# 最近公共祖先

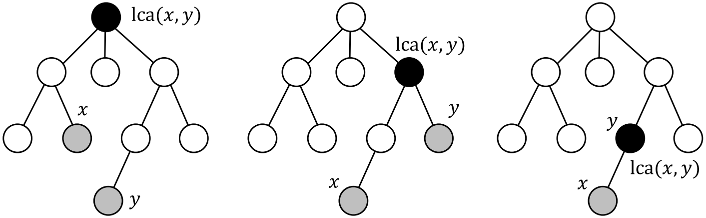
## Lowest Common Ancestor(LCA)

河南省实验中学信息技术组

2025 年 12 月 18 日

# 最近公共祖先

- 给定一颗有根树，若结点 $z$ 既是结点 $x$ 的祖先，也是结点 $y$ 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，深度最大的一个称为 $x, y$ 的最近公共祖先 (Lowest Common Ancestor,LCA)，记为 $\text{lca}(x, y)$




- 求最近公共祖先的方法通常有三种：
  - 朴素算法
  - 树上倍增法
  - Tarjan 算法

# 向上标记法

- 从 $x$ 向上走到根结点，并标记所有经过的结点。
- 从 $y$ 向上走到根结点，当第一次遇到已标记的结点时，就找到了 $\mathrm{lca}(x, y)$。
- 单次询问时间复杂度：$\mathrm{O}(N)$。

# 逐步上跳法

- 利用 DFS/BFS 遍历得出每个结点的深度和双亲信息。
- 找出结点 $x$ 和结点 $y$ 中深度较大大的结点 (例如结点 $y$)，然后让结点 $y$ 上跳至与结点 $x$ 同样的深度。
- 然后结点 $x$ 和结点 $y$ 同时上跳，直到相遇，此时的结点即为 $\mathrm{lca}(x, y)$。
- 单次询问时间复杂度：$\mathrm{O}(N)$。

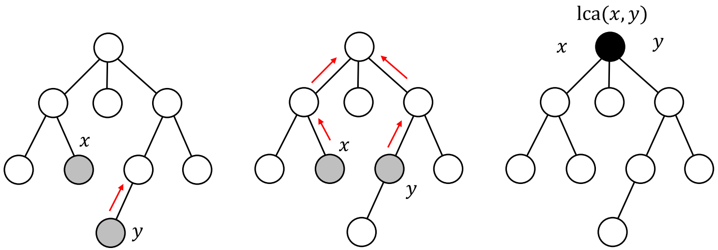



图: 逐步上跳法

# 逐步上跳法

```cpp
int p[N], d[N] = {0, 1}; // 双亲和深度 令 1 号顶点为根

void dfs(int x, int fa)
{
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if(y == fa) continue;
        p[y] = x, d[y] = d[x] + 1;
        dfs(y, x);
    }
}

int lca(int x, int y)
{
    if(d[x] > d[y]) swap(x, y);
    while(d[y] > d[x]) y = p[y];
    if(x == y) return x;
    while(x != y) x = p[x], y = p[y];
    return x;
}
```

# 树上倍增法

- 在朴素的上跳过程中，因为每次只跳跃一步，算法会比较缓慢。故而需要进行大步跳跃，这也就要求除了记录直接祖先外，还需要记录多级祖先。
- 定义 $f(x,k)$ 表示结点 $x$ 的 $2^k$ 辈祖先，即从 $x$ 向根结点走 $2^k$ 步到达的结点。
- 显然有 $f(x,0)$ 是 $x$ 的双亲结点。
- 对于每个结点 $x$，它的 $2^k$ 辈祖先应当是它的 $2^{k-1}$ 辈祖先的 $2^{k-1}$ 辈祖先，即

$$\forall k \in [1, \log n], f(x,k) = f(f(x,k-1),k-1)$$

- 利用 DFS/BFS 遍历得出每个结点的深度和 $f(x,0)$，同时利用倍增法，求出 $f(x,k)$。
- 预处理时间复杂度：$\mathrm{O}(N\log N)$。

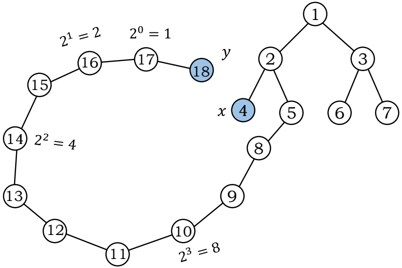
# 树上倍增法

```cpp
const int T = 20;
int f[N][T + 5];
int d[N] = {0, 1};  // 结点 1 为根结点

void dfs(int x, int fa)
{
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if(y == fa) continue;
        d[y] = d[x] + 1;
        f[y][0] = x;
        for(int k = 1; k <= T; ++k) f[y][k] = f[f[y][k - 1]][k - 1];
        dfs(y, x);
    }
}
```

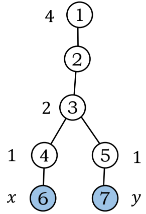
# 树上倍增法

- 已经预处理好 $f(x,k)$，那么如何让结点 $y$(深度较大) 跳到与 $x$ 同样深度呢？
- 例如，从结点 18(深度为 13) 上跳到结点 5(深度为 2)，需要跳 $13-2=11$ 步，那么显然 11 步可以拆分为 $2^3+2^1+2^0$，所以依次跳 $2^3=8$、$2^1=2$、$2^0=1$ 步即可到达结点 5(与结点 2 同样深度)。



# 树上倍增法

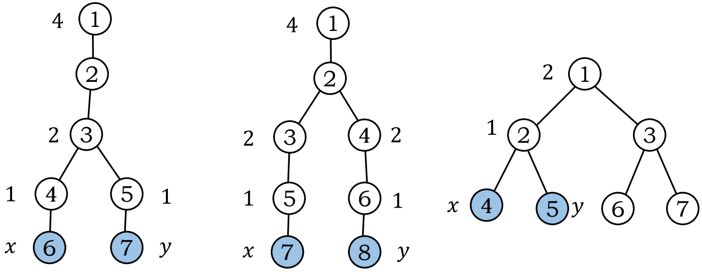
- 如果结点 $x$ 和上跳后的结点 $y$ 相等，那么 $\text{lca}(x,y)=x$。
- 但是大部分情况下，两点不会存在祖先—子孙关系，那么此时如何上跳到达最近公共祖先呢?
- 例如，结点 6 和结点 7 同时上跳，因为不知道跳多少步，那先跳最大一步 $2^2=4$ 步到达结点 1，从图中可以看出结点 1 不是结果。那此时只能跳 $2^1=2$ 步，到达结点 3，从图中可以看出结点 3 恰好是最近公共祖先。




- 在实际判断时，跳到结点 1 和跳到结点 3 时，我们获取到的条件完全一致，那么凭什么判定结点 1 不是最近公共祖先而结点 3 是最近公共祖先?

# 树上倍增法

- 所以跳到同一结点不能作为找到最近公共祖先的判定条件。
- 那么将条件改为跳到尽可能几乎相遇，也就是跳到两个结点的父节点相等，显然两个结点的父结点即为最近公共祖先。
- 我们可以用如下几个例子来演示：



# 树上倍增法



- 树上倍增法求最近公共祖先的算法：
- 利用 DFS/BFS 对于每个结点 $x$ 预处理出它的 $2^k$ 辈祖先 $f(x,k)$。
- 利用二进制拆分的方法，把结点 $y$(深度较大) 调整到与结点 $x$ 深度相同。
- 若此时结点 $x$ 的深度和上跳后结点 $y$ 相等，则 $\text{lca}(x,y)=x$。
- 否则利用二进制拆分的方法，把结点 $x,y$ 同上向上跳，保持二者深度一致且二者不相会。
- 此时结点 $x,y$ 的父结点就是最近公共祖先，即 $\text{lca}(x,y)=f(x,0)$。
- 单次询问时间复杂度：$\mathrm{O}(\log N)$。

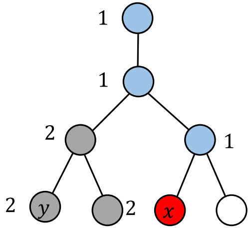
# 树上倍增法

```cpp
const int T = 20;
int f[N][T + 5];
int d[N] = {0, 1};  // 结点 1 为根结点

int lca(int x, int y)
{
    if(d[x] > d[y]) swap(x, y);
    for(int k = T; k >= 0; --k)   // 结点 y 上跳到与结点 x 同样深度
        if(d[f[y][k]] >= d[x]) y = f[y][k];
    if(x == y) return x;          // 结点 x 是结点 y 的祖先
    for(int k = T; k >= 0; --k)   // 结点 x 和结点 y 同时上跳
      if(f[x][k] != f[y][k]) x = f[x][k], y = f[y][k];
    return f[x][0];
}
```

# Tarjan 算法

- Tarjan 算法是一个离线算法，需要把 $m$ 个询问一次性读入，统一计算，最后统一输出。
- Tarjan 算法在对树进行后序遍历时，树中的结点分为三类：
  - 已经访问完毕并且回溯的结点，这些结点标记为 2。
  - 已经开始递归，但尚未回溯的结点，这些结点标记为 1。
  - 尚未访问的结点，这些结点没有标记。
- 对于正在访问的结点 $x$，它到根结点的路径上的结点都标记为 1。对于询问 $\text{lca}(x, y)$，若结点 $y$ 已经访问完毕并且回溯的结点 (标记为 2)，则 $\text{lca}(x, y)$ 就是从结点 $y$ 向上走到根，第一个遇到的标记为 1 的结点。



# Tarjan 算法

- 如此，查询结点 $y$ 最近的标记为 1 的祖先将花费主要时间，可以利用并查集进行优化。
- 当一个结点访问完毕并回溯时 (标记变为 2)，把它所在的集合合并到它的父结点所在的集合中。
- 合并时，它的父结点的标记一定为 1，而且单独构成一个集合，那么查询结点 $y$ 最近的标记为 1 的祖先结点等于查询集合的代表元。
- 当遍历到结点 $x$ 时，扫描与 $x$ 相关的所有询问，若询问中的另一个结点 $y$ 的标记为 2，则 $\mathrm{lca}(x, y) = get(y)$。
- 后序遍历的时间复杂度为 $\mathrm{O}(N)$，对于每个询问需要查询两次，复杂度为 $\mathrm{O}(M)$，故整体的时间复杂度为 $\mathrm{O}(N + M)$。

# Tarjan 算法

```cpp
struct Node { int y, id; }; // 询问的另外一个顶点和询问的序号
vector<Node> q[N]; // 每个顶点的询问

void tarjan(int x, int fa)
{
    vis[x] = 1; // 刚访问 标记为 1
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if(y == fa) continue;
        tarjan(y, x);
        f[y] = x;    // 合并到父结点
    }
    // 访问完毕并且回溯
    // 注意不能写在最后 否则不能处理结点相同的询问
    vis[x] = 2;
    for(int i = 0; i < q[x].size(); ++i)
    {
        int y = q[x][i].y, id = q[x][i].id;
        if(vis[y] == 2) lca[id] = get(y);
    }
}
```

# 树上前缀和

- 以任意一个结点 (例如结点 1) 作为根结点进行 BFS/DFS 得出每个结点 $x$ 到根结点的距离 $d[x]$。
- 那么显然 $d[x] = d[fa(x)] + w(x)$ 类似于前缀和，那么结点到其任意祖先的距离都可以通过前缀和做差的方式求出。

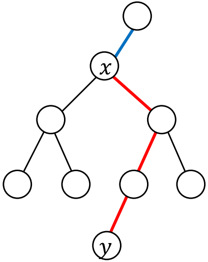
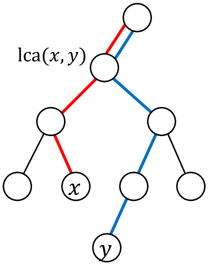



图: $dist(x, y) = d[y] - d[x]$

# 树上前缀和

- 对于一个有 $n$ 个结点的带权树，共有 $m$ 个询问，询问树中任意两个结点 $x, y$ 的距离。
- 对于询问 $x, y$ 之间的距离则为结点 $x$ 到最近公共祖先的距离加上加上结点 $y$ 到最近公共祖先到根结点距离，即

$$
\begin{aligned}
dist(x, y) &= d[x] - d[\mathrm{lca}(x, y)] + d[y] - d[\mathrm{lca}(x, y)] \\
&= d[x] + d[y] - 2 \times d[\mathrm{lca}(x, y)]
\end{aligned}
$$



# 树上差分



- 普通差分维护的是区间加减 $v$ 的情况，推广到树上顶点 $x$ 到根结点某一段路径上的顶点或边的某个权值加减 $v$。
- 根据修改的是路径上的顶点还是边的权值，树上差分可以分为点差分和边差分，二者实现上略有不同。
- 实际操作过程中，并非是针对从顶点 $x$ 根结点某一段路径上的权值修改，而是针对 $x$ 到 $y$ 的路径上的权值修改。

# 点差分

- 对于顶点 $x$ 到 $y$ 路径上的所有顶点权值都加上 $v$，可以转化对顶点差分值的修改

$$b[x]+=v, b[y]+=v$$
$$b[lca(x,y)]-=v, b[fa(lca(x,y))]-=v$$

- 一般会有多次修改，在所有修改操作完成后，可以计算一次子树和，就能得到更新后的点权值。

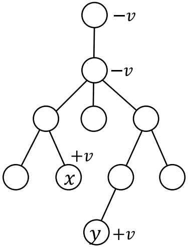



图: 对点的差分值修改

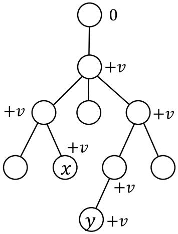



图: 对树求完子树和后

# 边差分

- 对于顶点 $x$ 到 $y$ 路径上的所有边权值都加上 $v$，一般在边的孩子结点中存储边权值，于是仍转化对顶点差分值的修改

$$b[x]+=v, b[y]+=v$$
$$b[lca(x,y)]-=2v$$

- 一般会有多次修改，在所有修改操作完成后，可以计算一次子树和，就能得到更新后的边的权值。

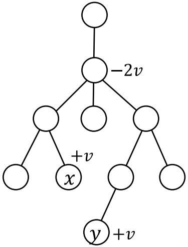



图: 对边的差分值修改

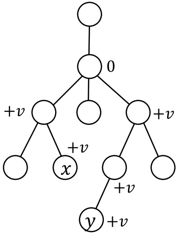



图: 对树求完子树和后

# 【例】最大流量

【题目描述】
Farmer John 在他的谷仓中安装了 $N-1$ 条管道，用于在 $N$ 个牛棚之间运输牛奶 $(2 \le N \le 50,000)$，牛棚方便地编号为 $1 \dots N$。每条管道连接一对牛棚，所有牛棚通过这些管道相互连接。
FJ 正在 $K$ 对牛棚之间泵送牛奶（$1 \le K \le 100,000$）。对于第 $i$ 对牛棚，你被告知两个牛棚 $s_i$ 和 $t_i$，这是牛奶以单位速率泵送的路径的端点。FJ 担心某些牛棚可能会因为过多的牛奶通过它们而不堪重负，因为一个牛棚可能会作为许多泵送路径的中转站。请帮助他确定通过任何一个牛棚的最大牛奶量。如果牛奶沿着从 $s_i$ 到 $t_i$ 的路径泵送，那么它将被计入端点牛棚 $s_i$ 和 $t_i$，以及它们之间路径上的所有牛棚。
【输入格式】
输入的第一行包含 $N$ 和 $K$。
接下来的 $N-1$ 行每行包含两个整数 $x$ 和 $y$（$x \neq y$），描述连接牛棚 $x$ 和 $y$ 的管道。
接下来的 $K$ 行每行包含两个整数 $s$ 和 $t$，描述牛奶泵送路径的端点牛棚。
【输出格式】
输出一个整数，表示通过谷仓中任何一个牛棚的最大牛奶量。

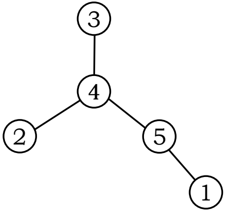
# 【例】最大流量

【样例 1 输入】

```
5 10
3 4
1 5
4 2
5 4
5 4
5 4
3 5
4 3
4 3
1 3
3 5
5 4
1 5
3 4
```

【样例 1 输出】

```
9
```



# 【例】最大流量

- 每一次从 $s$ 到 $t$ 的牛奶泵送都会将路径上的顶点权值加 1，显然可以用点差分解决。
- 令 $b[s]$ 和 $b[t]$ 都 $+1$，令 $b[lca(s,t)]$ 和 $b[fa(lca(s,t))]$ 都 $-1$。
- 最后遍历整棵树，计算子树和即为每个顶点的权值和，取最大值即可。
- 时间复杂度：$\mathrm{O}(N+M)$。

```cpp
tarjan(1, 0);
for(int i = 1; i <= m; ++i)
{
    int x = a[i].x, y = a[i].y, z = lca[i];
    ++d[x], ++d[y], --d[z], --d[p[z]];
}
dfs(1, 0);
int ans = 0;
for(int i = 1; i <= n; ++i) ans = max(ans, d[i]);
```

# 【例】暗之链锁



【题目描述】

传说中的暗之连锁被人们称为 Dark。Dark 是人类内心的黑暗的产物，古今中外的勇者们都试图打倒它。

经过研究，你发现 Dark 呈现无向图的结构，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。

# 【例】暗之链锁



**【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 1 输入】**

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

**【样例 1 输出】**

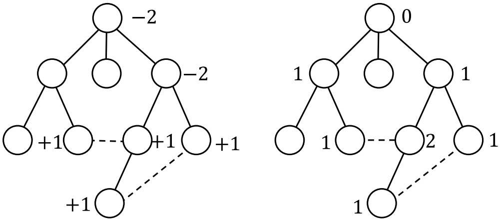
```
3
```

# 【例】暗之链锁

- 主要边构成一棵树，添加附加边后会形成环。
- 将附加边 $(x, y)$ 加入树中，会与树上 $x, y$ 之间构成一个环。此时，如果第一步选择切断 $x, y$ 之间路径上某条边，那么第二步就必须切断附加边 $(x, y)$，图才能不连通。
- 每条附加边 $(x, y)$ 相当于把树上 $x, y$ 之间路径上的每条边都覆盖一次。
- 统计每条主要边被覆盖的次数：
  - 如果第一步把覆盖了 0 次的主要边切断，那么第二步可以切断任意一条附加边。
  - 如果第一步把覆盖了 1 次的主要边切断，那么第二步只能切断那个覆盖了这条主要边一次的附加边。
  - 如果第一步把覆盖了 2 次及以上的主要边切断，那么第二步无论如何操作都无法使图不连通。

# 【例】暗之链锁

- 问题转化为：给定一棵树，然后给定若干次操作，每次将 $x$ 到 $y$ 路径上的每一条边覆盖一次，最终询问每条边覆盖的次数，显然是一个边差分问题。
- 定义 $d[x]$ 表示每个顶点的权值，那么对于操作 $(x, y)$，将 $d[x]$ 和 $d[y]$ 加 $1$，将 $d[lca(x, y)]$ 减 $1$。
- 然后进行一次 DFS，求出子树和也存在 $d[x]$ 中，那么 $d[x]$ 就是 $x$ 与双亲结点之间的边被覆盖的次数。
- 对于每一条边，如果覆盖次数为 $0$，答案加上 $m$，否则答案加 $1$。
- 时间复杂度：$\mathrm{O}(N + M)$。



LCA

河南省实验中学信息技术组



# 【例】暗之链锁

```
tarjan(1, 0);
for(int i = 1; i <= m; ++i)
{
    int x = a[i].x, y = a[i].y, z = lca[i];
    ++d[x], ++d[y], d[z] -= 2;
}
dfs(1, 0);
int ans = 0;
for(int i = 2; i <= n; ++i)
    if(d[i] == 0) ans += m;
    else if(d[i] == 1) ans += 1;
```

# 【例】雨天的尾巴

【题目描述】给定一棵有 $n(n \le 10^5)$ 的树，有 $m(m \le 10^5)$ 次发放操作，每次选择两个点 $x, y$，将 $x$ 到 $y$ 的路径上 (包括 $x$ 和 $y$) 的每一个点发放一袋 $z(z \le 10^9)$ 类型的物品。求完成所有发放操作后，每个点存放的最多的是哪种类型的物品。

【输入格式】

第一行包含两个整数 $n$ 和 $m$。

接下来 $n-1$ 行，每行两个整数 $u, v$，表示点 $u$ 与 $v$ 之间有边。

接下来 $m$ 行，每行三个整数 $x, y, z$，表示在点 $x, y$ 的路径上每个点发放一袋 $z$ 类型的物品。

【输出格式】

输出共 $n$ 行，第 $i$ 行一个整数表示第 $i$ 个点存放最多的是哪种类型的物品。

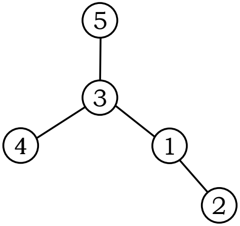
# 【例】雨天的尾巴

【样例 1 输入】

```
5 3
1 2
3 1
3 4
5 3
2 3 3
1 5 2
3 3 3
```

【样例 1 输出】

```
2
3
3
0
2
```



# 【例】雨天的尾巴

- 显然问题是一个点差分问题，每次操作相当于 $x$ 到 $y$ 路径上每个点都被类型 $z$ 覆盖了一次。
- 设 $d$ 表示差分计数的集合，那么每次操作转化为：点 $x$ 处增加一次物品 $z$，点 $y$ 处增加一次物品 $z$，点 $lca(x, y)$ 处减少一次物品 $z$，点 $fa(lca(x, y))$ 处减少一次物品 $z$。
- 问题是如何维护每个点上的所有物品的变化信息?

# 【例】雨天的尾巴

- 朴素做法
- 对物品的类型进行离散化 (物品类型 $\leq 10^9$)，然后对于每个顶点记录一个最大长度为 $m$ 的差分计数数组即可。
- 在最后 DFS 求树的子树和时，对每一种物品分别累加子树中对应物品的计数值即可。
- 时间复杂度：$O(NM \log M)$；空间复杂度：$O(NM)$。

# 【例】雨天的尾巴

- 动态开点权值线段树
- 对每个点，建立一棵动态开点的权值线段树代替差分计数数组，该线段树支持单点修改、查询区间最值及最值位置。
- 在最后 DFS 求树的子树和时，采用线段树合并的方式合并子树的线段树即可。
- 时间复杂度：$\mathrm{O}((N+M)\log(N+M))$。

# 练习

- 最近公共祖先 (洛谷 P3379)
- 牧场旅行 (COGS 186)
- 政党 (COGS 803)
- 距离咨询 (COGS 1588)
- 最大流量 (洛谷 P3128)
- 暗之链锁 (COGS 2434)
- 雨天的尾巴 (COGS 3380)
- 天天爱跑步 [NOIP 2016](COGS 2557)
- 异象石 (COGS 3552)
- 次小生成树 (COGS 2453)
- 疫情控制 [NOIP 2012](COGS 1267)
- 聚会 [AHOI 2008](COGS 3507)