在序列中插入或删除；例如插入节点 x，它在序列中的前后分别是节点 l 和 r，就令 ans 减去 $\text{path}(l,r)$，减去 $\text{path}(l,x)+\text{path}(x,r)$。

最近公共祖先 Lowest Common Ancestor

给定一颗有根树，若节点 z 既是节点 x 的祖先，也是节点 y 的祖先，则称 z 是 x,y 的公共祖先。

在 x,y 的所有公共祖先中，深度最大（靠 x,y 最近）的一个节点称之为 x,y 的最近公共祖先，记为 $\mathrm{LCA}(x,y)$。



$\mathrm{LCA}(x,y)$ 是 x 到根的路径与 y 到根的路径的交会点，也是 x 与 y 之间的路径上深度最小的节点。

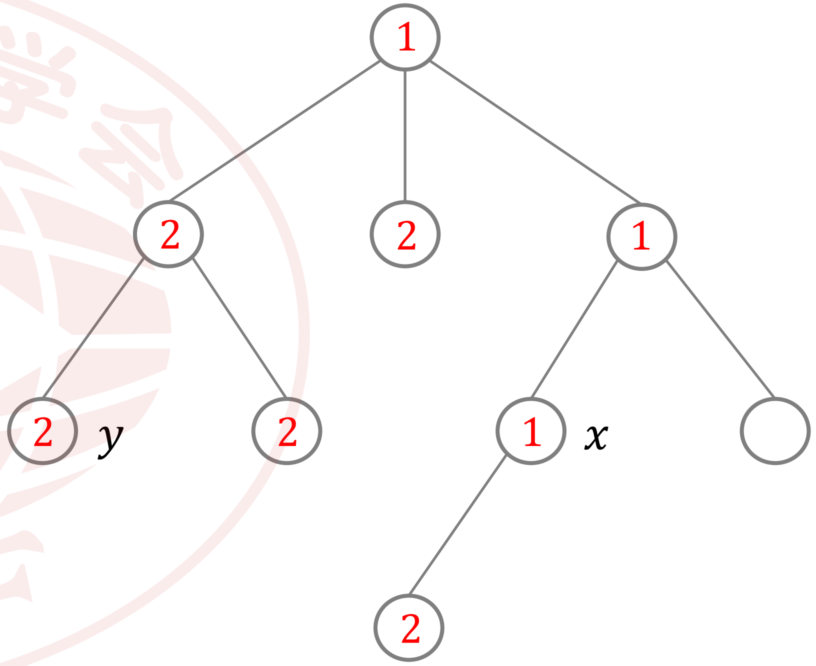
向上标记法求 LCA ：从 x 向上走到根节点，并标记所有经过的节点；从 y 向上走到根节点，当一次遇到已标记的节点时，就找到了 $\mathrm{LCA}(x,y)$。对于每个询问，向上标记法的查询时间复杂度最坏为 $O(n)$ 。

LCA 的 Tarjan 算法

在对树 DFS 遍历的任意时刻，树上节点分为三类：

1. 已经开始递归，但尚未回溯的节点（即正在访问的节点 x 以及 x 的祖先）。此类节点标记为整数 1；
2. 已经访问完毕并且回溯的节点。此类节点标记为整数 2；
3. 尚未访问的节点。此类节点没有标记。

对于正在访问的节点 x，它到根节点的路径已经标记为 1。若 y 是已经访问完毕并且回溯的节点，则 LCA(x,y) 就是从 y 向上走到根，第一个遇到的标记为 1 的节点。



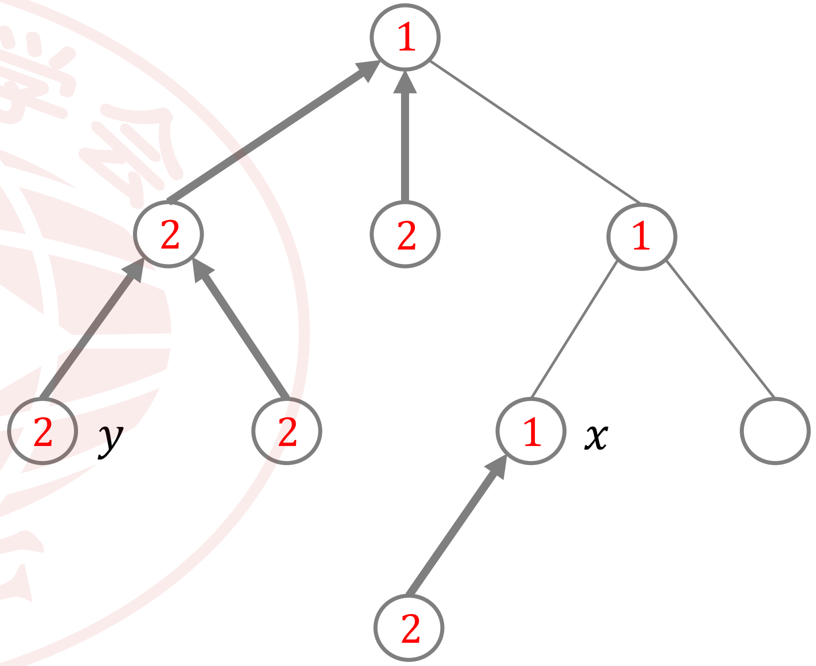
LCA 的 Tarjan 算法

如何快速查询已回溯的节点 y 向上走到根的路径上第一个标记为 1 的节点?

可以利用**并查集**进行优化:

当一个节点获得整数 2 的标记时,把它所在的集合**合并**到它的父节点所在的集合中(合并时它的父节点的标记一定为 1,且单独构成一个集合)。

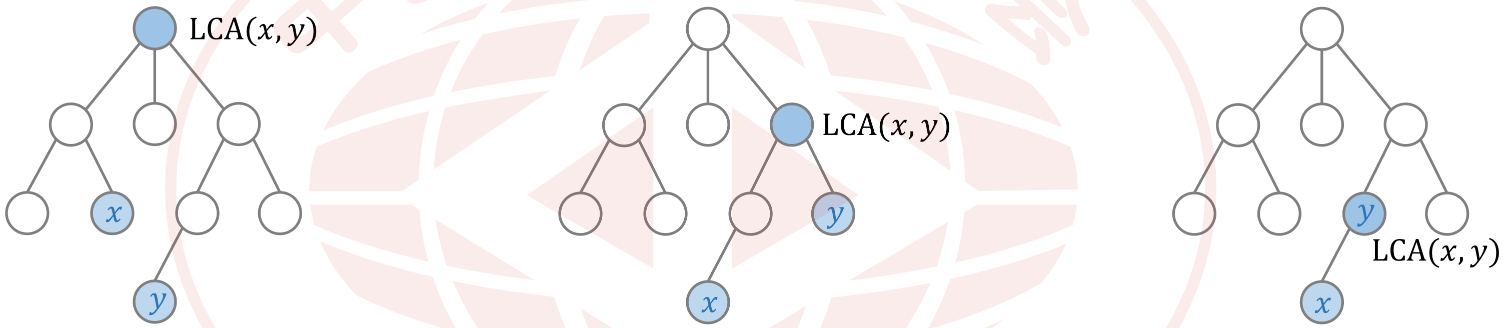
这相当于每个完成回溯的节点都有一个指针指向它的父节点,只需要查询 y 所在的集合的代表元素,就等价于从 y 向上一直走到一个开始递归但尚未回溯的节点(具有标记 1),即 $\mathrm{LCA}(x,y)$。



完整题面请访问
Hdu 2856

How far away

题意：给定一棵树，回答多次询问树上两点之间的距离。



根据 LCA 的性质，可知 $d(x,y) = h(x) + h(y) - 2h(\mathrm{LCA}(x,y))$，其中 $d(x,y)$ 是树上两点间的距离，h 代表某点到树根的距离。

当访问到 x 点时，扫描与 x 相关的所有询问，对于每一个询问，如另一节点 y 的标记为 2，则可知 $\mathrm{LCA}(x,y)$的答案应为 y 在并查集中的代表元素。

把 m 个询问一次性读入，统一计算，最后统一输出。时间复杂度为 $O(m+n)$。

紧急集合

树上有三个节点 a,b,c，在树上确定一个节点 x，使 a,b,c 三点各自到达 x 的距离之和最小，求这个值为多少？

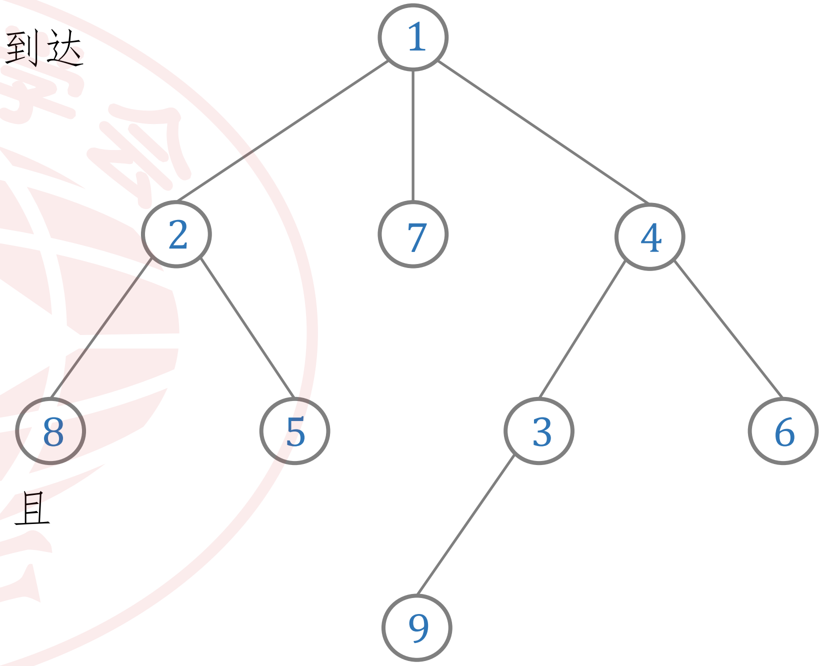
分类讨论：

1. a,b,c 三点处于一条链上，如图中节点 1,2,8；
2. a,b 两点处于一条链上，c 点在链外，如图中节点 1,8,5；
3. a,b 两点处于一条链上，c 点在链外，如图中节点 8,5,4；

均可得到结论：LCA(a,b)、LCA(a,c) 和 LCA(b,c) 中必有两点相同，且 x 点位于深度较大的一个 LCA 点上。

所求答案即为：

$$deep(a)+deep(b)+deep(c)-deep(\text{最深LCA点})-deep(\text{最浅LCA点})\times 2$$



倍增算法

如果状态空间很大，线性递推无法满足，可选择成倍增长的方式。

如果状态空间可以按 2 的次幂来划分，那就只记录或选取在 2 的整数次幂位置上的值，来作为代表。当需要其他值时，利用“任意整数可以表示为若干个 2 的次幂项的和”的性质来解决即可。

常见应用有 ST 表求 RMQ，树状数组。

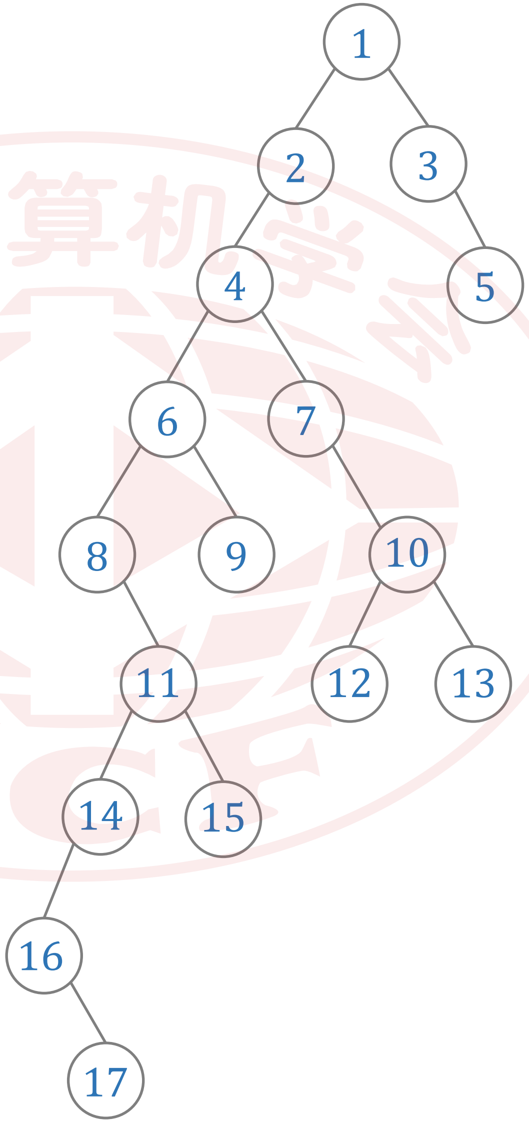
树上倍增法求 LCA

1. 对树进行 BFS/DFS 遍历，预处理 F 数组，时间复杂度为 $O(n\log n)$

- 设 $F[x,k]$ 为 x 的 2^k 辈祖先，即从 x 向根节点走 2^k 步到达的节点。$F[x,0]$ 为 x 的父节点；
- 如果该节点不存在，则令 $F[x,k]=0$；
- 对于其他节点，

  $\forall k\in[1,\log n], F[x,k]=F[F[x,k-1],k-1]$，

  即 x 的 2^{k-1} 辈祖先的 2^{k-1} 辈祖先，是 x 的 2^k 辈祖先。



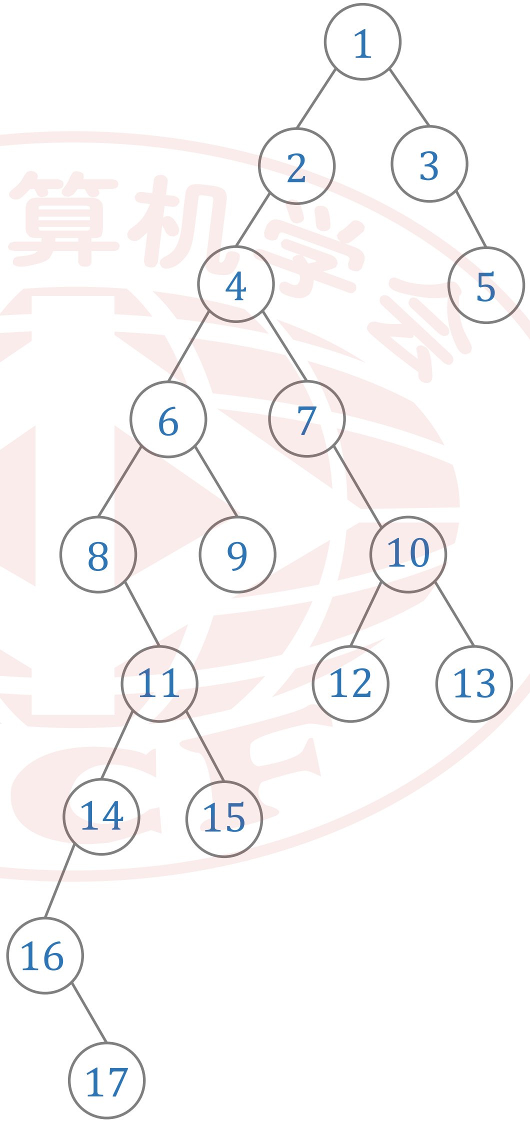
$F[x,k]$ 表格

id	0	1	2	3
1	0	0	0	0
2	1	0	0	0
3	1	0	0	0
4	2	1	0	0
5	3	1	0	0
6	4	2	0	0
7	4	2	0	0
8	6	4	1	0
9	6	4	1	0
10	7	4	1	0
11	8	6	2	0
12	10	7	2	0
13	10	7	2	0
14	11	8	4	0
15	11	8	4	0
16	14	11	6	0
17	16	14	8	1

树上倍增法求 LCA

2. 基于 F 数组计算 LCA(x,y)，为 $O(\log n)$

- 设 $d[x]$ 表示 x 的深度，可交换 x,y，以确保 $d[x] \geq d[y]$；
- 依次尝试从 x 向上走 $k = 2^{\log n},\cdots,2^1,2^0$ 步（注意间距从大到小），检查节点 $x' = F[x,k]$ 是否满足 $d[x'] \geq d[y]$，如果满足，则 $x = x'$，即向上跃升。当退出倍增跃升时，$d[x] = d[y]$。
- 如果此时 $x = y$，则 LCA$(x,y) = y$，即 x 和 y 在一条路径上，y 是 x 的祖先节点。
- 否则说明 x 和 y 各居 LCA(x,y) 的一侧，则依次尝试从 x,y 同时向上走 $k = 2^{\log n},\cdots,2^1,2^0$ 步，如果 $F[x,k] \neq F[y,k]$，则令 $x = F[x,k], y = F[y,k]$。当退出倍增跃升时，x,y 必定只差一步就相遇，它们的父节点 $F[x,0]$ 就是 LCA。



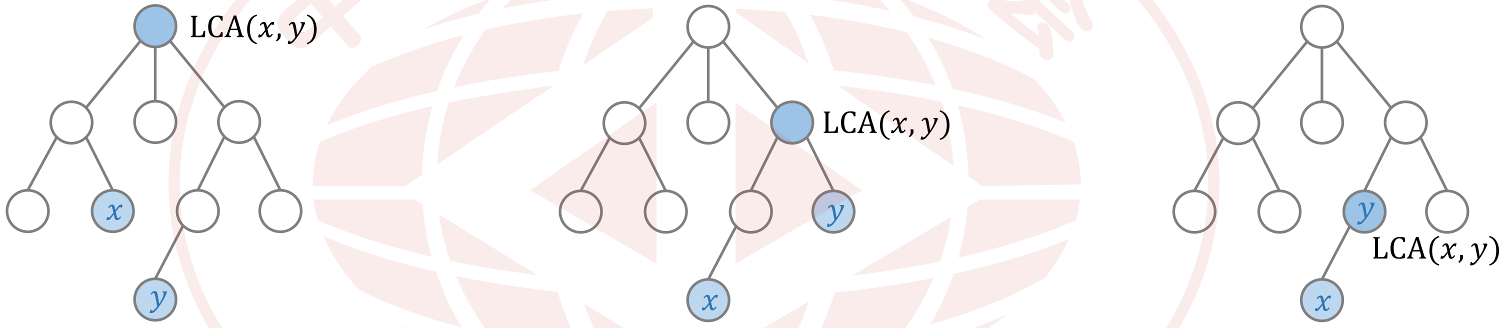
$F[x,k]$ 表格

id	0	1	2	3
1	0	0	0	0
2	1	0	0	0
3	1	0	0	0
4	2	1	0	0
5	3	1	0	0
6	4	2	0	0
7	4	2	0	0
8	6	4	1	0
9	6	4	1	0
10	7	4	1	0
11	8	6	2	0
12	10	7	2	0
13	10	7	2	0
14	11	8	4	0
15	11	8	4	0
16	14	11	6	0
17	16	14	8	1

完整题面请访问
Hdu 2856

How far away

题意：给定一棵树，回答多次询问树上两点之间的距离。



通过树上倍增法求 LCA，多次查询树上两点之间的距离，时间复杂度为 $O((n+m)\log n)$。

次小生成树

完整题面请访问
LibreOJ 10133
LuoGu P4180
AcWing 356

给定一张 $N(N \leq 10^5)$ 个点 $M(M \leq 3\times10^5)$ 条边的无向图，求无向图的次小生成树。

设最小生成树的边权之和为 sum，严格次小生成树就是指边权之和大于 sum 的生成树中最小的一个。

次小生成树

完整题面请访问
LibreOJ 10133
LuoGu P4180
AcWing 356

先求出任意一颗最小生成树，设边权之和为 sum，树上每条边为“树边”，共 $N-1$ 条，其他 $M-N+1$ 条边为“非树边”。

把一条非树边 (x,y,z) 添加到最小生成树中，会与 $\delta(x,y)$ 形成环。设 $\delta(x,y)$ 中最大边权为 val_1，严格次大边权为 $val_2(val_1 > val_2)$。

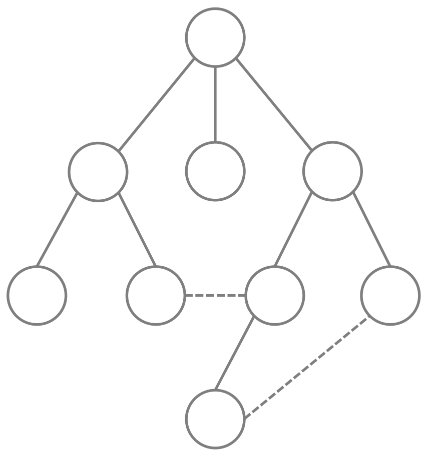
若 $z > val_1$，则把 val_1 对应的边替换成 (x,y,z)，就得到严格次小生成树的一个候选答案，边权之和为 $sum - val_1 + z$。

若 $z = val_1$，则把 val_2 对应的边替换成 (x,y,z)，就得到严格次小生成树的一个候选答案，边权之和为 $sum - val_2 + z$。

根据最小生成树的定义，不会出现 $z < val_1$ 的情况。

枚举每条非树边，添加到最小生成树中，计算出上述所有“候选答案”。在候选答案中取最小值就得到了整张无向图的严格次小生成树。

问题是：**如何快速求出一条路径上的最大边权与严格次大边权。**



次小生成树

完整题面请访问
LibreOJ 10133
LuoGu P4180
AcWing 356

树上倍增算法进行预处理：设 $F[x,k]$ 表示 x 的 2^k 辈祖先，$G[x,k,0]$ 和 $G[x,k,1]$ 分别表示从 x 到 $F[x,k]$ 的路径上的最大边权和严格次大边权（最大边权不等于次大边权）。于是 $\forall k \in [1, \log N]$ 有：

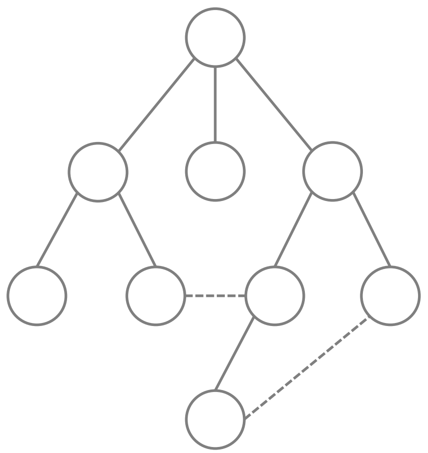
$$F[x,k] = F[F[x,k-1],k-1]$$

$$G[x,k,0] = \max(G[x,k-1,0], G[F[x,k-1],k-1,0])$$

$$G[x,k,1] = \begin{cases} \max(G[x,k-1,1], G[F[x,k-1],k-1,1]) & G[x,k-1,0] = G[F[x,k-1],k-1,0] \\ \max(G[x,k-1,0], G[F[x,k-1],k-1,1]) & G[x,k-1,0] < G[F[x,k-1],k-1,0] \\ \max(G[x,k-1,1], G[F[x,k-1],k-1,0]) & G[x,k-1,0] > G[F[x,k-1],k-1,0] \end{cases}$$

当 $k = 0$ 时，有初值：

$$F[x,0] = \text{father}(x) \qquad G[x,0,0] = \text{edge}(x, \text{father}(x)) \qquad G[x,0,1] = -\infty(\text{不存在次大值})$$



考虑每条非树边 (x,y,z)。采用倍增计算 $\text{LCA}(x,y)$ 的框架，x,y 每向上移动一段路径，就将这段路径对应的最大边权和严格次大边权按照与求 G 数组类似的方法合并到答案中，最后即可得到树上 x,y 之间的路径上的最大边权和严格次大边权。

整个算法的时间复杂度为 $O(M \log N)$。

用欧拉序将 LCA 转化为 RMQ 问题

对一棵树进行 DFS，无论是第一次访问还是回溯，每次到达一个结点时都将编号记录下来，可以得到一个长度为 $2n-1$ 的序列，这个序列被称作这棵树的**欧拉序**。

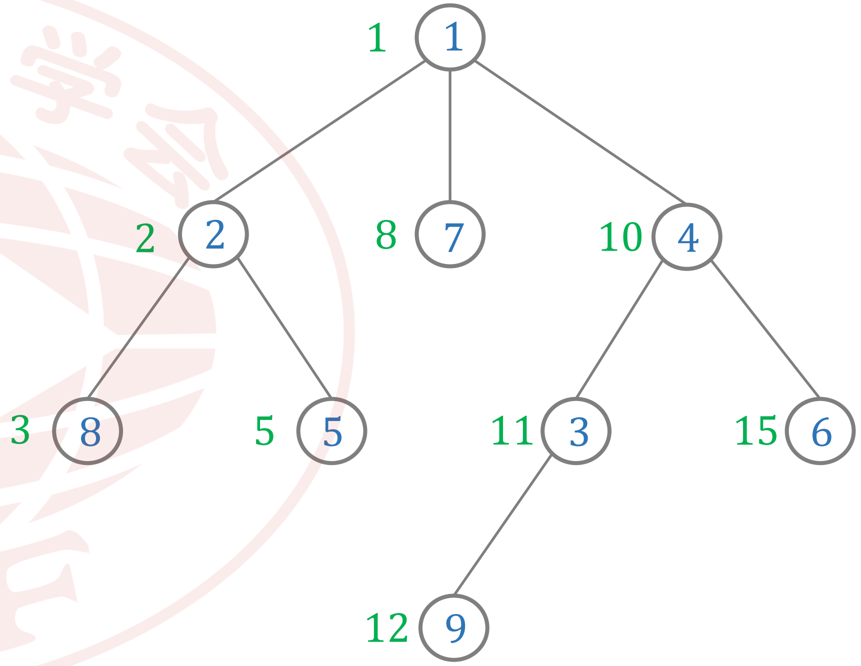
把结点 u 在欧拉序中第一次出现的位置编号记为 $pos(u)$（也称作节点 u 的欧拉序），把欧拉序本身记作 $E[1..2n-1]$ 。

从 u 走到 v 的过程中一定会经过 $\mathrm{LCA}(u,v)$，但不会经过 $\mathrm{LCA}(u,v)$ 的祖先。

因此，从 u 走到 v 的过程中经过的欧拉序最小的结点就是 $\mathrm{LCA}(u,v)$ 。

$$pos\big(\mathrm{LCA}(u,v)\big)=min\{pos(k)|k\in E[pos(u)..pos(v)]\}$$

用 DFS 计算欧拉序列的时间复杂度是 $O(n)$，且欧拉序的长度也是 $O(n)$，所以 LCA 问题可以在 $O(n)$ 的时间内转化成等规模的 RMQ 问题。



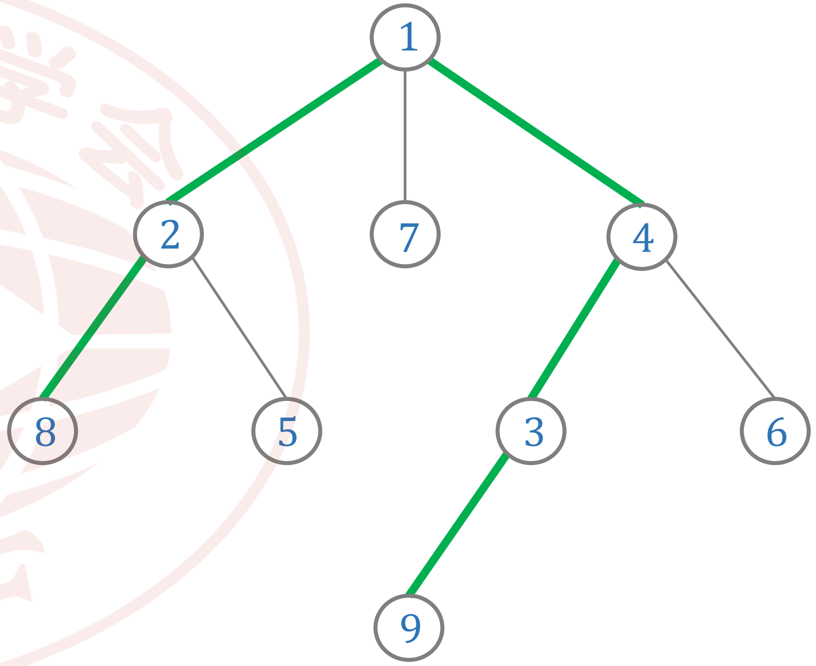
$1,2,8,2,5,2,1,7,1,4,3,9,3,4,6,4,1$

$1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17$

树的直径

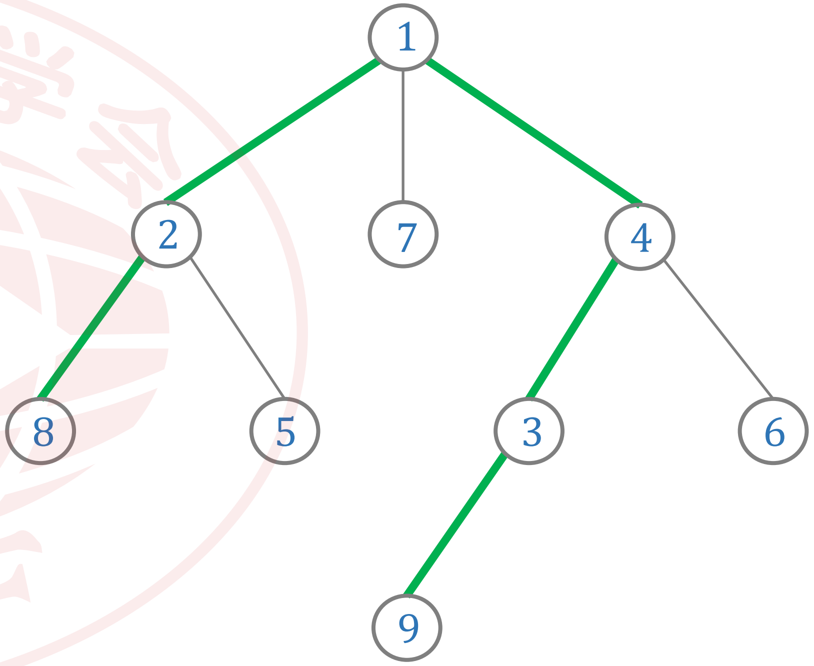
给定一棵树，树中的每一条边都有一个权值，**树中两点之间的距离**定义为连接两点的路径上的边权之和。树中最远的两个节点之间的距离被称为**树的直径**，连接这两点的路径被称为**树的最长链。**

直径既是一个数值概念，也可代指一条路径。



两次 BFS 求树的直径

1. 从任意一个节点 u 出发，通过 BFS(或 DFS)对树进行一次遍历，求出与 u 距离最远的节点 p。
2. 从节点 p 出发，通过 BFS(或 DFS)对树再进行一次遍历，求出与 p 距离最远的节点 q。
3. 从 p 到 q 的路径，记作 $\delta(p,q)$，即为直径。



两次 BFS 求树的直径

证明：**在树上，以任意节点出发所能到达的最远节点，一定是该树的直径的端点之一。**

假设 $\delta(s,t)$ 为直径，而出发节点 y 达到最远的节点为 z（不是 s,t 中的任意一个），路径为 $\delta(y,z)$。可分为两种情况：

1. 当 y 在 $\delta(s,t)$ 上时，这时将 $\delta(y,z)$ 与不与之重合的 $\delta(y,s)$ 拼接，可以得到一条更长的直径，与前提 $\delta(s,t)$ 为直径矛盾。
2. 当 y 不在 $\delta(s,t)$ 上时，又分两种情况：

① 当 $\delta(y,z)$ 横穿 $\delta(s,t)$ 时，与之相交的节点为 x。此时有 $\delta(y,z)=\delta(y,x)+\delta(x,z)$。而此时 $\delta(y,z)>\delta(y,t)$，故可得$\delta(x,z)>\delta(x,t)$，即 $\delta(s,x)+\delta(x,z)>\delta(s,x)+\delta(x,t)$，与前提 $\delta(s,t)$ 为直径矛盾。

② 当 $\delta(y,z)$ 与 $\delta(s,t)$ 不相交时，设 y 到 t 的最短路首先与 $\delta(s,t)$ 相交于 x 点。

由假设可知 $\delta(y,z)>\delta(y,x)+\delta(x,t)$，而 $\delta(y,z)+\delta(y,x)+\delta(x,s)$ 可以组成 $\delta(z,s)$。

而 $\delta(z,s)>\delta(y,x)+\delta(x,t)+\delta(y,x)+\delta(x,s)=\delta(s,t)+2\delta(y,x)$，与前提 $\delta(s,t)$ 为直径矛盾。

完整题面请访问

LuoGu P3629
AcWing 350

巡逻

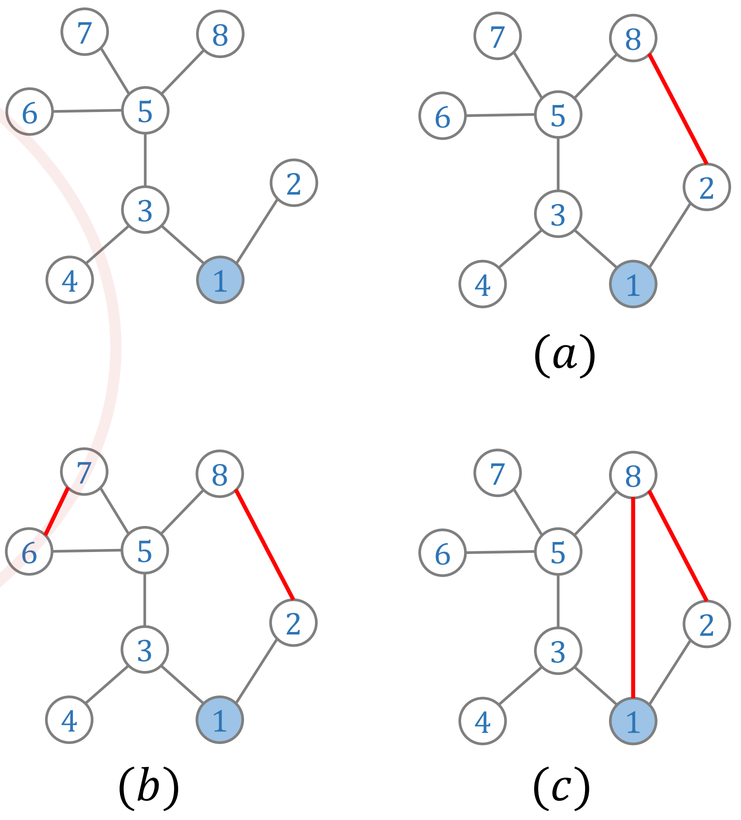
在一个地区有 $n\,(3 \le n \le 10^5)$ 个村庄，编号为 $1,2,\dots,n$。有 $n-1$ 条道路连接着这些村庄，每条道路刚好连接两个村庄，从任何一个村庄，都可以通过这些道路到达其他任一个村庄。每条道路的长度均为 1 个单位。

为保证该地区的安全，巡警车每天都要到所有的道路上巡逻。警察局设在编号为 1 的村庄里，每天巡警车总是从警局出发，最终又回到警局。

为了减少总的巡逻距离，该地区准备在这些村庄之间建立 K 条新的道路，每条新道路可以连接任意两个村庄。两条新道路可以在同一个村庄会合或结束，甚至新道路可以是一个环。因为资金有限，所以 K 只能为 1 或 2。

同时，为了不浪费资金，每天巡警车**必须经过新建的道路正好一次**。

在给定村庄间道路信息和需要新建的道路数的情况下，计算出最佳的新建道路的方案，使得总的巡逻距离最小。



完整题面请访问

巡逻

LuoGu P3629
AcWing 350

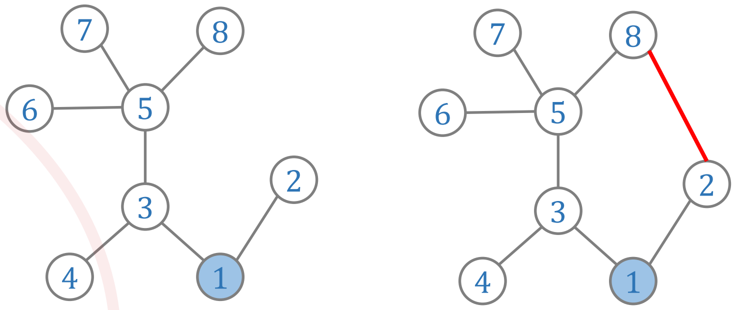
如不建立新道路，从 1 号节点出发，把整颗树上的每条边遍历至少一次，再回到 1 号节点，会恰好经过每条边 2 次，路线总长度为 $2(n-1)$。

如只建立 1 条新道路，且要求必须仅经过 1 次，如 (a) 所示：相当于 $1-2-8-5-3$ 构成环，且环上路径只需经过 1 次。所以，当 $K=1$ 时，只需找到树的最长链，在两端点间加一条新的道路，就能让总的距离最小。若树的直径为 L_1，答案就是 $2(n-1)-L_1+1$。

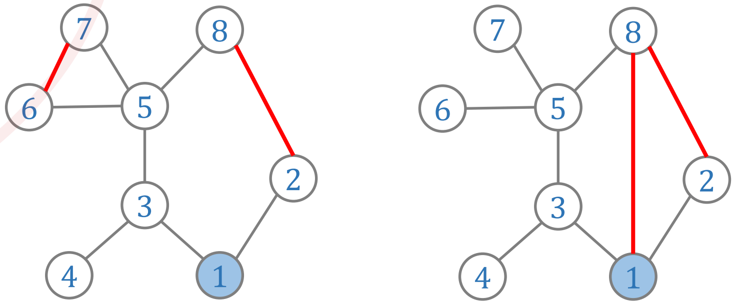
如需建立 2 条道路，将会形成两条环，如 (b) 和 (c) 所示，分为无重叠路径和有重叠路径两种情况，而在有重叠的情况下，这部分路径仍需经过 2 次。

综上可得如下算法：

1. 在最初的树上求直径 L_1，然后将直径上的边权取反；
2. 在树上再求一次直径 L_2；
3. 答案为 $2(n-1)-(L_1-1)-(L_2-1)=2n-L_1-L_2$。



(a)



(b) (c)

如果 L_2 这条直径包含 L_1 取反的部分，就相当于两个环部分重叠。那么，减去 (L_1-1) 后，重叠的部分变成了“只需经过一次”，减掉 (L_2-1) 后，相当于把重叠的部分加回来，变回“需要经过两次”。时间复杂度为 $O(n)$。

树形动态规划求树的直径

设 1 号节点为根，“N 个点 $N-1$ 条边的无向图”就可以看作“有根树”。

设 $D[x]$ 表示从节点 x 出发走向 x 为根的子树，能够到达的最远节点的距离。

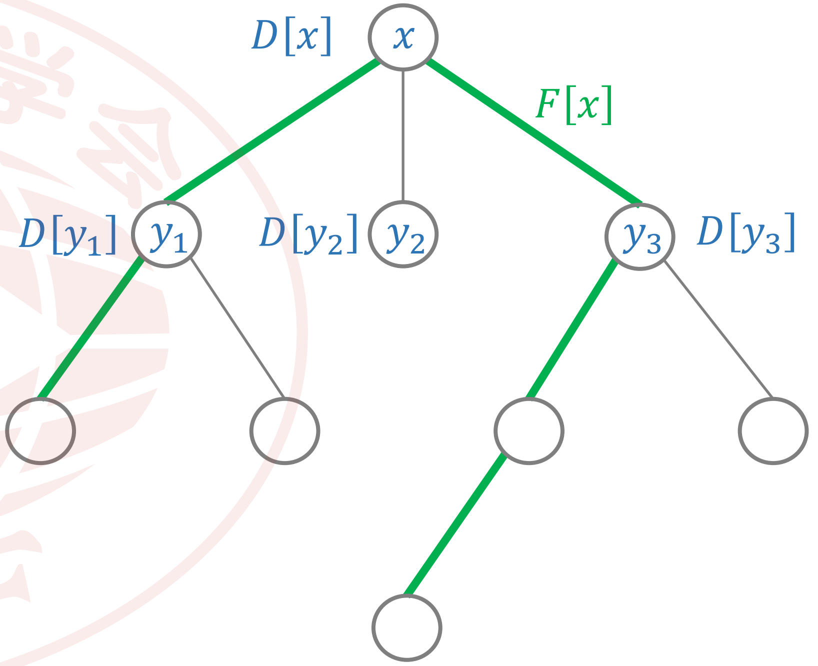
设 x 的子节点为 $y_1, y_2, \cdots, y_t$，$\text{edge}[x,y]$ 表示边权，有

$$D[x] = \max_{1 \le i \le t}\{D[y_i] + \text{edge}(x, y_i)\}$$

考虑对每个节点 x 求出“经过节点 x 的最长链的长度”$F[x]$，整颗树的直径就是 $\max_{1 \le x \le n}\{F[x]\}$。

对于 x 的任意两个节点 y_i 和 y_j（设 $j < i$），

$$F[x] = \max_{1 \le j < i \le t}\{D[y_i] + D[y_j] + \text{edge}(x, y_i) + \text{edge}(x, y_j)\}$$



树形动态规划

在树上设计动态规划算法是，一般就以**节点从深到浅（子树从小到大）**的顺序作为 DP 的“阶段”。

DP 的状态表示中，第一维通常是节点编号（代表以该节点为根的子树）。

大多数时候，采用**递归**的方式实现树形动态规划。

对于每个节点 x，先递归在它的每个子节点上进行 DP，在回溯时，从子节点向节点 x 进行状态转移。

没有上司的舞会

某大学有 N 名职员，编号为 $1 \sim N$。他们的关系就像一颗以校长为根的树，父节点就是子节点的直接上司。每个职员有一个快乐指数 H_i。现在要召开一场周年庆舞会，不过，没有职员愿意和直接上司一起参会。

在满足这个条件的前提下，主办方希望邀请一部分职员参会，使得所有参会职员的快乐指数总和最大，求此最大值。

没有上司的舞会

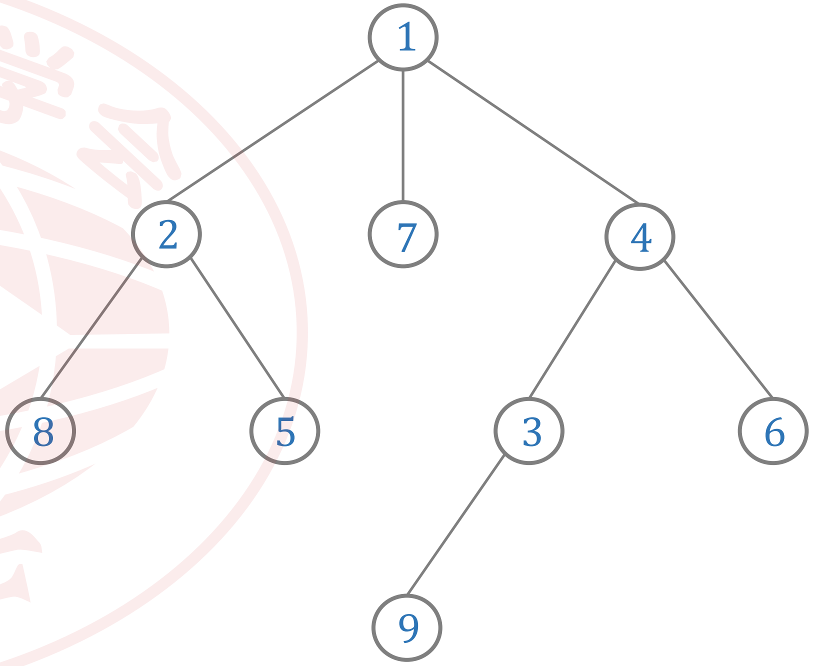
以节点编号（子树的根）作为 DP 状态的第一维。一名职员是否愿意参加只跟他的直接上司是否参加有关，所以在每棵子树递归完成时，保留两个“代表信息”：根节点参加时，整棵子树的最大快乐指数总和，以及根节点不参加时，整棵子树的最大快乐指数总和，就可满足“最优子结构”性质。

设 $F[x,0]$ 表示从以 x 为根的子树中邀请一部分职员参会，并且 x **不参加**舞会时，快乐指数总和的最大值。此时，x 的子节点（直接下属）可以参会，也可以不参会。其中，$Son(x)$ 表示 x 的子节点集合。

$$F[x,0]=\sum_{s\in Son(x)}\max(F[s,0],F[s,1])$$

设 $F[x,1]$ 表示从以 x 为根的子树中邀请一部分职员参会，并且 x **参加**舞会时，快乐指数总和的最大值。此时，x 的所有子节点（直接下属）都不可以参会。

$$F[x,1]=H[x]+\sum_{s\in Son(x)}F[s,0]$$



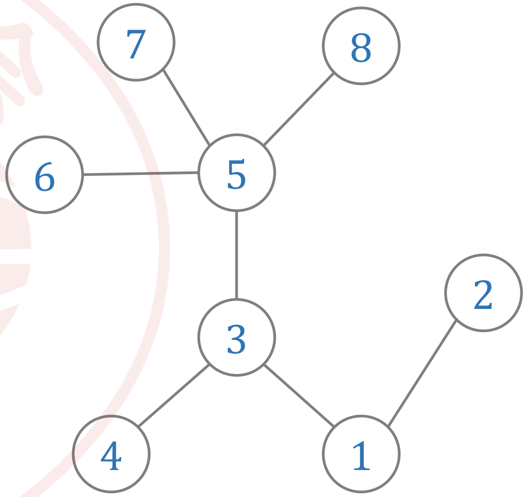
设根节点为 $root$，DP 的目标为 $\max(F[root,0],F[root,1])$，时间复杂度为 $O(N)$。

树形水系

有一树形水系，由 $N-1$ 条河道和 $N(N \le 2\times10^5)$ 个交叉点组成，从交叉点 x 至 y 的河道有容量限制 $c(x,y)$，河道中单位时间流过的水量不超过容量限制。

水系中有一个节点为水源，除此之外，树形水系中度数为 1 的节点都是汇点，水从这些节点流出。除了源点和汇点之外，其他节点不储存水，流入与流出的水量相等。整个水系的流量定义为源点单位时间流出的水量。

在流量不超过河道容量的前提下，求哪个点作为源点时，整个水系的流量最大。

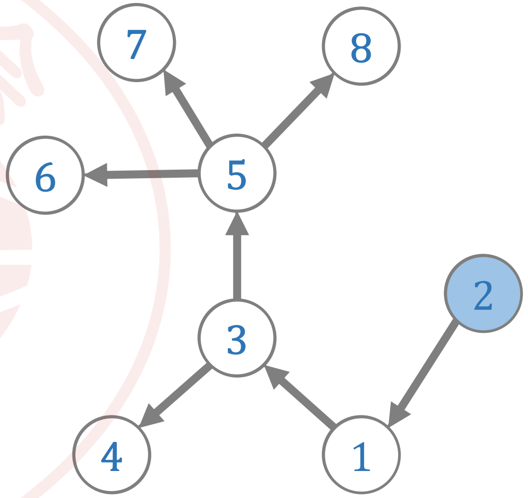


树形水系

对于题中的树形结构，需枚举每个节点作为源点（即树的根节点）转为有根树进行统计。

每个节点将从其父节点获得水源，并流向自己的子节点，每个节点的“流域”就是以该点为根的子树。这符合树形DP 的应用场景——每棵子树都是一个“子问题”。

设 $D_s[x]$ 表示在以 x 为根的子树中，将 x 作为源点，从 x 出发流向子树的流量最大是多少。



$$D_s[x] = \sum_{y \in Son(x)} \begin{cases} \min\left(D_s[y], c(x,y)\right) & y\text{的度数} > 1 \\ c(x,y) & y\text{的度数} = 1 \end{cases}$$

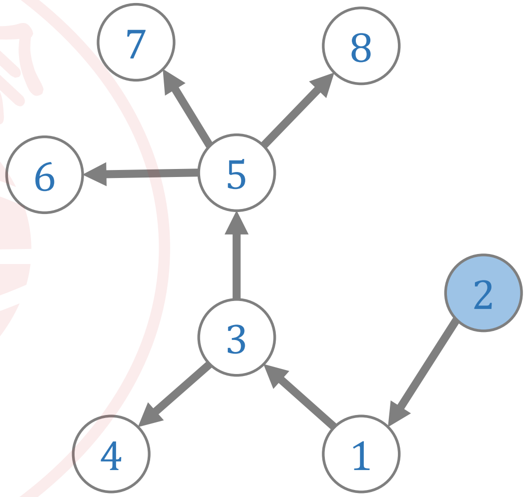
对于枚举的每个源点 s，可用树形 DP 在 $O(N)$ 的时间内求出 D_s 数组，并用 $D_s[s]$ 更新答案。时间复杂度为 $O(N^2)$。

二次扫描与换根法

可以通过如下方式来改进算法：

1. 第一次扫描时，任选一个点为根，在“有根树”上执行一次树形 DP，也就是在回溯时发生的、自底向上的状态转移。
2. 第二次扫描时，从刚才选出的根出发，对整棵树执行一次深度优先遍历，在每次递归前进行自顶向下的推导，计算出“换根”后的解。

用“二次扫描与换根法”代替源点的枚举，可以在 $O(N)$ 的时间内解决此问题。



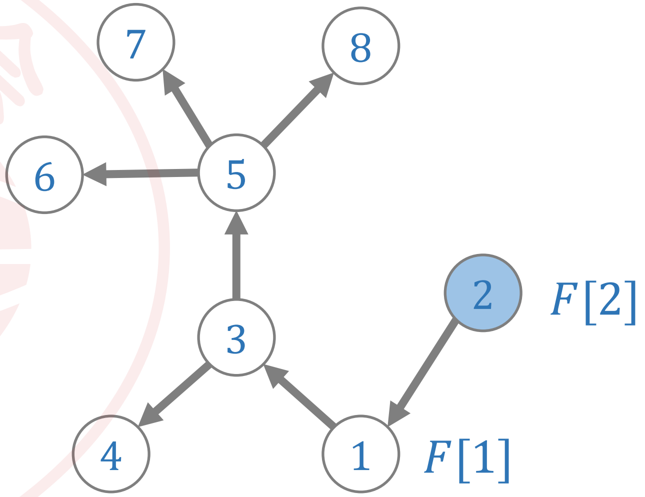
二次扫描与换根法

首先，任选一个节点作为根节点 $root$，进行一次树形 DP，求出 D_{root} 数组，记作 D 数组。

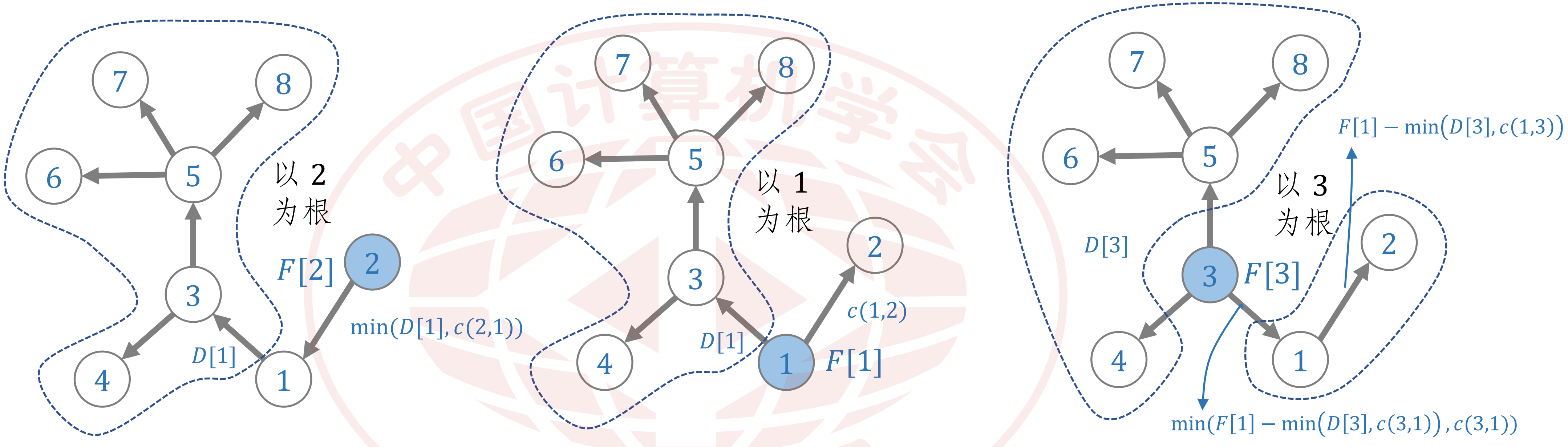
设 $F[x]$ 表示把 x 作为源点，流向**整个水系**，流量最大为多少。对于根节点 $root$，显然 $F[root] = D[root]$。

假设 $F[x]$ 已被正确求出，考虑其子节点 y，$F[y]$ 尚未被计算。$F[y]$ 包含两部分：

1. 从 y 流向以 y 为根的子树的流量，已经计算并保存在 $D[y]$ 中。
2. 从 y 沿着到父节点 x 的河道，进而流向水系中其他部分的流量。



二次扫描与换根法



因为把 x 作为源点的总流量为 $F[x]$，从 x 流向 y 的流量为 $\min(D[y], c(x,y))$，所以从 x 流向除 y 以外其他部分的流量就是二者之差。于是把 y 作为源点，先流到 x，再流向其他部分的流量就是把这个“差”再与 $c(x,y)$ 取最小值后的结果。

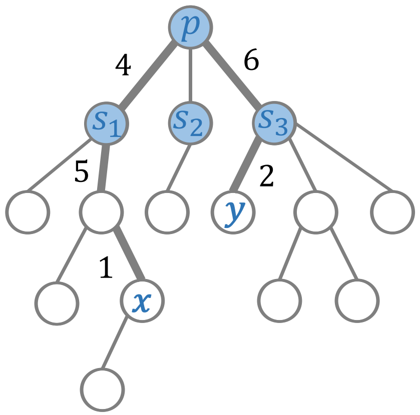
$$F[y] = D[y] + \begin{cases} \min\left(F[x] - \min(D[y], c(x,y)), c(x,y)\right) & x\text{的度数} > 1 \\ c(x,y) & x\text{的度数} = 1 \end{cases}$$

$F[y]$ 就是把源点从 x 换到 y 后流量的计算结果。这是一个自顶向下的递推，通过一次 DFS 即可实现。

点分治

适用场景：在树上两点之间的路径，如果不考虑对其进行修改操作，仅对具有某些限定条件的路径静态地进行统计。

给定一颗有 N 个点的无根树，每条边都有一个权值。树上两个节点 x,y 之间的路径长度就是路径上各条边的权值之和。求长度不超过 K 的路径有多少条。



$b[x]=s_1$
$b[y]=s_3$
$d[x]=10$
$d[y]=8$

若指定节点 p 为根，则对 p 而言，树上的路径可分为两类：

1. 经过根节点 p；
2. 包含于 p 的某一颗子树中（不经过根节点）。

根据分治的思想，对于第 2 类路径，显然可以把 p 的每颗子树作为子问题，递归进行处理。

对于第 1 类路径，可以从根节点 p 分成 “$x\sim p$” 与 “$p\sim y$” 两段。在对树 DFS 过程中，预处理 $d[x]$ 表示节点 x 到根节点 p 的距离；$b[x]$ 表示节点 x 属于根节点 p 的哪一颗子树，特别地，令 $b[p]=p$。

满足要求的第 1 类路径就是满足以下两个条件的点对 (x,y) 的个数：

1. $b[x]\neq b[y]$
2. $d[x]+d[y]\leq k$

点分治

点分治算法流程：

1. 任选一个根节点 p；
2. 从 p 出发进行一次 DFS，求出 d 数组和 b 数组；
3. 在以 p 为根的树中统计符合条件的点对个数；
4. 删除根节点 p，对 p 的每颗子树递归执行 1～4 步。

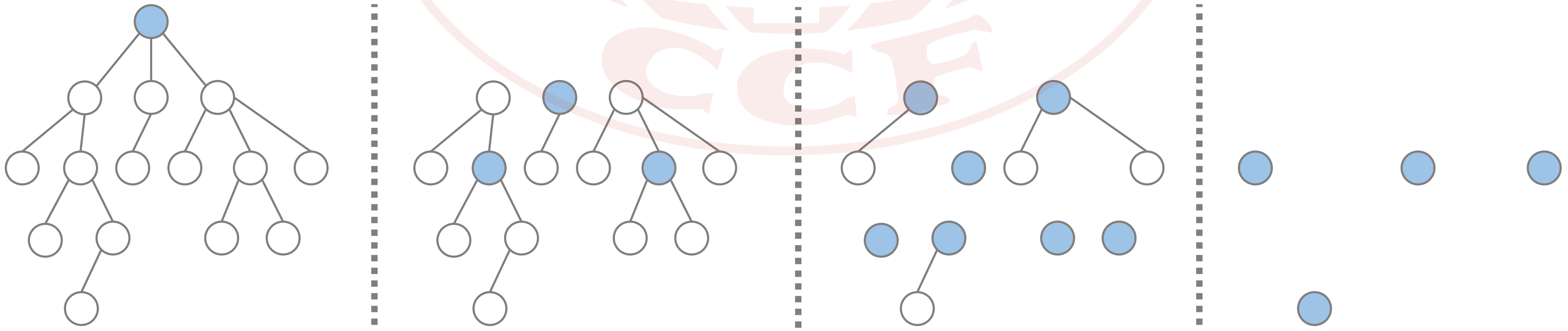
在点分治过程中，每层所有递归过程合计对每个节点处理 1 次。

若递归最深到达第 T 层，整个算法的时间复杂度为 $O(TN\log N)$

如果问题中的树形态为一条链，最坏情况下每次以链的一端为根，那么点分治将递归 N 层，时间复杂度将退化为 $O(N^2\log N)$。

为了避免这种情况，应**每次选择树的重心作为根节点** p。

因为 p 的每颗子树不会超过整棵树大小的一半，点分治只多递归 $O(\log N)$ 层，整个算法的时间复杂度即为 $O(N\log^2 N)$。



前缀和

对于一个序列 A，它的“前缀和” 数列 S 可通过递推计算

$$S[i]=\sum_{j=1}^{i}A[j]$$

区间和（部分和），即序列 A 在 $[l,r]$ 之间的数之和，可通过“前缀和” 相减求得

$$\mathrm{Sum}[l,r]=\sum_{i=l}^{r}A[i]=S[r]-S[l-1]$$

差分

对于一个序列 A，定义基于 A 的差分序列 B：

$$B[1]=A[1],\qquad B[i]=A[i]-A[i-1]\quad(2\le i\le n)$$

前缀和与差分是互逆运算，差分序列的前缀和序列即原序列 A，而前缀和序列的差分序列也是原序列 A。

把序列 A 的区间 $[l,r]$ 加 d（区间修改），其差分序列 B 则变为 B_l+d ，$B_{r+1}-d$ ，其他元素不变化，即将“区间修改”，变为“单（两）点修改”。

树上差分

在“前缀和与差分”中，定义一个序列的前缀和与差分序列，并通过差分技巧，可以把“区间”的增减转化为“左端点加 1，右端点减 1”。

根据“差分序列的前缀和是原序列”这一性质，在树上可以进行类似的简化，其中“区间操作”对应为“路径操作”，“前缀和”对应为“子树和”。

暗之连锁

完整题面请访问
AcWing 352

一个无向图，有 $N(N \le 10^5)$ 个节点和两类边：一类称为“主要边”，共 $N-1$ 条，无向图的任意两个节点之间都存在一条只由主要边构成的路径；另一类称为“附加边”，共 $M(M \le 2\times10^5)$ 条。

现可以选择删除一条“主要边”和一条“附加边”，使得图分为不联通的两部分，求方案数量。

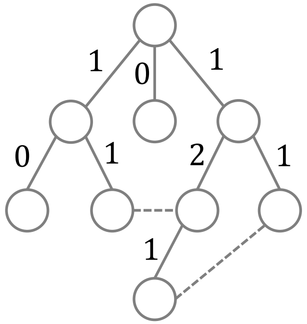
完整题面请访问
AcWing 352

暗之连锁

在图中，“主要边”构成一棵树，“附加边”是“非树边”。把一条附加边 (x,y) 添加到主要边构成的树中，会和“树上 x,y 间的路径”记作 $\delta(x,y)$ 构成环。

如果第一步选择删除 $\delta(x,y)$ 的某条边，第二步就必须删除 (x,y)，才能将图分为不联通的两部分。

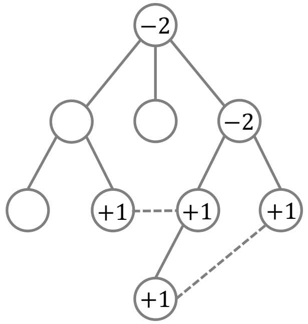
附加边 (x,y) 的作用范围是 $\delta(x,y)$，可视作 (x,y) 将 $\delta(x,y)$ 的每一条边“覆盖了一次”。



主要边被覆盖的次数

1. 为 0 次：可以任意删除一条附加边
2. 为 1 次：删除唯一的一条边
3. 为 2 次及以上：无解

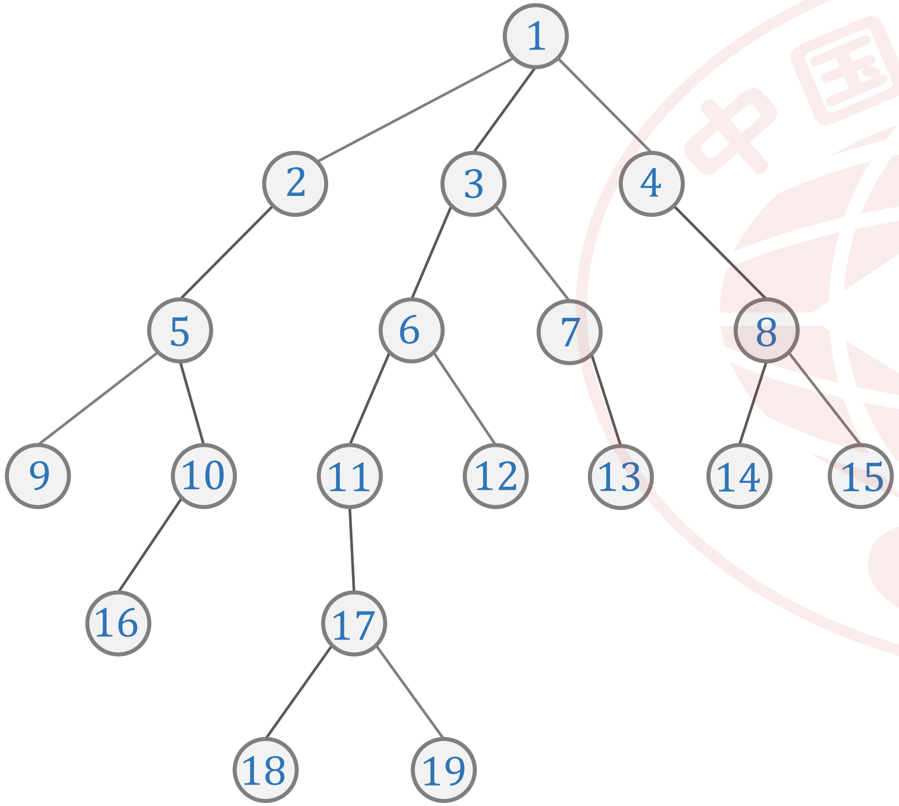
树上差分：树上的每个节点初始权值为 0，对于每条附加边 (x,y)，令节点 x 和 y 的权值均加 1，$\text{LCA}(x,y)$ 的权值减 2。



对路径统计前缀和，即“以 x 为根的子树中各节点的权值之和”，就是 x 与它父节点之间的“主要边”被覆盖的次数。

时间复杂度为 $O(N+M)$。

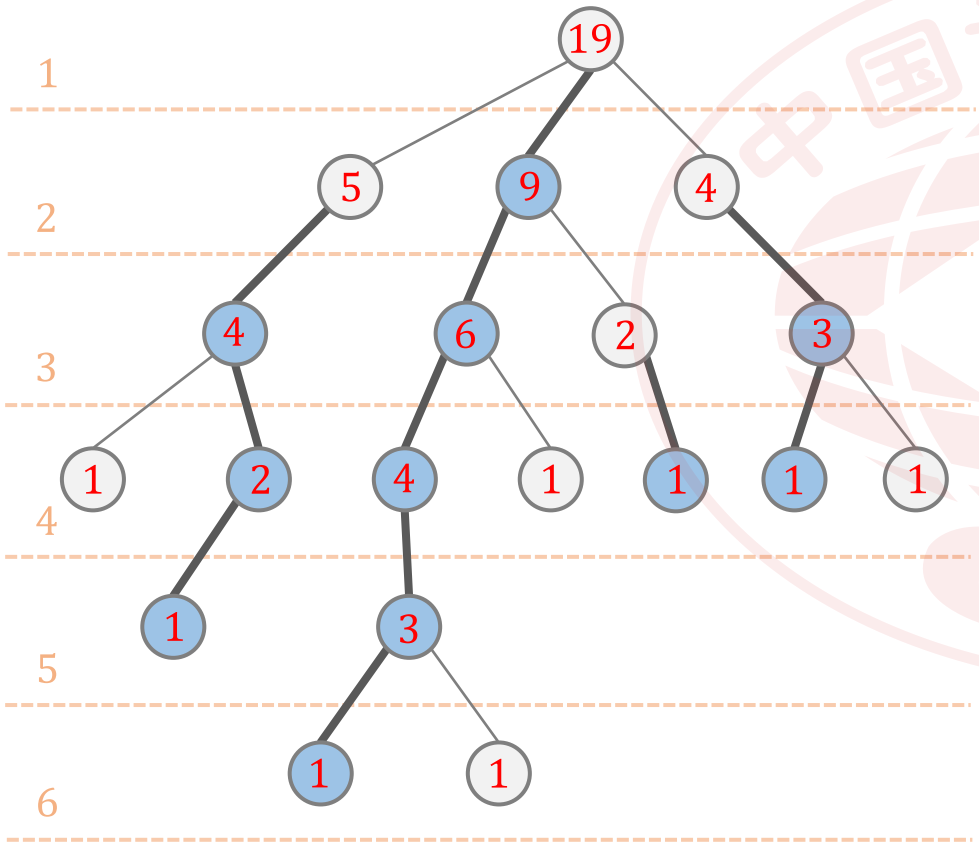
树链剖分



将整棵树**剖分为若干条链**，使它组合成**线性结构**，然后用其他的数据结构维护信息。

树链剖分有多种形式，如**重链剖分**，**长链剖分**和用于 Link/Cut Tree 的**实链剖分**。大多数情况下，树链剖分都指**重链剖分**。

树链剖分



19 子树大小

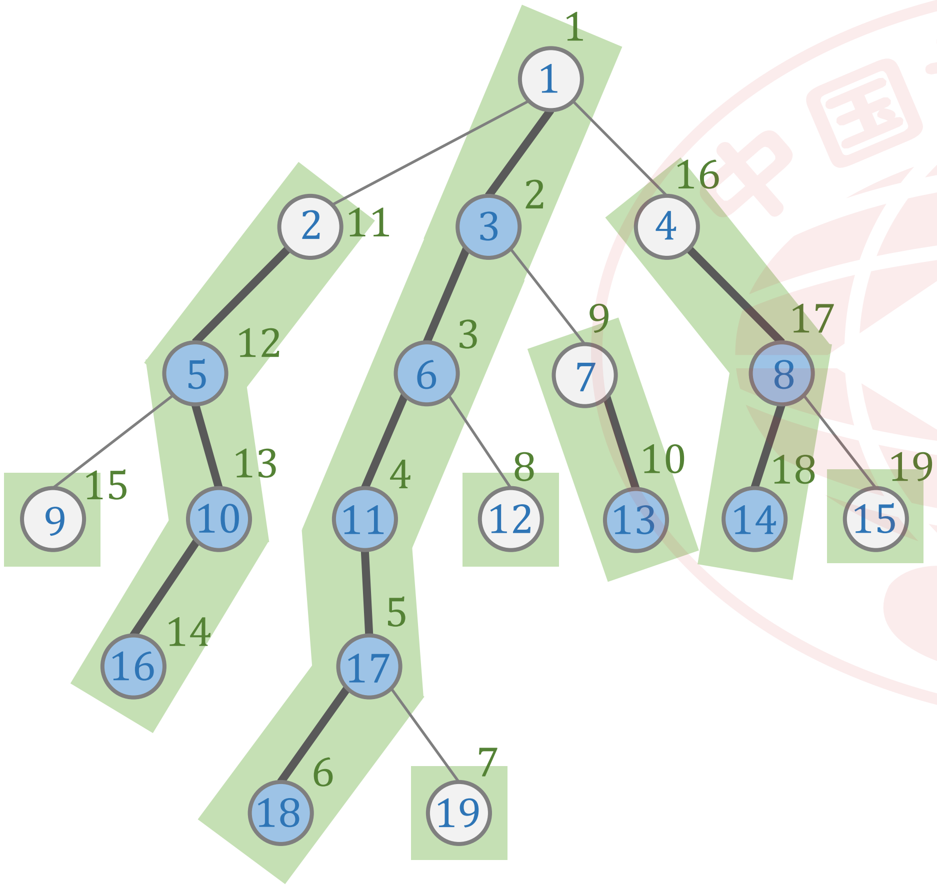
重子节点 表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。

轻子节点 表示剩余的所有子结点

重边 从这个结点到重子节点的边

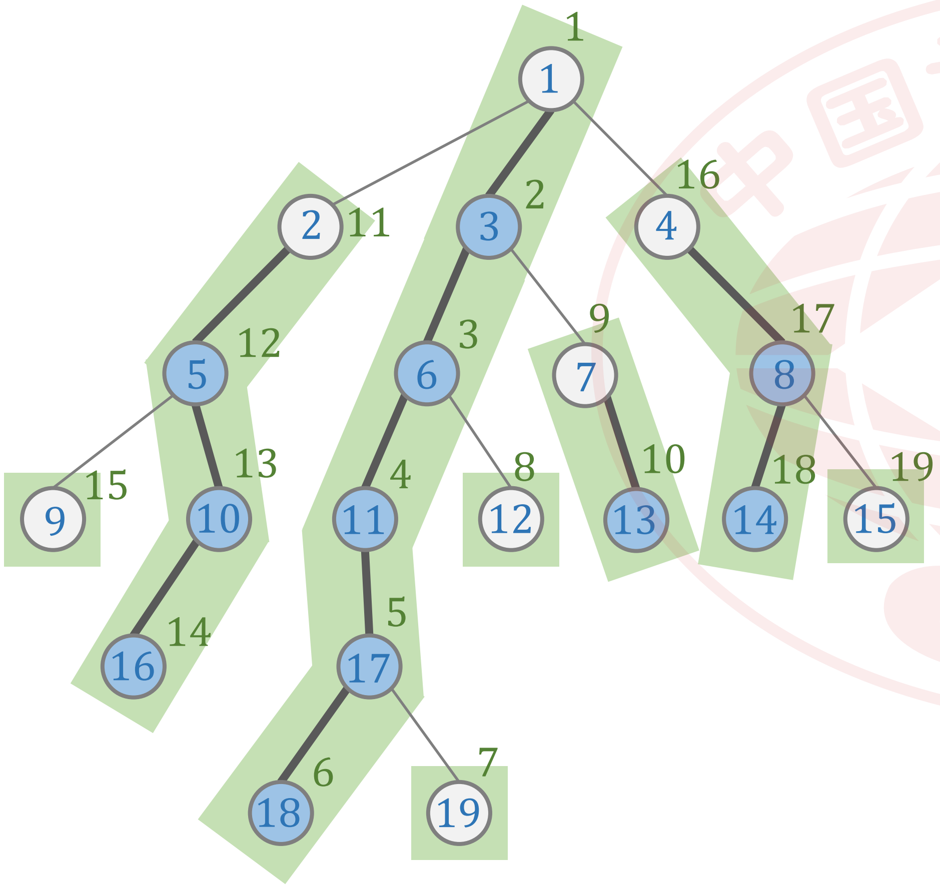
轻边 到其他轻子节点的边

树链剖分



19	节点编号	
	重子节点	表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。
	轻子节点	表示剩余的所有子结点
	重边	从这个结点到重子节点的边
	轻边	到其他轻子节点的边
	重链	由若干条首尾衔接的重边构成，把落单的结点也当作重链，那么整棵树就被剖分成若干条重链。
19	dfn	在每条重链上的节点的 dfn 是连续的

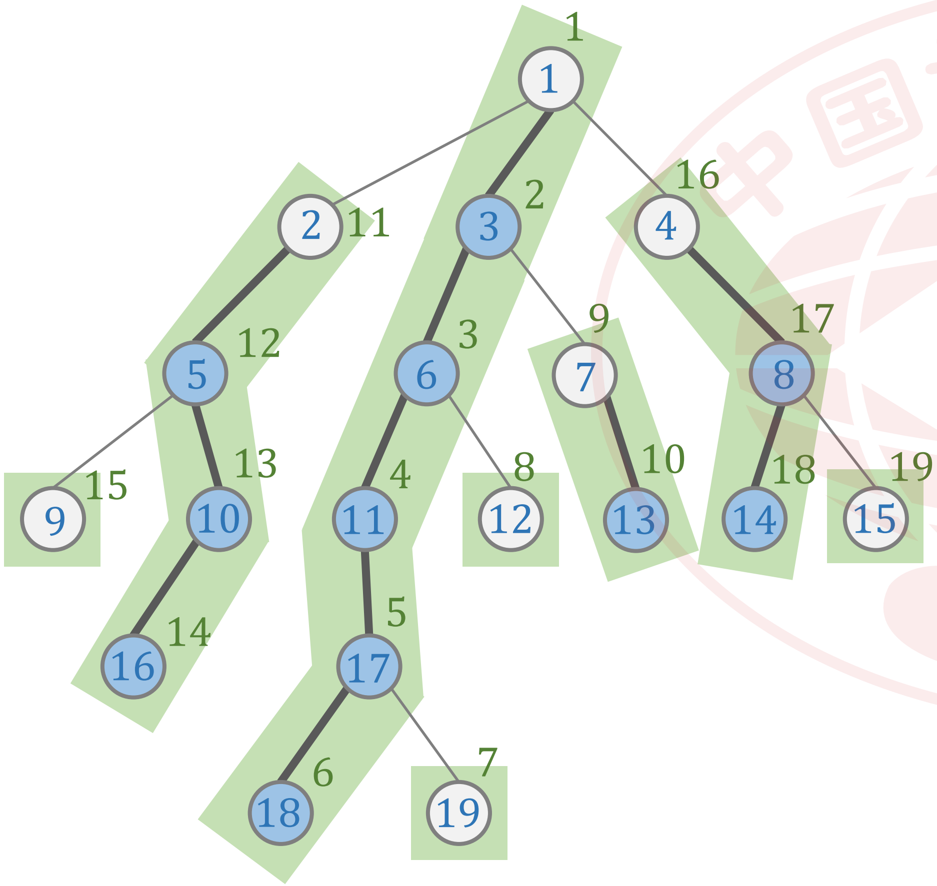
树链剖分



重链剖分的性质

- 树上每个节点都属于且仅属于一条重链。
- 重链开头的结点不一定是重子节点。
- 所有的重链将整棵树完全剖分。
- 在剖分时，重边优先遍历，重链内的 dfn 是连续的，一颗子树内的 dfn 是连续的。按 dfn 排序后的序列即为剖分后的链。
- 当向下经过一条轻边时，所在子树的大小至少会除以 2。因此，对于树上的任意一条路径，把它拆分成从 LCA 分别向两边往下走，分别最多走 $O(\log n)$ 次，因此，树上的每条路径都可以被拆分成不超过 $O(\log n)$ 条重链。

树链剖分



树剖的实现分两个 DFS 的过程：

第一个 DFS 记录每个结点的父节点(*father*)、深度(*deep*)、子树大小(*size*)、重子节点(*hson*)，伪代码如下：

$\mathbf{TREE-BULID}(u, dep)$

$\quad u.hson \leftarrow 0$

$\quad u.hson.size \leftarrow 0$

$\quad u.deep \leftarrow dep$

$\quad u.size \leftarrow 1$

$\quad$ **for** each u's son v

$\qquad u.size \leftarrow u.size + \mathbf{TREE-BULID}(v, dep+1)$

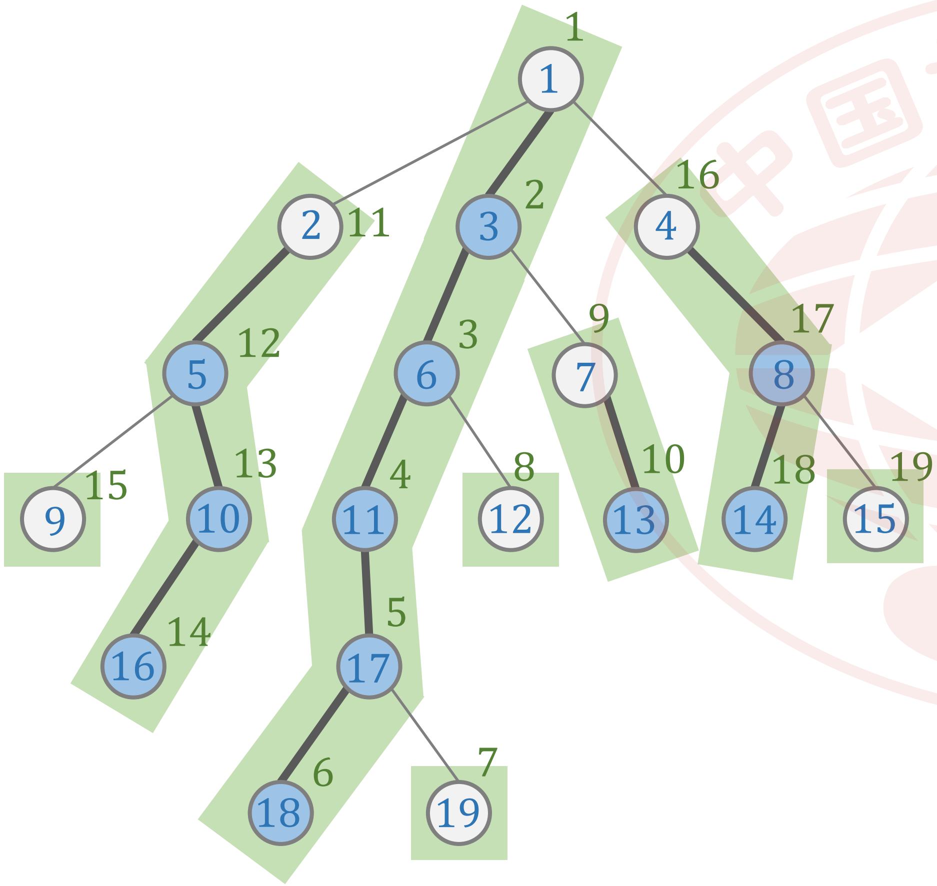
$\qquad v.father \leftarrow u$

$\qquad$ **if** $v.size > u.hson.size$

$\qquad\quad u.hson \leftarrow v$

$\quad$ **return** $u.size$

树链剖分



树剖的实现分两个 DFS 的过程：

第二个 DFS 记录所在链的链顶(top，应初始化为结点本身)、重边优先遍历时的 DFS 序(dfn)、DFS 序对应的节点编号($rank$)

TREE − DECOMPOSITION(u, top)

$u.top \leftarrow top$

$cnt \leftarrow cnt + 1$

$u.dfn \leftarrow cnt$

$rank(cnt) \leftarrow u$

if $u.hson$ is not 0

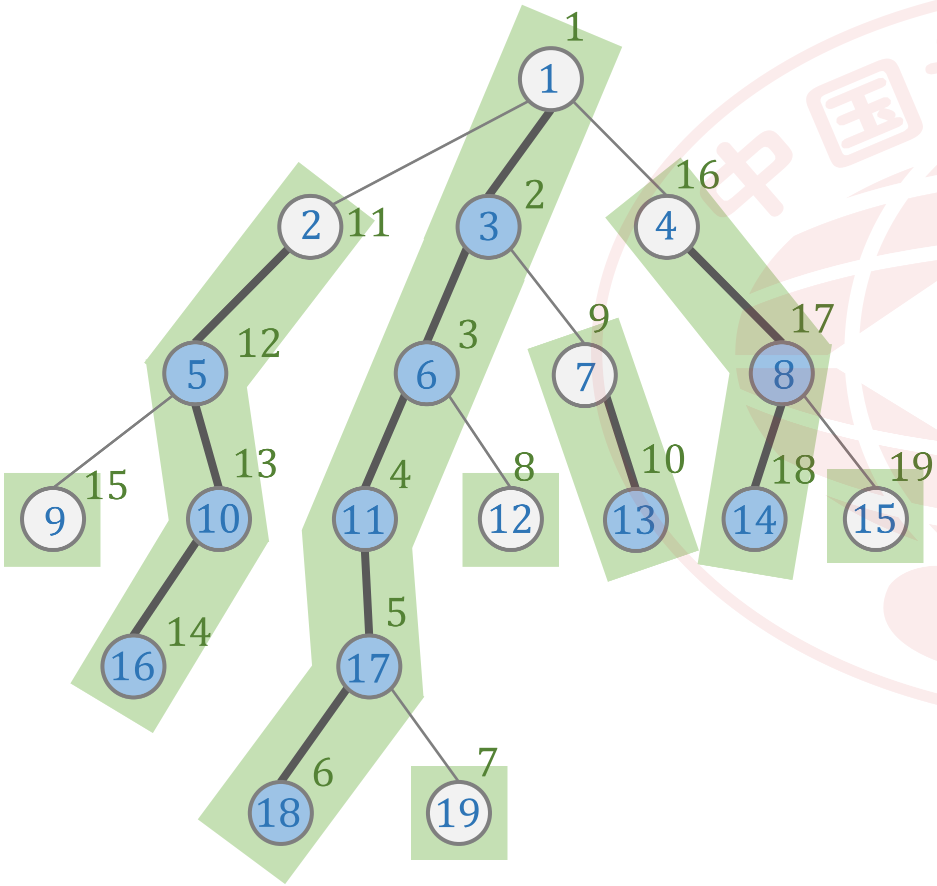
  TREE − DECOMPOSITION $(u.hson, top)$

  for each u's son v

    if v is not $u.hson$

      TREE − DECOMPOSITION (v, v)

树链剖分



由于重链剖分的特点，可以方便地用一些维护序列的数据结构（如线段树）来维护树上路径的信息。如：

- 修改 **树上两点之间的路径上** 所有点的值。
- 查询 **树上两点之间的路径上** 节点权值的 **和/极值/其它（在序列上可以用数据结构维护，便于合并的信息）**。

除了配合数据结构来维护树上路径信息，树链剖分还可以用来 （且常数较小）地求 LCA。

树的统计

完整题面请访问
LibreOJ 10138
LuoGu P2590
AcWing 1278

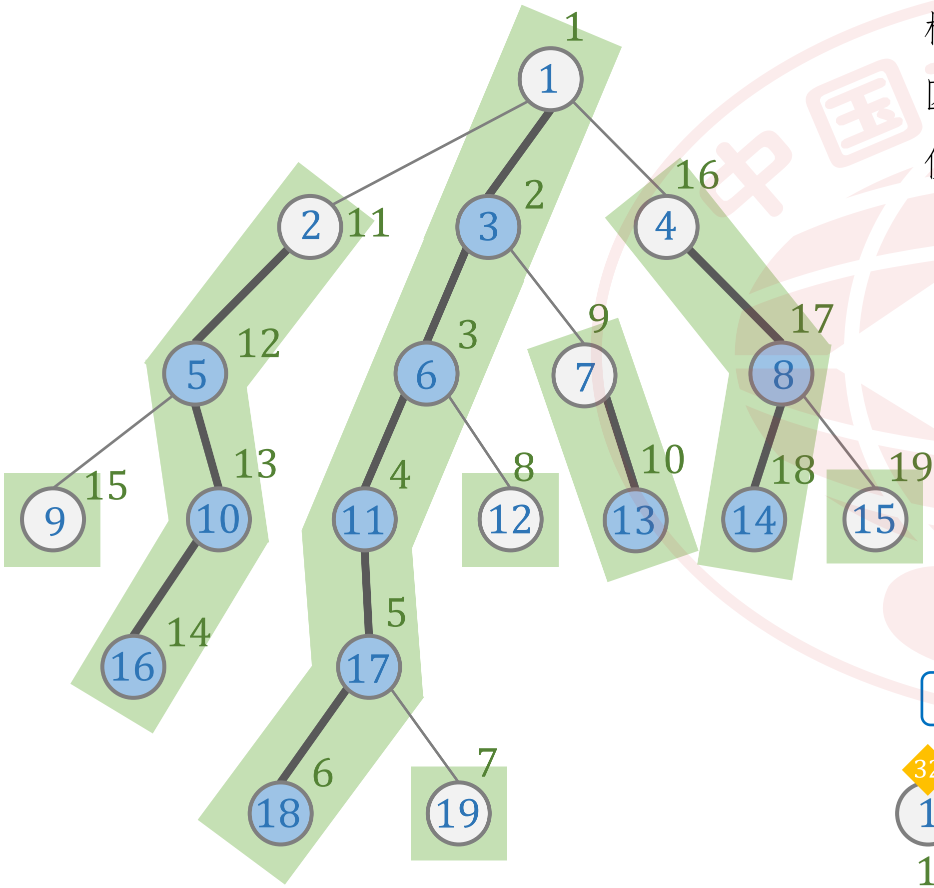
一棵树上有 n 个节点，编号分别为 1 到 n，每个节点都有一个权值 w。

我们将以下面的形式来要求你对这棵树完成一些操作：

1. CHANGE u t ：把结点 u 的权值改为 t。
2. QMAX u v：询问从点 u 到点 v 的路径上的节点的最大权值。
3. QSUM u v：询问从点 u 到点 v 的路径上的节点的权值和。

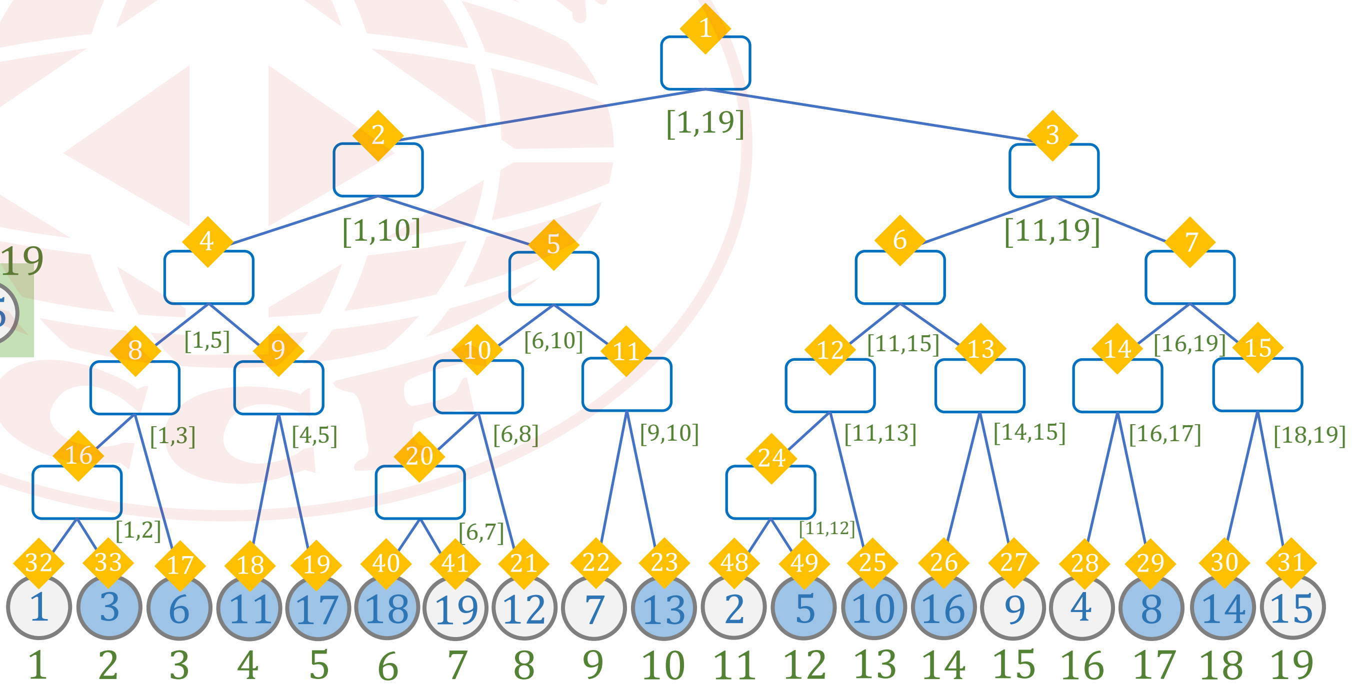
完整题面请访问
LibreOJ 10138
LuoGu P2590
AcWing 1278

树的统计



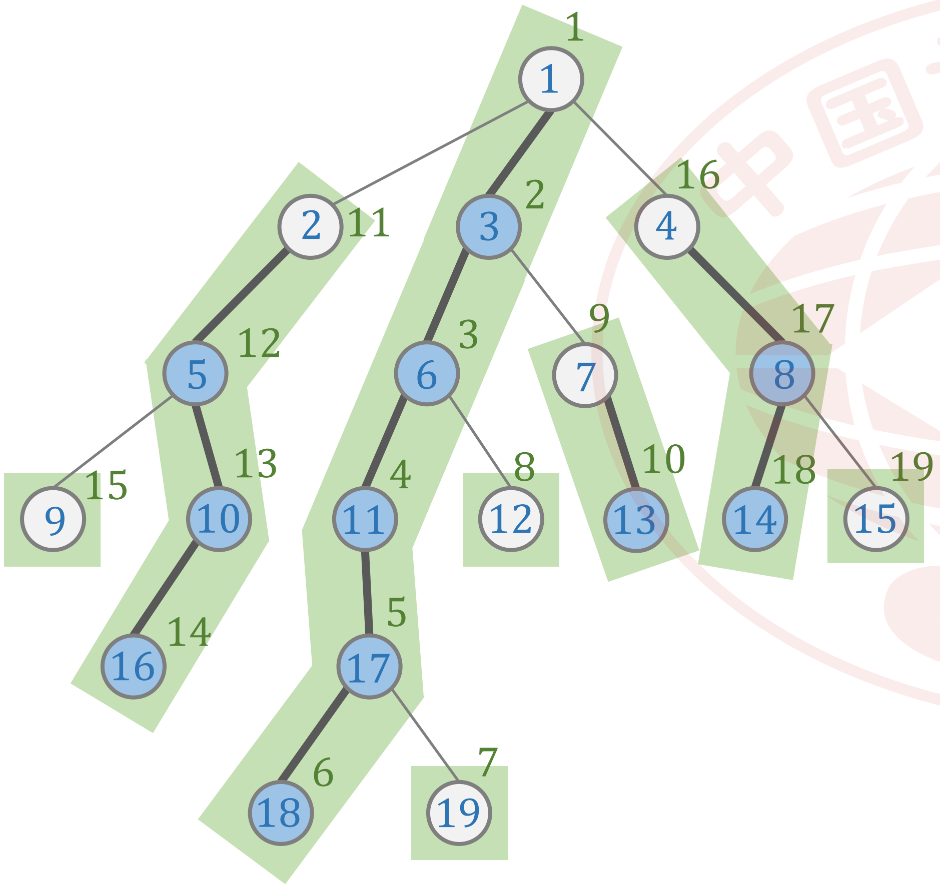
根据题意，需要维护三种操作：单点修改；
区间查询最大值；区间查询和。
使用线段树维护整棵树的链。

1 线段树节点编号
[1,19] 线段树区间节点
18 19 树上节点/线段树叶节点



完整题面请访问
LibreOJ 10138
LuoGu P2590
AcWing 1278

树的统计



如何将**树上的两点间路径上查询最值和统计求和**，转移到**线段树上的询问区间最值和区间求和**？

考虑使用类似倍增求 LCA 的技巧：

- 使两个节点，在所在链上向上跳跃，
- 如果当前节点在重链上，向上跳到重链顶端，
- 否则，则在其顶端向上跳一步，跳到重链上。
- 直到两个节点跳到位于同一条链上。

在向上跳的过程中，沿途维护线段树上的区间信息。

对于每个询问，最多经过 $O(\log n)$ 条重链，每条重链上线段树的复杂度为 $O(\log n)$，因此总时间复杂度为 $O(n\log n + q\log^2 n)$。实际上重链个数很难达到 $O(\log n)$，所以树剖在一般情况下常数较小。

软件包管理器

完整题面请访问
LibreOJ 2130
LuoGu P2146
AcWing 918

设计一个软件包管理器并解决软件包之间的依赖问题：

如果软件包 a 依赖软件包 b，那么安装软件包 a 以前，必须先安装软件包 b。同时，如果想要卸载软件包 b，则必须卸载软件包 a。

现已获得所有的软件包之间的依赖关系。而且由于之前的工作，除 0 号软件包以外，在管理器当中的软件包都会依赖一个且仅一个软件包，而 0 号软件包不依赖任何一个软件包，且依赖关系不存在环。

现要为软件包管理器写一个依赖解决程序：根据反馈，用户希望在安装和卸载某个软件包时，快速地知道这个操作实际上会改变多少个软件包的安装状态（即安装操作会安装多少个未安装的软件包，或卸载操作会卸载多少个已安装的软件包）。

注意，安装一个已安装的软件包，或卸载一个未安装的软件包，都不会改变任何软件包的安装状态，即在此情况下，改变安装状态的软件包数为 0。

软件包管理器

完整题面请访问
LibreOJ 2130
LuoGu P2146
AcWing 918

结合样例很容易理解，有两种操作：

1. 安装第 x 号软件包，等效于将根节点到 x 节点都进行安装，即树上路径修改。
2. 卸载第 x 号软件包，等效于将 x 为根的子树全部卸载，即树上子树修改。

同样使用线段树维护即可。

雨天的尾巴

完整题面请访问
LuoGu P4556
AcWing 353

有 $N(N \le 10^5)$ 个点，形成树状结构。

有 $M(M \le 10^5)$次发放操作，每次选择两个点 x,y，对 x 到 y 的路径上（包括 x,y）的每个点发放一个 $z(z \le 10^9)$ 类型的物品。

求完成所有操作后，每个点存放最多的是哪种类型的物品。

雨天的尾巴

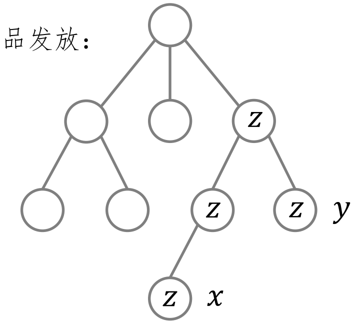
完整题面请访问
LuoGu P4556
AcWing 353

朴素算法：对每个节点 x 建立计数数组 $c[x][1\sim M]$（M 为对 z 的离散化后数量）。

对于每次发放操作，对 $\delta(x,y)$ 上的每个点 p，令 $c[p][z]+1$。最后遍历数组 c 得到答案，空间和时间复杂度均为 $O(NM)$。

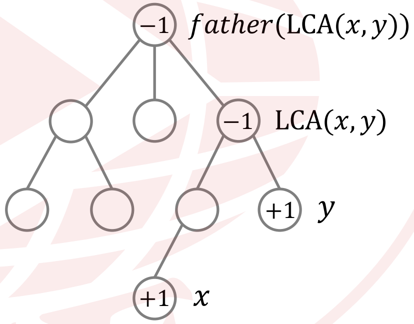
树上差分：$\delta(x,y)$ 上的所有点被类型 z 覆盖了一次。

物品发放：



设 b 为差分数组，发放操作转化为：

- 节点 x,y 处产生 z;
- 节点 $\mathrm{LCA}(x,y)$ 处 z 消失;
- 节点 $father(\mathrm{LCA}(x,y))$ 处 z 消失。



计数数组 c 等于 b 的**子树和**。

若 x 的子节点为 $s_1,s_2,\cdots,s_k$，

则 $c[x]$ 是 $c[s_1],c[s_2],\cdots,c[s_k]$

和 $b[x]$ 这些数组对应位置相加

以后得到的数组。

为节省空间并快速的相加两个计数数组，采用**线段树合并**算法。

对于每个点 x，建立一颗**动态开点的权值线段树**代替 $b[x]$，支持“修改某个位置”，维护区间最大值以及最大值的位置。

执行完 M 次发放操作后，对树进行 DFS，采用“线段树合并”统计子树和。

时间和空间复杂度均为 $O((N+M)\log(N+M))$

动态开点的线段树

为了降低空间复杂度，可以**不建立**出整棵线段树的结构。

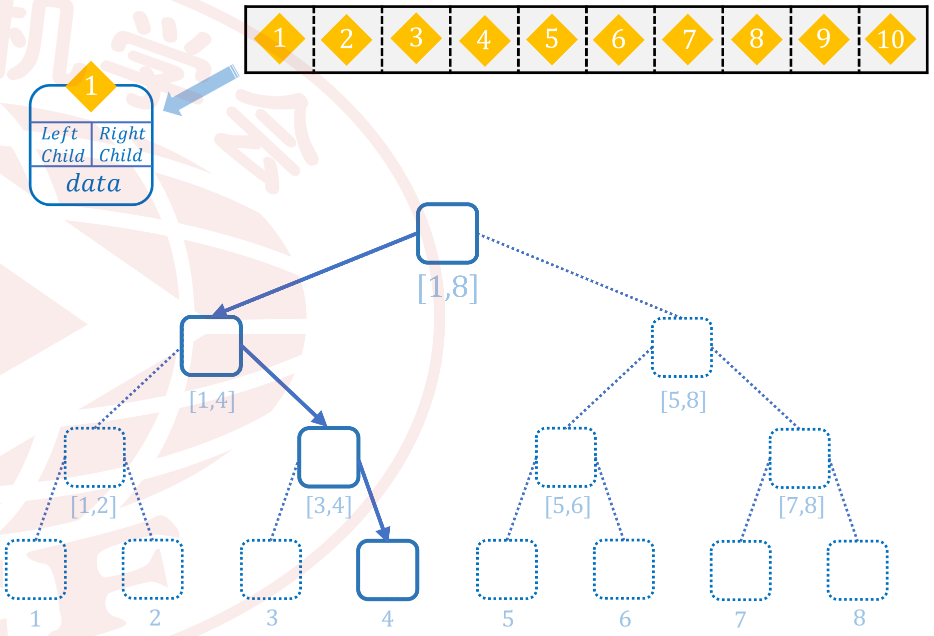
线段树中所有的**节点**都保存在结构体数组中。

与传统完全二叉树父子节点的 2 倍编号规则不同，每个节点使用变量记录左右子节点的编号。

在最初只建立一个根节点，代表整个区间，当需要访问线段树的某棵子树（某个子区间）时，再建立代表这个子区间的节点。

同时，它也不再保存每个节点代表的区间，而是在每次递归访问的过程中作为参数传递。

一颗维护值域 $[1,n]$ 的动态开点线段树在经历 m 次单点操作后，节点数量的规模为 $O(m\log n)$，最终至多有 $2n-1$ 个节点。



线段树合并

如果有若干棵线段树，它们都维护相同的值域 $[1,n]$，那么它们**对各个子区间的划分显然是一致的**。

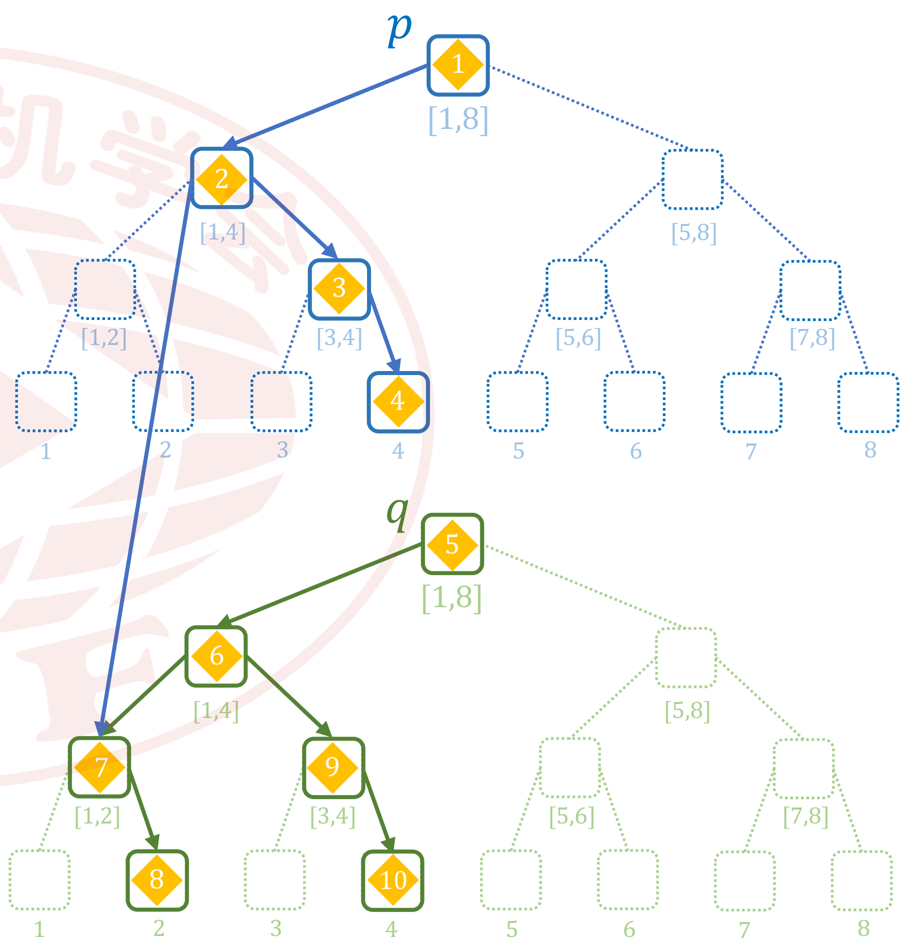
假设有 m 次单点修改操作，每次操作都在某一棵线段树上执行。

当所有操作完成后，将这些线段树**对应位置上的值相加**，同时维护区间最大值。这称为**线段树合并算法**。

合并两颗线段树，用两个指针 p,q 从两个根节点出发，以递归的方式同步遍历两棵线段树，p,q 指向的总是代表相同的子区间。

1. 如果 p,q 之一为空，则以非空的那个作为合并后的节点；
2. 如果 p,q 都不为空，则递归合并两个左子树和右子树，然后删除节点 q，以 p 为合并后的节点，自底向上更新最值信息。若已经达到叶节点，则直接把两个最值相加即可。

若线段树合并过程中发生递归，必定导致 p,q 之一被删除。因此完成合并后，合并操作次数不超过所有节点总数加一。合并时间复杂度为 $O(m\log n)$，与完成所有单点修改操作的时间复杂度相同。



天天爱跑步

完整题面请访问
LibreOJ 2359
LuoGu P1600
AcWing 354

这个游戏的地图可以看作一棵包含 n 个节点和 $n-1$ 条边的树，任意两个节点存在一条路径互相可达。树上节点的编号是 $1 \sim n$ 之间的连续正整数。

现在有 m 个玩家，第 i 个玩家的起点为 S_i，终点为 T_i。每天打卡任务开始时，所有玩家在第 0 秒同时从自己的起点出发，以每秒跑一条边的速度，不间断地沿着最短路径向着自己的终点跑去，跑到终点后该玩家就算完成了打卡任务（因为地图是一棵树，所以每个人的路径是唯一的）。

每个节点上都有一个观察员。在节点 j 的观察员会选择在第 W_j 秒观察玩家，一个玩家能被这个观察员观察到当且仅当该玩家在第 W_j 秒也正好到达了节点 j。

注意：一个玩家到达自己的终点后，该玩家就会结束游戏，他不能等待一段时间后再被观察员观察到。即对于把节点 j 作为终点的玩家：若他在第 W_j 秒前到达终点，则在节点 j 的观察员不能观察到该玩家；若他正好在第 W_j 秒到达终点，则在节点 j 的观察员可以观察到这个玩家。

请计算每个观察员会观察到多少人？

完整题面请访问
LibreOJ 2359
LuoGu P1600
AcWing 354

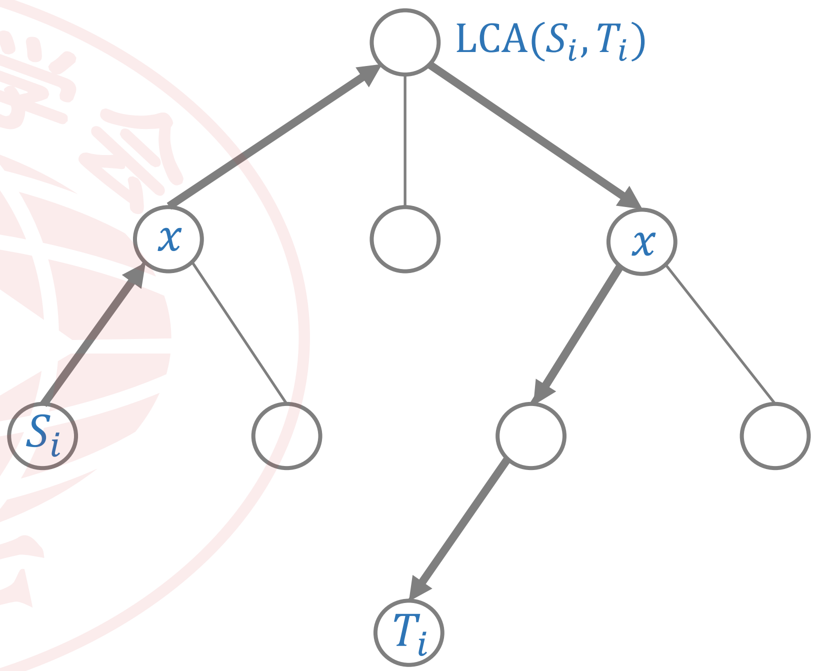
天天爱跑步

每个玩家跑步的路线可以拆成“上升”和“下降”两段，即 $[S_i, \mathrm{LCA}(S_i,T_i)]$ 和 $(\mathrm{LCA}(S_i,T_i),T_i]$。

位于节点 x 的观察员能观察到第 i 个玩家，当且仅当满足以下两个条件之一：

1. 点 x 处于 $\delta(S_i, \mathrm{LCA}(S_i,T_i))$ 上，并且满足 $d[S_i]-d[x]=W[x]$；
2. 点 x 处于 $\delta(\mathrm{LCA}(S_i,T_i),T_i)$ 上（不含 $\mathrm{LCA}(S_i,T_i)$ 这个端点），并且满足 $d[S_i]+d[x]-2\times d[\mathrm{LCA}(S_i,T_i)]=W[x]$。

这两个条件代表玩家从 S_i 跑到 x 所用的时间，等于观察员出现的时间 $W[x]$。



中国计算机学会 China Computer Federation

完整题面请访问
LibreOJ 2359
LuoGu P1600
AcWing 354

天天爱跑步

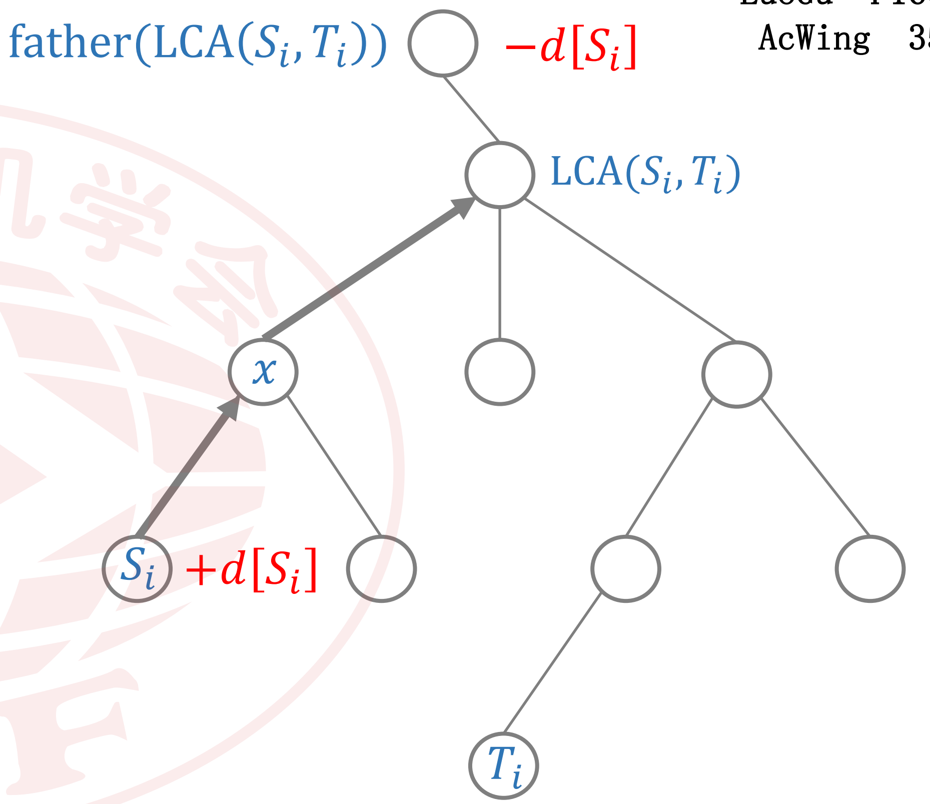
以上升路径为例，

等式 $d[S_i]-d[x]=W[x]$ 移项为 $d[S_i]=W[x]+d[x]$

转化为以下模型：

有 m 个玩家，其中第 i 个玩家给 $\delta(S_i, \mathrm{LCA}(S_i,T_i))$ 上的每个节点增加一个类型为 $d[S_i]$ 的物品。最终求每个点 x 处类型为 $W[x]+d[x]$ 的物品有多少个。

通过树上差分转化为：起点 S_i 物品增加 $d[S_i]$ 和终点 $\mathrm{father}(\mathrm{LCA}(S_i,T_i))$ 物品 $d[S_i]$ 消失。



天天爱跑步

完整题面请访问
LibreOJ 2359
LuoGu P1600
AcWing 354

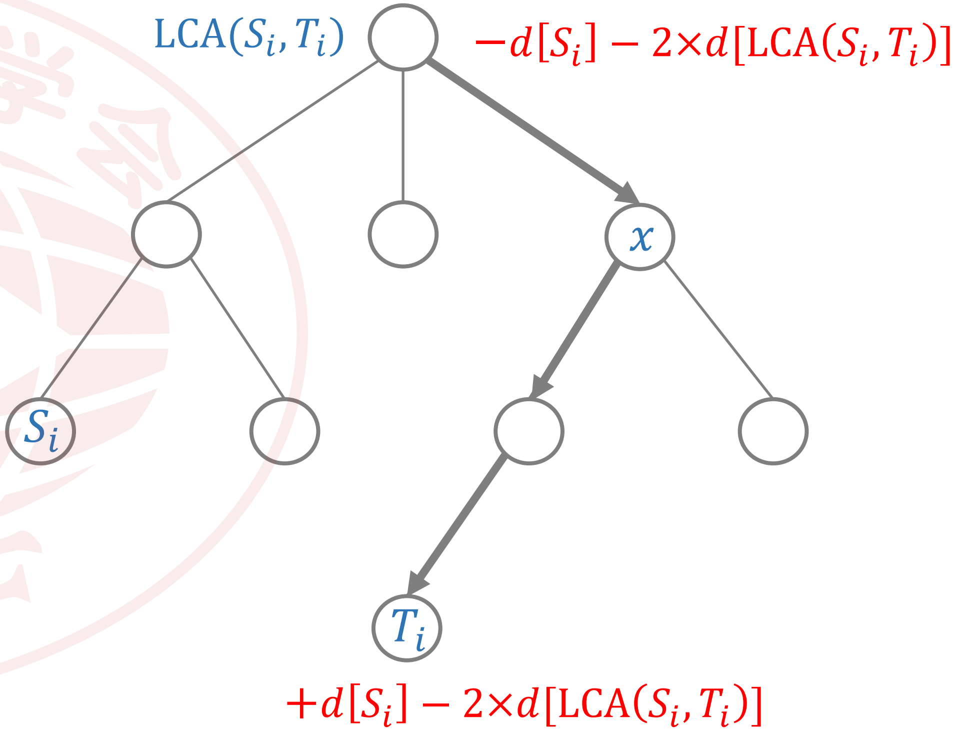
“下降”路径：

$d[S_i]-2\times d[\mathrm{LCA}(S_i,T_i)]=W[x]-d[x]$

通过树上差分转化为：物品 $d[S_i]-2\times d[\mathrm{LCA}(S_i,T_i)]$ 在点 T_i 产生，在点 $\mathrm{LCA}(S_i,T_i)$ 处消失，最后求每个点 x 处类型为 $W[x]-d[x]$ 的物品数量。与“上升”路径得到的结果相加，就是点 x 的观察员能观察到的玩家总数。

注意：此时物品类型可能是负数，需要对计数数组的下标范围进行平移或离散化。

在树上每个点处使用动态开点线段树维护计数数组，用线段树合并算法计算子树和，即可得到答案。



天天爱跑步

完整题面请访问
LibreOJ 2359
LuoGu P1600
AcWing 354

如果是计算区间最值，线段树是较为适合的数据结构，而本题仅需计算求和。

可以在树上每个节点处建立 4 个 vector，分别记录上升、下降路径的增加和消失物品编号。

通过统计“子树递归和回溯之间的差值”来实现计算，算法如下：

1. 扫描 m 个玩家，在每个玩家活动路径的 4 个端点的对应 vector 上记录“产生”和“消失”的物品编号；
2. 建立全局数组 c，对每种类型的物品进行计数，初值全为 0；
3. 对整棵树进行 DFS：
   ① 递归进入每个点 x 时，用局部变量 val_1 记录 $c[W[x]+d[x]]$，val_2 记录 $c[W[x]-d[x]]$；
   ② 递归遍历 x 的所有子树，在子树中更新全局数组 c。
   ③ 扫描点 x 的 4 个 vector，在 c 中执行修改（按照物品类型进行增减）；
   ④ 从x 回溯之前，用新的 $c[W[x]+d[x]]$ 减去 val_1，加上 $c[W[x]-d[x]]$ 减去 val_2 的值，就是“子树和”，即点x 处类型为 $W[x]+d[x]$ 和 $W[x]-d[x]$ 的物品数量。

“遍历以 x 为根的子树过程中累加的值”等于“遍历完成时的值”与“遍历开始时的值”之差，也是树上差分的一种形式。

参考资料

[NOI大纲]

[《算法竞赛进阶指南》]

[OI Wiki/图论/树上问题]

中国计算机学会
China Computer Federation

T H A N K S

谢谢