实验舱
青少年编程
走近科学 走进名校

提高算法班

简单树上问题

Mas

树的直径

树上任意两节点之间最长的简单路径即为树的直径

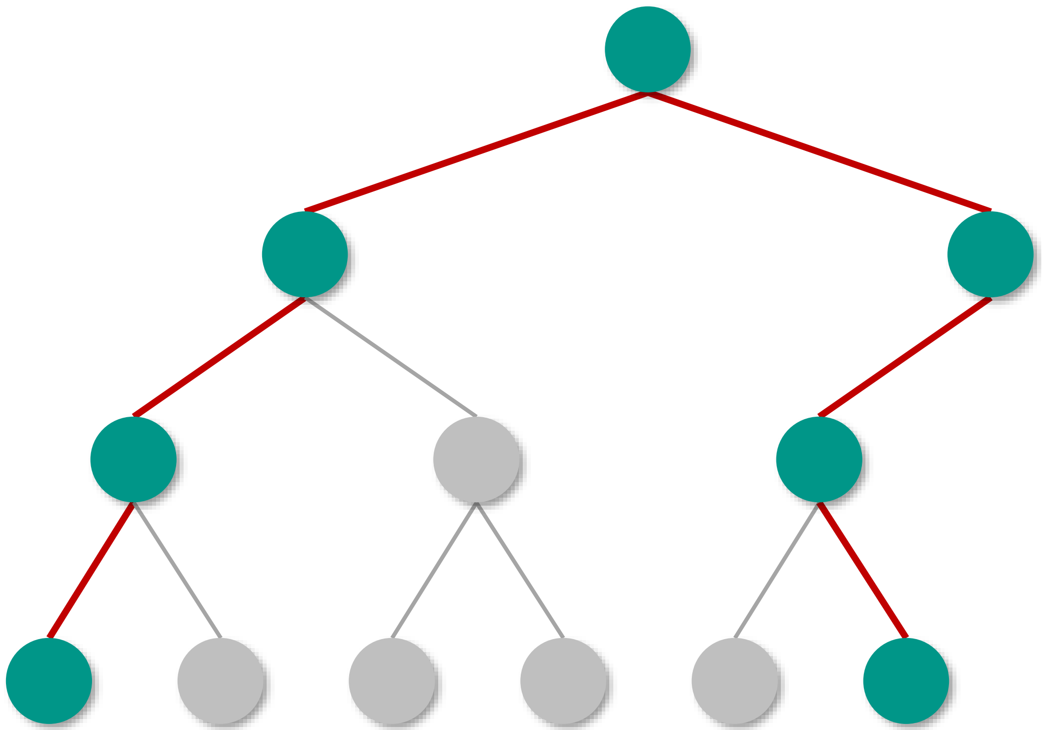
一棵树可以有多条直径,它们的长度相等

树的直径有两种常用求法,分别是 两次 DFS/BFS 以及 树形DP

其中两次 DFS/BFS 仅适用于非负边权树

两次 DFS/BFS 实现

- 从 **任意** 一点 P 出发,通过 DFS/BFS 寻找离它最远的点 Q
- 再次从点 Q 出发,通过 DFS/BFS 寻找离它最远的 W
- 直径即为 WQ 距离



时间复杂度 $O(|V| + |E|)$

在一棵非负边权的树上,从任意节点 P 开始进行一次 DFS/BFS , 到达的距离其最远的节点 Q 必为直径的一端

如何证明?

树的直径

若点 P 在直径上

根据的定义 Q 必定是直径的一个端点

若 Q 不是，则 $PQ + PW$ 能组成一条更长的简单路径，与定义矛盾

若点 P 不在直径上

设直径为 AB

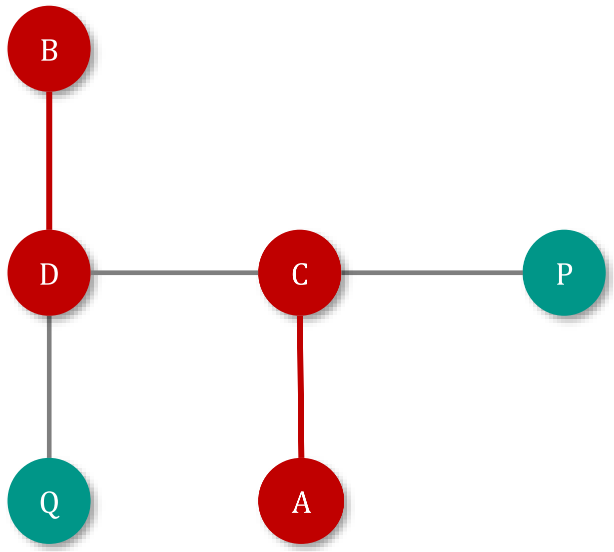
若 Q 不在 AB 上,有以下两种情况:

- AB 与 PQ 有交点 C

根据上述 DFS/BFS 规则有

$$PC + CD + DQ > PC + CD + DB \Rightarrow CD + DQ > CD + DB$$

$$\Rightarrow AC + CD + DQ > AC + CD + DB$$



树的直径

即 $AQ > AB$, 不符合直径定义,矛盾

- AB 与 PQ 无交点

M 为 AB 上任意一点 ,N 为 PQ 上任意一点

根据上述 DFS/BFS 规则有

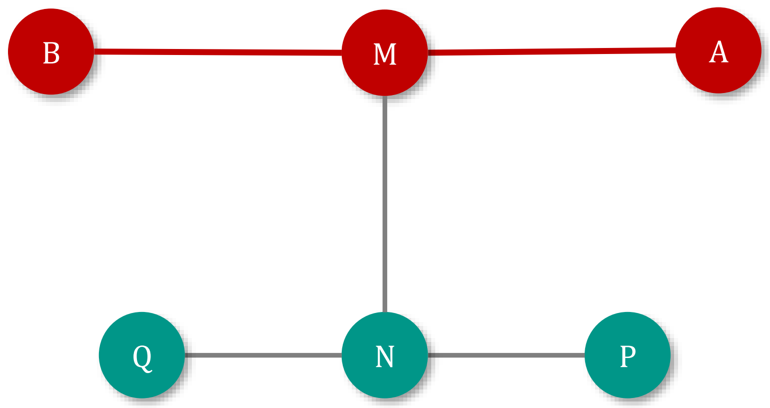
$$PN + NQ > PN + NM + MB$$

$$\Rightarrow NQ > NM + MB$$

$$\Rightarrow NQ + MN + MA > NM + MB + MN + AM$$

即 $PQ > AB + 2NM$

不符合直径定义 , 矛盾



当树上距离存在负边权 , 该结论并不成立

#2416、树的直径

题目描述

对于一棵树，其直径被定义为相距最远的两个点之间的距离

现在给出一棵 n 个节点的树,你需要求出这棵树的直径

输入格式

输入第一行包含一个整数 n 表示树的节点数

接下来 $n-1$ 行每行两个整数 x,y 表示一条树边 (x,y)

输出格式

输出一行一个整数表示树的直径

数据范围

对于 20% 的数据，$n \le 300$

对于 50% 的数据，$n \le 5000$

对于 100% 的数据，$n \le 10^6$

样例输入

```
7
1 2
2 3
1 4
4 5
4 6
5 7
```

样例输出

```
5
```

树上差分

树上差分可以理解为对树上的某一段路径进行差分操作(自底向上)

若对树上路径进行频繁操作，并询问某条边或者某个点在经过操作后的值时,可运用树上差分思想优化

点差分

多次操作使 u 到 v 间的简单路径经过点的次数加 1，最后询问格点经过次数

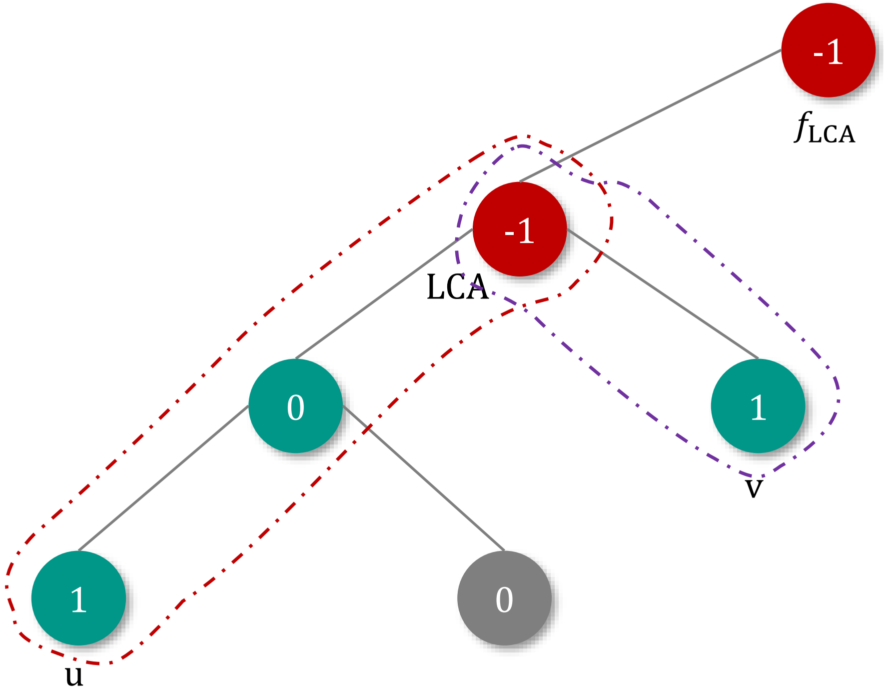
令 d_u为 节点 u 的自底向上的差分数组

只需令

$$d_u \leftarrow d_u + 1 \qquad d_v \leftarrow d_v + 1$$

$$d_{\text{LCA}} \leftarrow d_{\text{LCA}} - 1 \qquad d_{f_{\text{LCA}}} \leftarrow d_{f_{\text{LCA}}} - 1$$

DFS 回溯时累加 d 数组即可还原出每个点经过的次数



树上边差分

边差分

多次操作使 u 到 v 间的简单路径经过边的次数加 1，最后询问格点经过次数

将边经过次数回缩给边下方的端点

令 d_u为 节点 u 的自底向上的差分数组

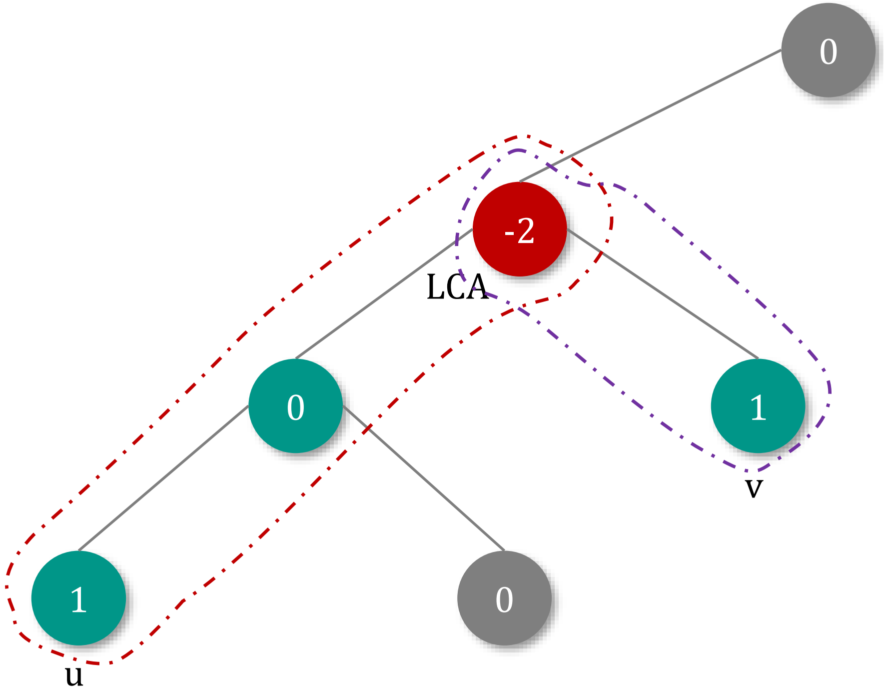
路径上的点需在 LCA 处抵消两倍贡献

只需令

$$d_u \leftarrow d_u + 1 \qquad d_v \leftarrow d_v + 1$$

$$d_{LCA} \leftarrow d_{LCA} - 2$$

DFS 回溯时累加 d 数组即可还原出各边经过的次数



#720、闇の連鎖

题目描述

$Dark$ 是一张无向图,图中有 N 个节点和两类边,一类边被称为主要边,而另一类被称为附加边

$Dark$ 有 $N-1$ 条主要边,并且 $Dark$ 的任意两个节点之间都存在一条只由主要边构成的路径

另外 $Dark$ 还有 M 条附加边

你的任务是把 $Dark$ 斩为不连通的两部分

一开始 $Dark$ 的附加边都处于无敌状态,你只能选择一条主要边切断

一旦你切断了一条主要边, $Dark$ 就会进入防御模式,主要边会变为无敌的而附加边可以被切断

但是你的能力只能再切断 $Dark$ 的一条附加边

现在你想要知道,一共有多少种方案可以击败 $Dark$

注意,就算你第一步切断主要边之后就已经把 $Dark$ 斩为两截,你也需要切断一条附加边才算击败了 $Dark$

数据范围与提示

对于 20% 的数据, $1 \le N, M \le 100$

对于 100% 的数据, $1 \le N \le 10^5, 1 \le M \le 2 \times 10^5$

数据保证答案不超过 $2^{31}-1$

输入格式

第一行包含两个整数 N 和 M

之后 $N-1$ 行,每行包括两个整数 A 和 B,表示 A 和 B 之间有一条主要边

之后 M 行以同样的格式给出附加边

输出格式

输出一个整数表示答案

样例输入

```
4 1
1 2
2 3
1 4
3 4
```

样例输出

```
3
```

#720、闇の連鎖

不难看出主要边构成一棵树，附加边可看作非树边

考虑删除一条主要边 $u \leftrightarrow v$ 以满足条件(不妨认为 $\mathrm{dep}_u > \mathrm{dep}_v$)

若不考虑附加边，以 v 为根的子树已经与上方节点断开

接下来考虑 v 为根的子树内存在的附加边

- 不存在附加边

  那么方案数 $+m$，即删除 $u \leftrightarrow v$ 再 删除任一条附加边皆可

- 存在一条附加边

  那么方案数 $+1$，即删除 $u \leftrightarrow v$ 再 删除唯一的附加边

- 若其简单路径上存在至少两条附加边

  那么方案数 $+0$，无法满足条件

#720、闇の連鎖

令主要边边权为 断开其还需断开的附加边数量

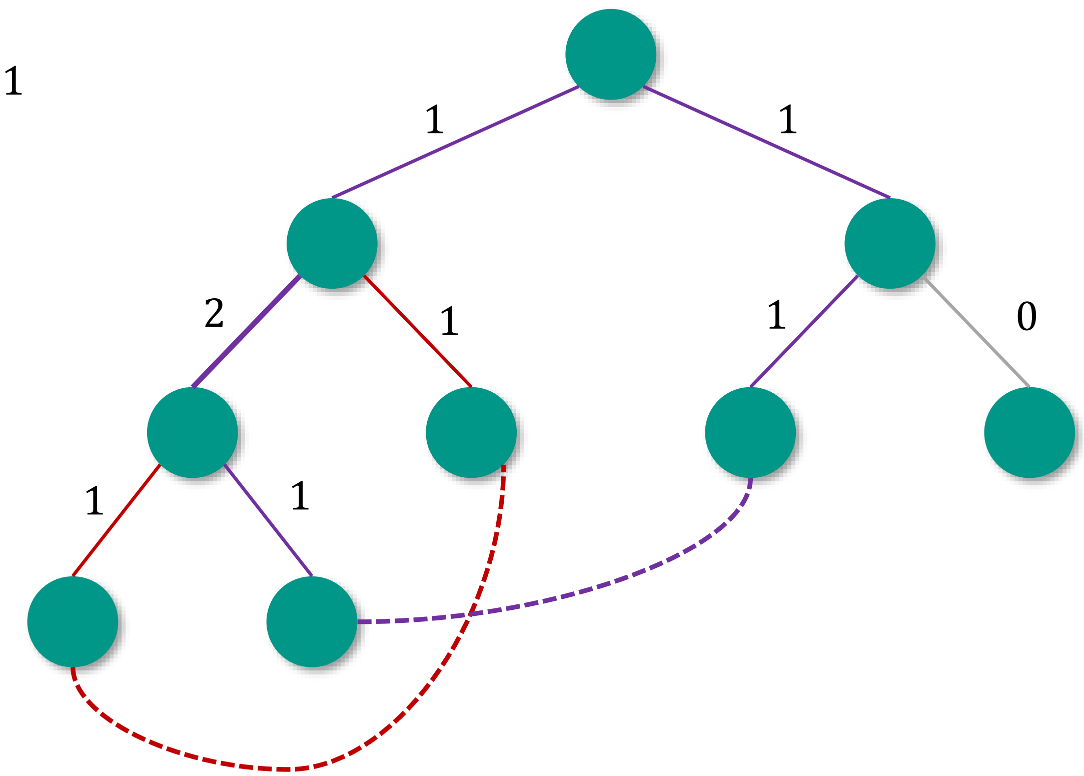
对于一条附加边 $u \leftrightarrow v$ 其可令 u, v 间简单路径上的主要边边权 $+1$

考虑树上边差分维护主要边的边权

考虑所有附加边，最终 DFS 求出主要边边权

对于条主要边 ，按照上述情况分类讨论即可

时间复杂度 $O(n + m \log n)$



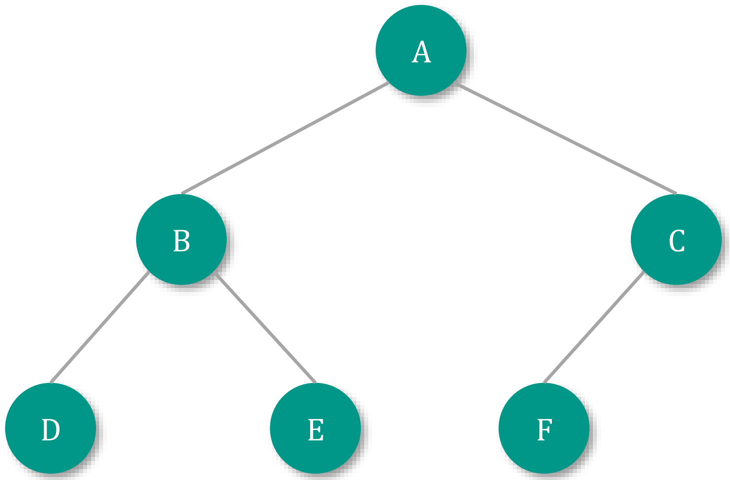
DFS序列

DFS 序列是指 DFS 调用过程中访问的节点编号的序列

各点进入时的时间戳称作 DFS 序

右图树的 DFS 序列为

1	2	3	4	5	6
A	B	D	E	C	F



DFS 序列中 i 的子树节点 DFS 序必然必然是连续的

DFS 序可以把一棵树区间化,即可以求出每个节点的管辖区间

可将树上子树修改,改为线性区间修改

进而可以使用 树状数组 或 线段树维护

#724、祖孙询问

题目描述

已知一棵 n 个节点的有根树

有 m 个询问,每个询问给出了一对节点的编号 x 和 y,询问 x 与 y 的祖孙关系

输入格式

输入第一行包括一个整数 n 表示节点个数

接下来 n 行每行一对整数对 a 和 b 表示 a 和 b 之间有连边。如果 b 是 -1,那么 a 就是树的根

第 $n+2$ 行是一个整数 m 表示询问个数

接下来 m 行,每行两个正整数 x 和 y,表示一个询问

输出格式

对于每一个询问
若 x 是 y 的祖先则输出 1
若 y 是 x 的祖先则输出 2
否则输出 0

数据范围与提示

对于 30% 的数据,$1 \le n, m \le 10^3$

对于 100% 的数据,$1 \le n, m \le 4 \times 10^4$,每个节点的编号都不超过 4×10^4

- 若 u 是 v 的祖先

  有 $in_u \le in_v \le out_u$
- 若 v 是 u 的祖先

  有 $in_v \le in_u \le out_v$
- 否则

  输出 0

只需一次 $O(n)$ 预处理

单次询问 $O(1)$

总时间复杂度 $O(n+m)$

#947、DFS 序 2

题目描述

这是一道模板题

给一棵有根树,这棵树由编号为 $1 \sim N$ 的 N 个结点组成

根结点的编号为 R。每个结点都有一个权值,结点 i 的权值为 v_i

接下来有 M 组操作,操作分为两类:

- 1 a x ,表示将结点 a 的子树上所有结点的权值增加 x .
- 2 a ,表示求结点 a 的子树上所有结点的权值之和

输入格式

第一行有三个整数 N, M 和 R

第二行有 N 个整数,第 i 个整数表示 v_i

在接下来的 $N-1$ 行中,每行两个整数,表示一条边

在接下来的 M 行中,每行一组操作

输出格式

对于每组 2 a 操作,输出一个整数,表示「以结点 a 为根的子树」上所有结点的权值之和

DFS 遍历树，求出 DFS 序

进入时时间戳 in_u

离开时时间戳 out_u

对与节点 u 的子树对应的区间则为 $\text{in}_u \sim \text{out}_u$

使用 树状数组 / 线段树 维护区间信息即可

时间复杂度 $O(m \log n)$

数据范围与提示

$1 \leq N, M \leq 10^6$, $1 \leq R \leq N$, $-10^6 \leq v_i, x \leq 10^6$.

#948、DFS 序 3

题目描述

不保证无快读的程序能过,请务必使用快读

给一棵有根树,这棵树由编号为 $1 \sim N$ 的 N 个结点组成

根结点的编号为 R

每个结点都有一个权值,结点 i 的权值为 v_i

接下来有 M 组操作,操作分为两类:

- `1 a b x`,表示将「结点 a 到结点 b 的简单路径」上所有结点的权值都增加 x
- `2 a`,表示求结点 a 的权值
- `3 a`,表示求 a 的子树上所有结点的权值之和

输入格式

第一行有三个整数 N, M 和 R

第二行有 N 个整数,第 i 个整数表示 v_i

在接下来的 $N-1$ 行中,每行两个整数,表示一条边

在接下来的 M 行中,每行一组操作

输出格式

对于每组 2 a 操作,输出一个整数,表示结点 a 的权值

数据范围与提示

对于所有数据, $1 \leq N, M \leq 10^6, 1 \leq R \leq N, -10^6 \leq v_i, x \leq 10^6$

DFS 遍历树，求出 DFS 序

考虑维护树上自底向上的差分数组 d

可用 树状数组 / 线段树 维护

对于 **操作1**

树上点分为维护即可（单点修改）

对于 **操作2**

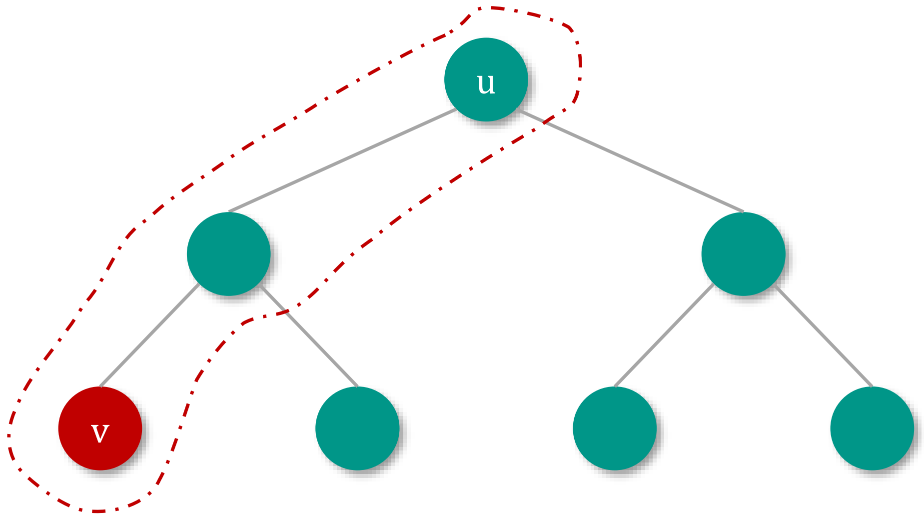
$\sum_{u \in \text{subtree}_a} d_u$ 即为答案（区间查询）

#948、DFS 序 3

对于 **操作3**

u 子树内答案为

$$\sum_{v \in \text{subtree}_u} \sum_{t \in \text{subtree}_v} d_t$$



对于任意 $v \in \text{subtree}_u$ 其对答案贡献 $(\text{dep}_v - \text{dep}_u + 1) \times d_v$

那么答案即为

$$\left(\sum_{v \in \text{subtree}_u} (\text{dep}_v - \text{dep}_u + 1) \times d_v\right) = \left(\sum_{v \in \text{subtree}_u} (\text{dep}_v \times d_v)\right) - (\text{dep}_u - 1) \times \sum_{v \in \text{subtree}_u} d_v$$

查询时 $\text{dep}_u - 1$ 已知，维护 d_v 以及 $\text{dep}_v \times d_v$ 即可

时间复杂度 $O(m \log n)$

#949、DFS 序 4

题目描述

本题严重卡常,请务必使用快读,不保证无快读的程序能过

给一棵有根树,这棵树由编号为 $1 \sim N$ 的 N 个结点组成

根结点的编号为 R

每个结点都有一个权值,结点 i 的权值为 v_i

接下来有 M 组操作,操作分为三类:

- `1 a x` ,表示将结点 a 的权值增加 x
- `2 a x` ,表示将 a 的子树上所有结点的权值增加 x
- `3 a b` ,表示求「结点 a 到结点 b 的简单路径」上所有结点的权值之和

输入格式

第一行有三个整数 N, M 和 R

第二行有 N 个整数,第 i 个整数表示 v_i

输出格式

对于每组 `3 a b` 操作,输出一个整数,表示「结点 a 到结点 b 的简单路径」上所有结点的权值之和(含结点 a, b)

数据范围与提示

对于 40% 的数据不含操作 2

对于全部的数据 $1 \le N, M \le 10^6, 1 \le R \le N, -10^6 \le v_i, x \le 10^6$

#949、DFS 序 4

DFS 遍历树，求出 DFS 序

若仅有 **操作1** 和 **操作3**

考虑维护各点到根节点的权值和，记 sum_u 为 $u \to R$ 的权值和

若点 u 发生修改，那么所有 $v \in \mathrm{subtree}_u$ 都应当修改

不妨维护 sum 在 DFS 序列上的差分 d（方便使用树状数组维护）

对于 **操作3** 可拆分成链查询即

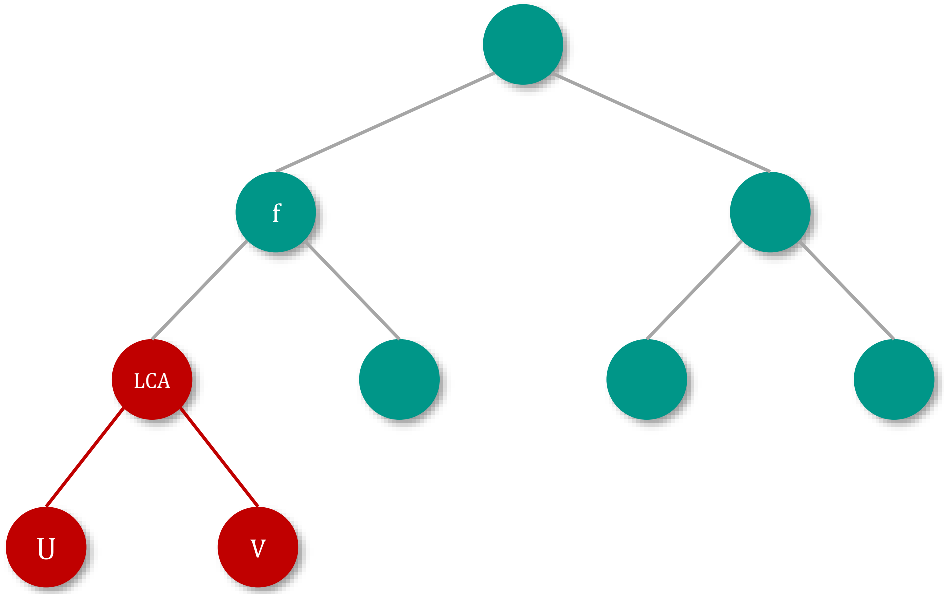
$$\mathrm{sum}_u + \mathrm{sum}_v - \mathrm{sum}_{\mathrm{LCA}(u,v)} - \mathrm{sum}_{f_{\mathrm{LCA}(u,v)}}$$

对于链查询对应为 DFS 序列 的前缀查询

考虑有 **操作2** 和 **操作3**

若 u 的子树增加 w，对于 $v \in \mathrm{subtree}_u$ 都应当产生影响

此时 $v \to u$ 共有 $\mathrm{dep}_v - \mathrm{dep}_u + 1$ 个点分别增加 w



#949、DFS 序 4

即 v 贡献为

$$(\mathrm{dep}_v - \mathrm{dep}_u + 1\,) \times w = \mathrm{dep}_v \times w - (\mathrm{dep}_u - 1) \times w$$

考虑 u 子树内所有 v 进行增量的维护

令树状数组 S_d 维护 sum 在 DFS 序列上的差分 d

对于 $-(\mathrm{dep}_u - 1) \times w$ 可作为 **操作1** 处理（即子树内各点加定值 $-(\mathrm{dep}_u - 1) \times w$）

额外令树状数组 S_w 维护各点 w 增量（即子树内各点加定值 w），需要统计答案时 $\times\ \mathrm{dep}_v$ 即可

那么对于一次 sum_u 询问

求出 S_w 中结果 P 以及 S_d 中结果 Q

$$(\mathrm{dep}_u \times P) + Q$$

即为所求

时间复杂度 $O(m \log n)$

欧拉序列

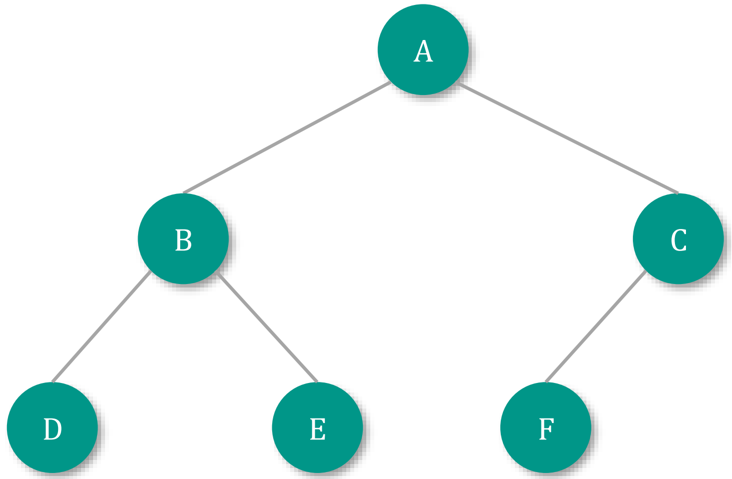
对一棵 n 个节点的树进行 DFS

无论是第一次访问还是回溯,每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，该序列被称作树的欧拉序列 E

欧拉序列中点回溯次数与节点度一致, 同时因访问时被加入序列

即



$$|E| = n + \sum d = n + n - 1$$

$$= 2n - 1$$

右图树的欧拉序列为

1	2	3	4	5	6	7	8	9	10	11
A	B	D	B	E	B	A	C	F	C	A

欧拉序列

记第一次进入节点 u 时的时间戳为 in_u

对于树上任意两点 u 和 v 不妨规定 $\mathrm{in}_v > \mathrm{in}_u$

$E_{\mathrm{in}_u \sim \mathrm{in}_v}$ 一定包含 u, v 的子树、两点间简单路径上所有点

由于须将子树内所有点遍历才会回溯至上一节点

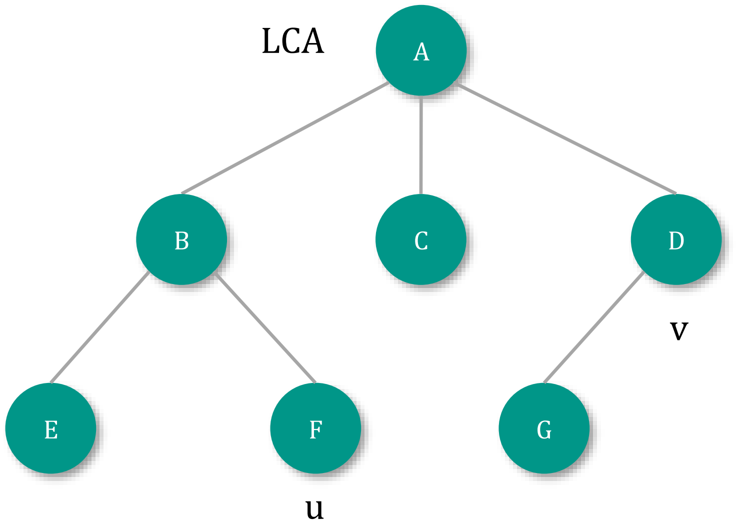
那么有 $\mathrm{LCA}(u, v)$ 为 $E_{\mathrm{in}_u \sim \mathrm{in}_v}$ 中 **深度最小的点**

于是将 LCA 转为了 RMQ 问题

可用 ST 表维护区间深度最小值下标

对于每次询问 $\mathrm{LCA}(u, v)$

回答 $E_{\mathrm{in}_u \sim \mathrm{in}_v}$中深度最小的点编号即可



id	1	2	3	4	5	6	7	8	9	10	11	12	13
u	A	B	E	B	F	B	A	C	A	D	G	D	A
dep	1	2	3	2	3	2	1	2	1	2	3	2	1

欧拉序列与LCA

```
void dfs(int u, int fa, int d)
{
    seq[++cnt] = u;
    dep[cnt] = d, in[u] = cnt;
    for (int i = head[u]; i; i = e[i].nxt)
    {
        if (e[i].v == fa)
            continue;
        dfs(e[i].v, u, d + 1);
        seq[++cnt] = u;
        dep[cnt] = d;
    }
}
void init()
{
    for (int i = 1; i <= cnt; i++)
        st[0][i] = i;
    for (int i = 1; i <= __lg(cnt); i++)
        for (int j = 1; j + (1 << i) - 1 <= cnt; j++)
        {
            int l = st[i - 1][j], r = st[i - 1][j + (1 << (i - 1))];
            st[i][j] = dep[l] < dep[r] ? l : r;
        }
}
int LCA(int u, int v)
{
    int l = in[u], r = in[v];
    if (l > r)
        swap(l, r);
    int s = __lg(r - l + 1);
    int a = st[s][l], b = st[s][r - (1 << s) + 1];
    return dep[a] < dep[b] ? seq[a] : seq[b];
}
```

对于 n 的节点 m 次询问 LCA

需要 $O(n\log n)$ 的预处理，每次询问 $O(1)$ 回答

总时间复杂度为 $O(n\log n + m)$

树链剖分

树链剖分用于将树分割成若干条链的形式,以维护树上路径的信息

具体来说将整棵树剖分为若干条链，使它组合成线性结构,进而使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、长链剖分和实链剖分

大多数情况下树链剖分都指重链剖分

重链剖分可将树上的任意一条路径划分成不超过 $\log n$ 条连续的链，每条链上的点深度互不相同

重链剖分能保证分出的每条链上的节点 DFS 序连续

因此可方便地用维护序列的数据结构来维护树上路径信息

除了配合数据结构来维护树上路径信息,树剖还可求 LCA

树链剖分

定义 **重子节点** 表示其子节点中子树最大的子结点

若有多个子树最大的子结点任取其一,若无子节点就无重子节点

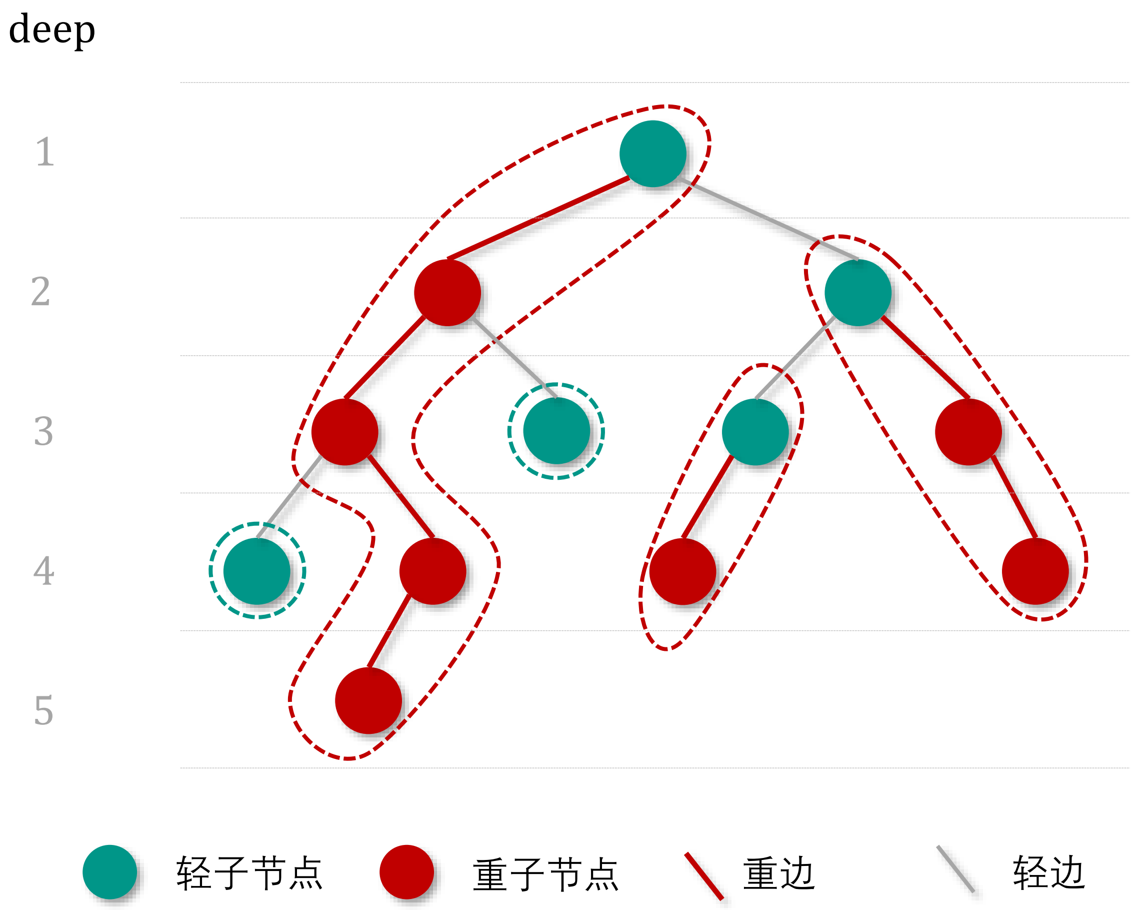
定义 **轻子节点** 表示剩余的所有子结点

从这个结点到重子节点的边为 **重边**

到其他轻子节点的边为 **轻边**

若干条首尾衔接的重边构成 **重链**

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



树链剖分

具体实现时需要如下数值:

fa_u 表示 u 的父节点编号

dep_u 表示 u 的在树上的深度

cnt_u 表示 u 的子树节点个数

son_u 表示 u 的重子节点编号

top_u 表示 u 的所在重链顶端点的编号(深度最小)

dfn_u 表示 u 的 DFS 序

rnk_x 表示 DFS 序 为 x 的节点编号, $\mathrm{rnk}_{\mathrm{dfn}_u} = u$

可进行两遍 DFS 预处理出这些值

其中第一次 DFS 求出 fa_u , dep_u , cnt_u , son_u

第二次 DFS 求出求出 top_u , dfn_u , rnk_x

```cpp
void dfs1(int u, int f = 0) // 第一次DFS
{
    son[u] = -1, cnt[u] = 1;
    dep[u] = dep[f] + 1, fa[u] = f;
    for (int i = head[u]; i; i = e[i].nxt)
    {
        if (e[i].v == f)
            continue;
        dfs1(e[i].v, u);
        cnt[u] += cnt[e[i].v];
        if (son[u] == -1 ||
            cnt[son[u]] < cnt[e[i].v]) //更新重子节点
            son[u] = e[i].v;
    }
}

void dfs2(int u, int t) // 第二次DFS
{
    top[u] = t, dfn[u] = ++tPos, rnk[tPos] = u;
    if (~son[u])   // 优先走重子节点
        dfs2(son[u], t);
    for (int i = head[u]; i; i = e[i].nxt)
        if (e[i].v != son[u] && e[i].v != fa[u])
            dfs2(e[i].v, e[i].v);
}
```

树链剖分

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(因为重边是对于每一个结点都有定义的)

所有的重链将整棵树 **完全剖分**

在剖分时重边优先遍历，最后树的 DFN 序上重链内的 DFN 序是**连续**的

按 DFN 排序后的序列即为剖分后的链

一颗子树内的 DFN 序是连续的

可以发现向下经过一条轻边 $u \leftrightarrow v$ 时 显然 $\mathrm{cnt}_v \le \mathrm{cnt}_u$（否则 v 应当为重孩子），即所在子树的大小至少会除以二

因此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 $\log n$ 次

因此树上的每条路径都可以被拆分成不超过 $\log n$ 条重链

树链剖分与LCA

若询问 LCA(u,v)

- 若 u 和 v 在同一条重链上

  答案为 u,v 中深度较小的点

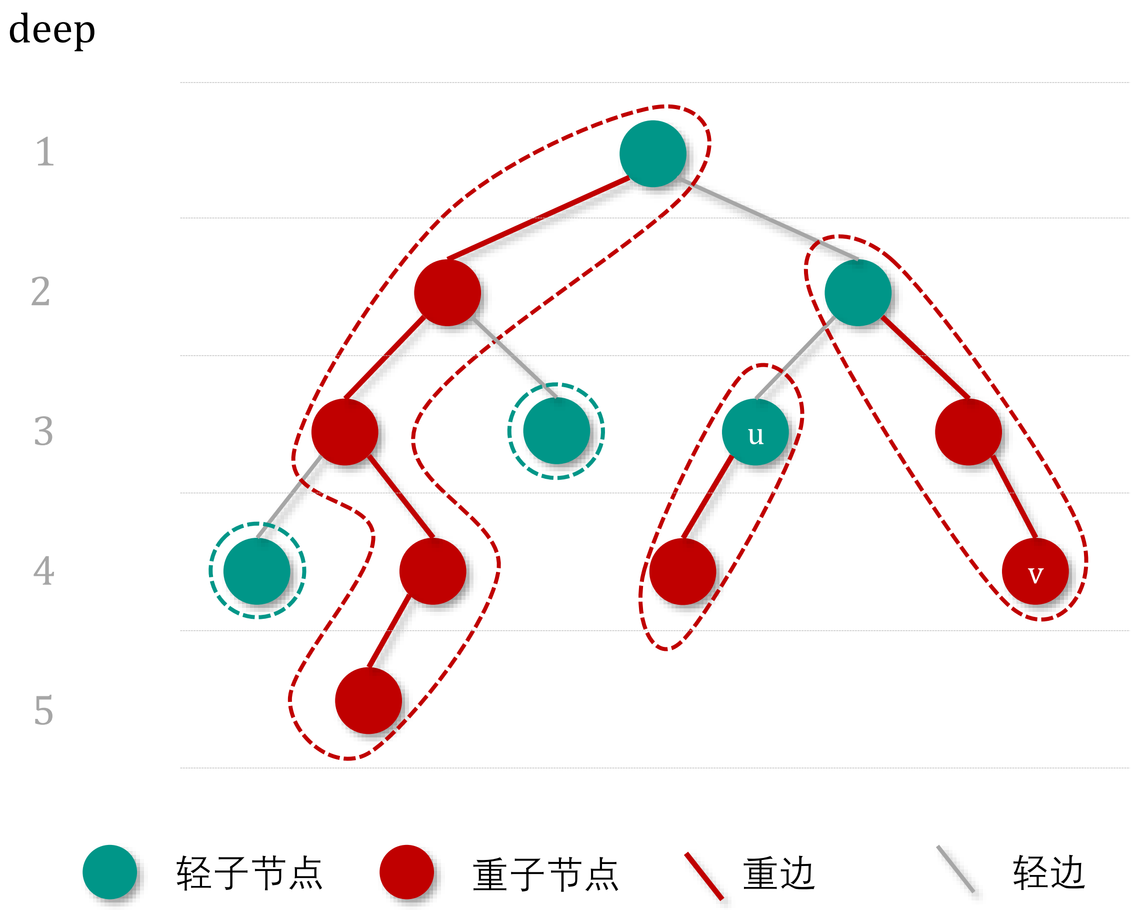
- 若 u 和 v 不在同一条重链上

  不断向上跳重链

  当跳到同一条重链上时,深度较小的结点即为 LCA

  向上跳重链时需要先跳所在重链顶端深度较大的那个

至多上跳 $\log n$ 条链，单次询问时间复杂度 $O(\log n)$



树链剖分与路径

路径信息维护

由于链上的 DFS 序连续

可使用线段树、树状数组维护整条链信息

每次选择重链顶端深度较大的链往上跳,直到两点在同一条链上

同样的跳链结构适用于维护、统计路径上的其他信息

由于至多经过 $\log n$ 条链 , 若使用线段树、树状数组维护, 单次维护时间复杂度$O(\log^2 n)$

子树信息维护

有时会要求,维护子树上的信息

在 DFS 搜索的时候,子树中的结点的 DFS 序是连续的

每一个结点记录 进出该节点的时间戳

可将子树信息转化为连续的一段区间信息

#1465、重链剖分

题目描述

如题,已知一棵包含 N 个结点的树(连通且无环)

节点 u 上有一个数值 val_u

现在需要你支持以下操作:

- `1 x y z` ,表示将树从 x 到 y 结点最短路径上所有节点的值都加上 z
- `2 x y` ,表示求树从 x 到 y 结点最短路径上所有节点的值之和
- `3 x z` ,表示将以 x 为根节点的子树内所有节点值都加上 z
- `4 x` 表示求以 x 为根节点的子树内所有节点值之和

输入格式

第一行包含 4 个正整数 N, M, R, P ,分别表示树的结点个数、操作个数、根节点序号和取模数(**即所有的输出结果均对此取模**)

接下来一行包含 N 个非负整数,分别依次表示各个节点上初始的数值

接下来 $N-1$ 行每行包含两个整数 x, y ,表示点 x 和点 y 之间连有一条边(保证无环且连通)

接下来 M 行每行包含若干个正整数,每行表示一个操作

输出格式

输出包含若干行,分别依次表示每个操作 2 或操作 4 所得的结果(**对** P **取模**)

数据规模

对于 30% 的数据, $1 \le N \le 10, 1 \le M \le 10$

对于 50% 的数据, $1 \le N \le 10^3, 1 \le M \le 10^3$

对于 100% 的数据, $1 \le N, M \le 10^5, 1 \le R, x, y \le N, 1 \le val_u, z, P \le 2^{31} - 1$

实验舱
青少年编程
走近科学 走进名校

谢谢观看