树上倍增、图论选讲

wcbf

Feb 14, 2019

最近公共祖先

昨天，我们介绍了最近公共祖先（LCA）问题：

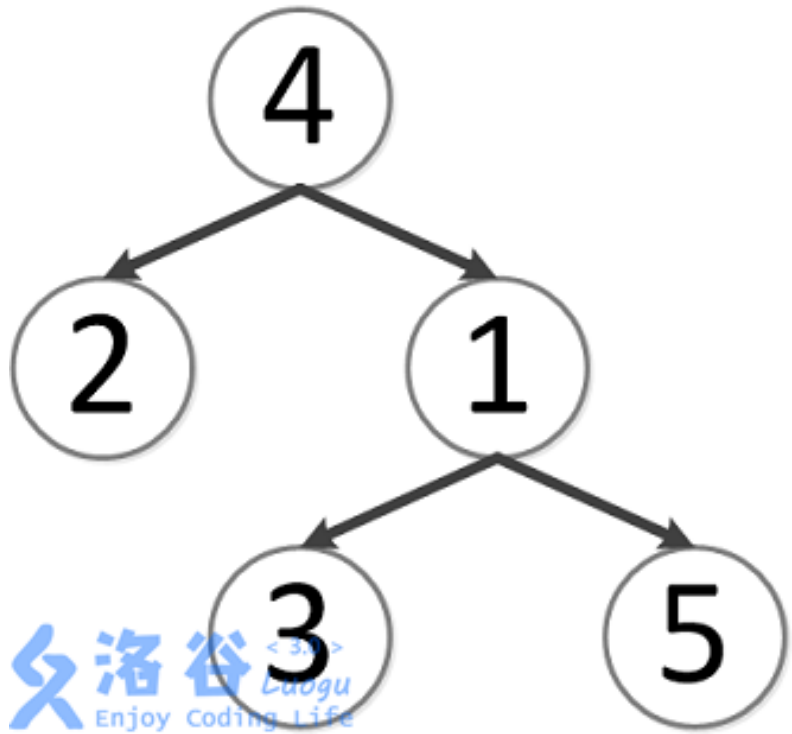
最近公共祖先

昨天，我们介绍了最近公共祖先（LCA）问题：

> 给定一棵树，每次询问给出树上的两个节点 u, v，需要回答 u, v 的公共祖先节点中，最深的那个，记为 $LCA(u, v)$。如果其中一个节点已经是另一个节点的祖先，那么，它们的 LCA 就定义成那个祖先。

最近公共祖先

例如，下图所示的树中，节点 4 是节点 2 和节点 3 的唯一公共祖先，因此 $LCA(2,3)=4$；节点 3 和节点 5 的公共祖先有 4 和 1，但 1 是最深的，因此 $LCA(3,5)=1$。此外，$LCA(4,5)=4$，因为节点 4 已经是节点 5 的祖先。



最近公共祖先

复习：用 ST 表求最近公共祖先

用 ST 表求最近公共祖先的主要步骤如下

1. 首先对树进行 dfs 操作，对于每个节点的每次访问，按照访问的先后顺序分配一个编号（每个节点可能有多个编号）；

最近公共祖先

复习：用 ST 表求最近公共祖先

用 ST 表求最近公共祖先的主要步骤如下

1. 首先对树进行 dfs 操作，对于每个节点的每次访问，按照访问的先后顺序分配一个编号（每个节点可能有多个编号）；
2. 询问 u, v 的 LCA 时，取 u, v 的任意一个编号（记为 $\mathrm{id}[u], \mathrm{id}[v]$），则编号在 $[\mathrm{id}[u], \mathrm{id}[v]]$ 中，深度最小的节点就是 u 和 v 的 LCA。

最近公共祖先

复习：用 ST 表求最近公共祖先

用 ST 表求最近公共祖先的主要步骤如下

1. 首先对树进行 dfs 操作，对于每个节点的每次访问，按照访问的先后顺序分配一个编号（每个节点可能有多个编号）；
2. 询问 u, v 的 LCA 时，取 u, v 的任意一个编号（记为 $\mathrm{id}[u], \mathrm{id}[v]$），则编号在 $[\mathrm{id}[u], \mathrm{id}[v]]$ 中，深度最小的节点就是 u 和 v 的 LCA。
3. 原问题转换为区间最值查询（RMQ）。RMQ 可以用 ST 表完成，预处理时间复杂度为 $O(n \log n)$，每次查询的复杂度为 $O(1)$。

树上倍增

除了可以用 ST 表求最近公共祖先外，另一个常用的求 LCA 的算法称为**树上倍增**。

树上倍增

除了可以用 ST 表求最近公共祖先外，另一个常用的求 LCA 的算法称为**树上倍增**。

当我们需要访问一个节点的祖先节点时，常用的方法是记录每个节点的父亲节点 $p[x]$，然后根据 $p[x]$ 逐级向上跳转，直到跳转到需要访问的祖先节点为止。

树上倍增

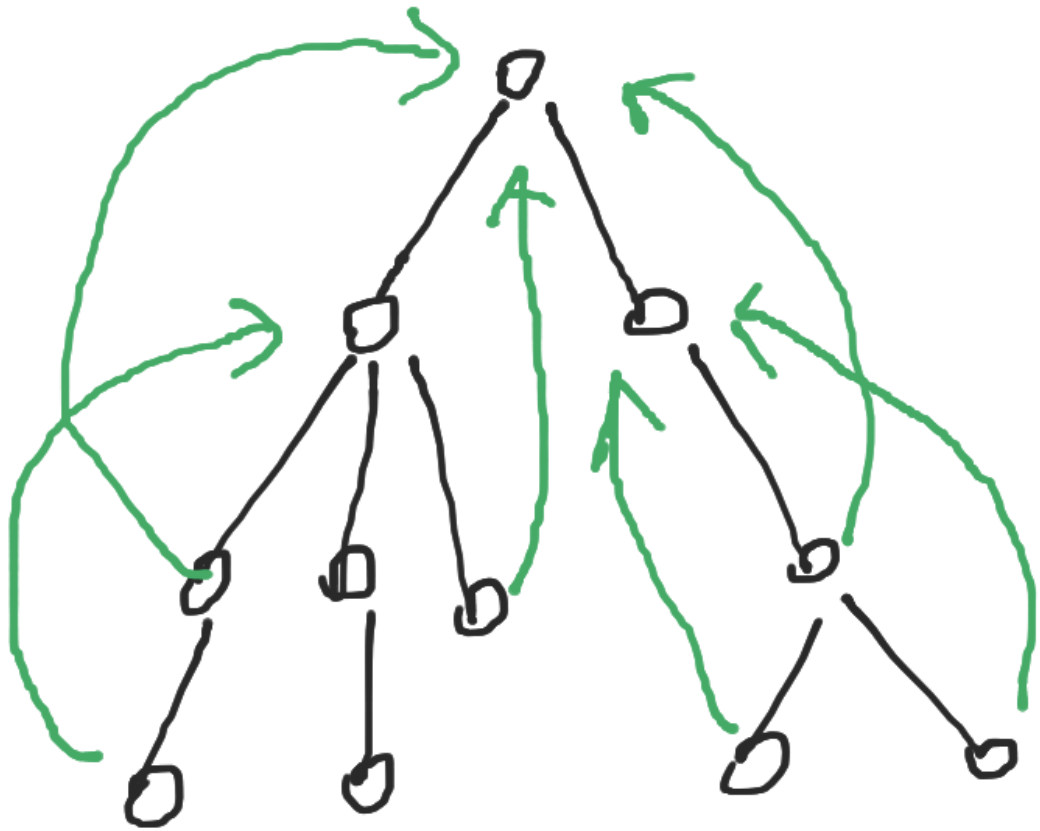
除了可以用 ST 表求最近公共祖先外，另一个常用的求 LCA 的算法称为**树上倍增**。

当我们需要访问一个节点的祖先节点时，常用的方法是记录每个节点的父亲节点 $p[x]$，然后根据 $p[x]$ 逐级向上跳转，直到跳转到需要访问的祖先节点为止。

如果祖先节点距离当前节点很远，这样做很耗时。那么，我们是不是可以一次跳多级，而不是一级一级往上跳呢？

树上倍增

下图中，黑色边连向父节点，绿色边指向二级祖先（并没有全部画出）



树上倍增

- 树上倍增使用了一个称为**倍增表**的数组实现了这个思想。

树上倍增

- 树上倍增使用了一个称为**倍增表**的数组实现了这个思想。
- 倍增表 fa 是一个二维数组，其中 $\mathsf{fa}[u][k]$ 表示 u 的 2^k 级祖先（如果不存在，则为 0）。

树上倍增

- 树上倍增使用了一个称为**倍增表**的数组实现了这个思想。
- 倍增表 fa 是一个二维数组，其中 $\mathsf{fa}[u][k]$ 表示 u 的 2^k 级祖先（如果不存在，则为 0）。
- $\mathsf{fa}[u][0]$ 就是 u 的父节点。

树上倍增

- 树上倍增使用了一个称为**倍增表**的数组实现了这个思想。
- 倍增表 fa 是一个二维数组，其中 $\mathsf{fa}[u][k]$ 表示 u 的 2^k 级祖先（如果不存在，则为 0）。
- $\mathsf{fa}[u][0]$ 就是 u 的父节点。
- 原理：任何数字都能唯一表示成若干不同的 2 的次幂的和（例如：$13 = 8 + 4 + 1$）。

树上倍增

首先，通过一遍 dfs 求出每个节点的父亲节点和深度。代码中，$depth[u]$ 存放 u 的深度，$p[u]$ 存放 u 的父亲节点。

```
vector<int> adj[MAXN];

void dfs(int u, int par, int dep) {
    depth[u] = dep;
    p[u] = par;
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (v == par) continue;
        dfs(v, u, dep + 1);
    }
}
```

树上倍增

有了每个节点的父亲信息，就很容易求出倍增表。

求倍增表和求 ST 表的过程很相似：要求出一个节点的 2^j 级祖先，只要求出它的 2^{j-1} 级祖先的 2^{j-1} 级祖先即可。

```
void build() {
    for (int i = 1; i <= n; i++) fa[i][0] = p[i];
    for (int j = 1; j < MAXM; j++)
        for (int i = 1; i <= n; i++)
            fa[i][j] = fa[fa[i][j-1]][j-1];
}
```

树上倍增

有了每个节点的父亲信息，就很容易求出倍增表。

求倍增表和求 ST 表的过程很相似：要求出一个节点的 2^j 级祖先，只要求出它的 2^{j-1} 级祖先的 2^{j-1} 级祖先即可。

```
void build() {
    for (int i = 1; i <= n; i++) fa[i][0] = p[i];
    for (int j = 1; j < MAXM; j++)
        for (int i = 1; i <= n; i++)
            fa[i][j] = fa[fa[i][j-1]][j-1];
}
```

如果我们把根节点的父节点设为 0，呢么倍增表中不存在的祖先都会被置为 0（想一想，为什么？）

树上倍增

用树上倍增求 LCA

现在我们求出了倍增表，那么如何用倍增表求出 u, v 的 LCA 呢？这主要分为两步

1. 如果 u 和 v 的深度不同（不妨假设 u 更深），将 u 上移到和 v 深度相同为止；

树上倍增

用树上倍增求 LCA

现在我们求出了倍增表，那么如何用倍增表求出 u, v 的 LCA 呢？这主要分为两步

1. 如果 u 和 v 的深度不同（不妨假设 u 更深），将 u 上移到和 v 深度相同为止；
2. 如果此时 u 和 v 相同，那这就是它们的 LCA；否则，同时将 u 和 v 上移，直到它们相遇为止。

树上倍增

用树上倍增求 LCA

现在我们求出了倍增表，那么如何用倍增表求出 u, v 的 LCA 呢？这主要分为两步

1. 如果 u 和 v 的深度不同（不妨假设 u 更深），将 u 上移到和 v 深度相同为止；
2. 如果此时 u 和 v 相同，那这就是它们的 LCA；否则，同时将 u 和 v 上移，直到它们相遇为止。

树上倍增

用树上倍增求 LCA

现在我们求出了倍增表，那么如何用倍增表求出 u, v 的 LCA 呢？这主要分为两步

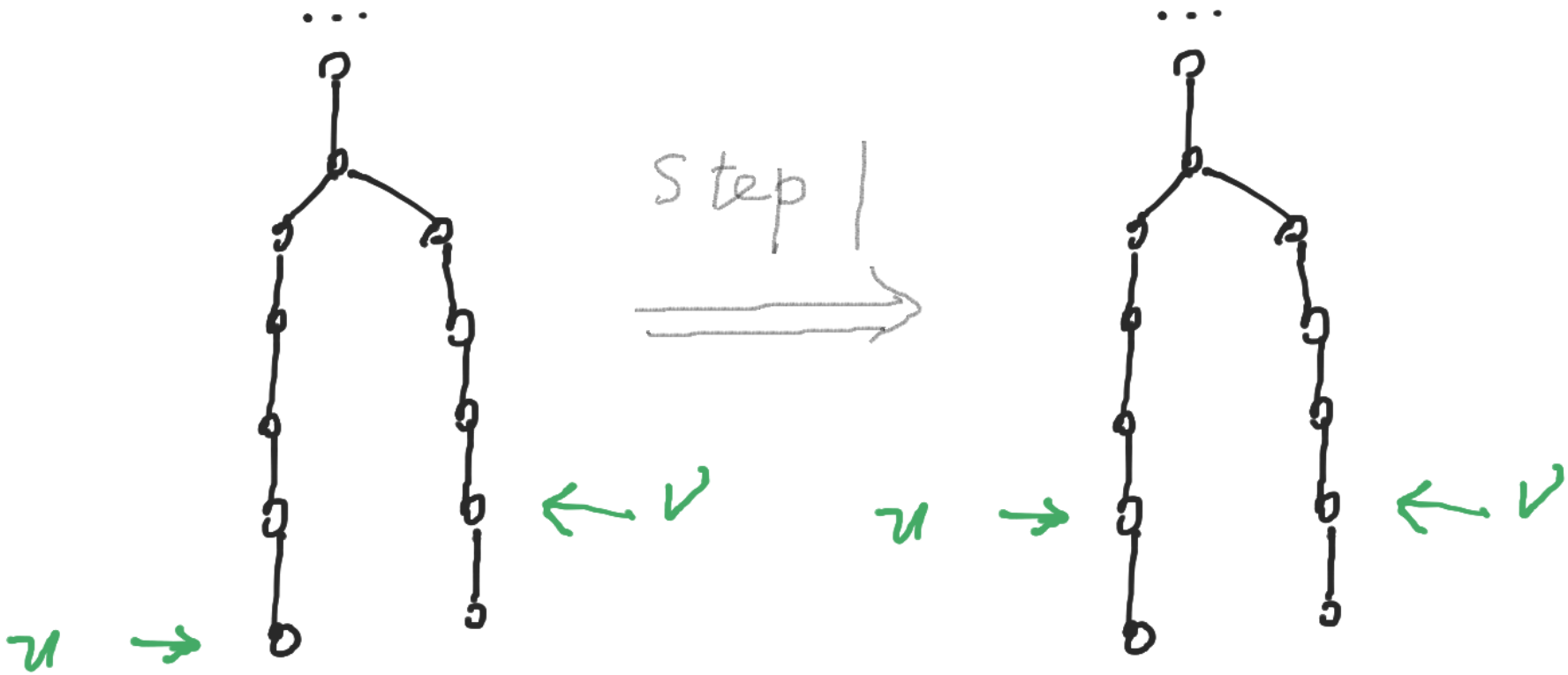
1. 如果 u 和 v 的深度不同（不妨假设 u 更深），将 u 上移到和 v 深度相同为止；
2. 如果此时 u 和 v 相同，那这就是它们的 LCA；否则，同时将 u 和 v 上移，直到它们相遇为止。

在这两步中，“上移”操作可以用倍增表很快地实现。

树上倍增

用树上倍增求 LCA

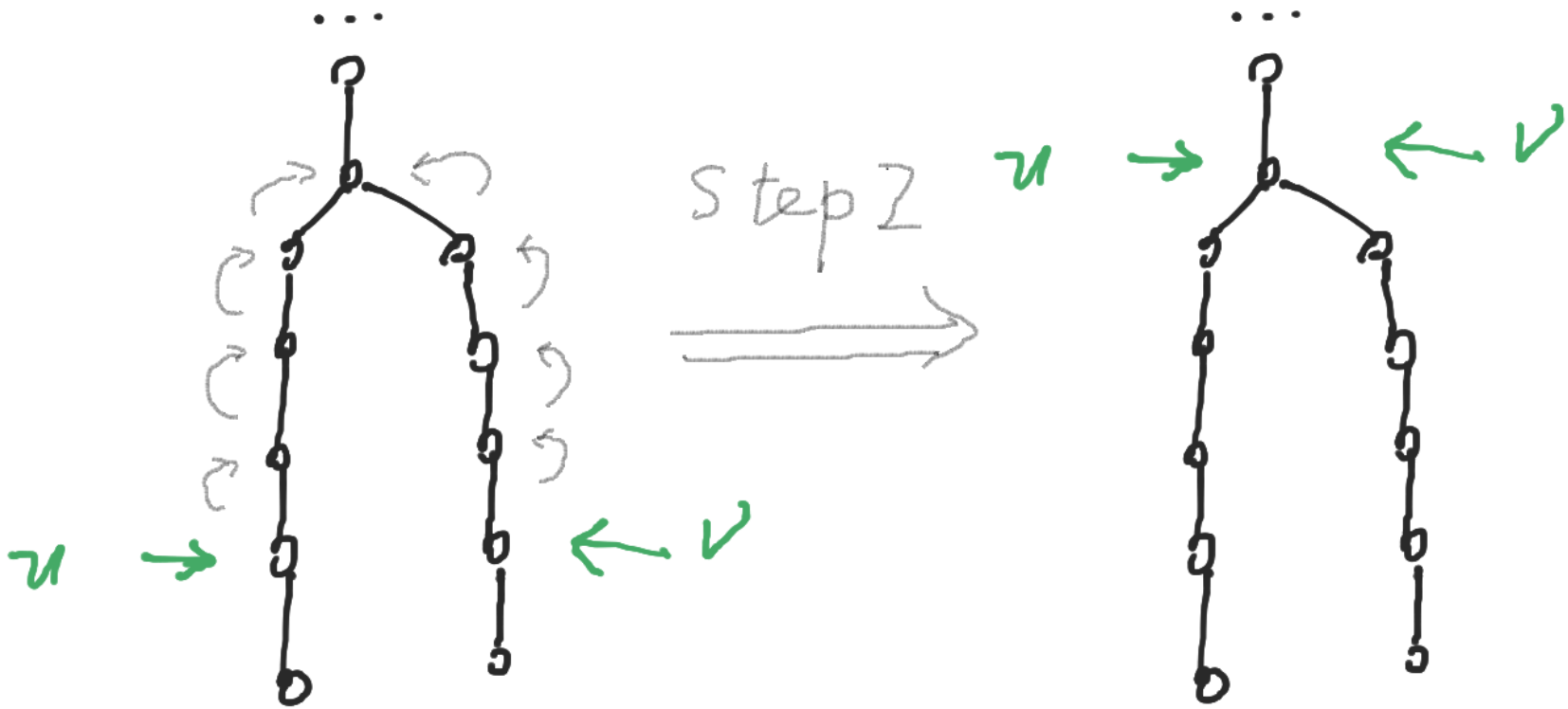
求 LCA 的第一步：



树上倍增

用树上倍增求 LCA

求 LCA 的第二步：



树上倍增

用树上倍增求 LCA

```
int lca(int u, int v) {
    // Step 1:
    if (depth[u] < depth[v]) swap(u, v);
    if (depth[u] > depth[v]) {
        for (int i = MAXM - 1; i >= 0; i--)
            if (depth[fa[u][i]] > depth[v])
                u = fa[u][i];
        u = p[u];
    }
    if (u == v) return u;
    // Step 2:
    for (int i = MAXM - 1; i >= 0; i--)
        if (fa[u][i] != fa[v][i]) {
            u = fa[u][i];
            v = fa[v][i];
        }
    return p[u];
}
```

树上倍增

用树上倍增求 LCA

从最大可跳的步数开始跳，如果跳完之后 u 比 v 深才跳。最后，在向上跳一级，两个节点的深度就相同了。

```
int lca(int u, int v) {
    // Step 1:
    if (depth[u] < depth[v]) swap(u, v);
    if (depth[u] > depth[v]) {
        for (int i = MAXM - 1; i >= 0; i--)
            if (depth[fa[u][i]] > depth[v])
                u = fa[u][i];
        u = p[u];
    }
    if (u == v) return u;
    // Step 2:
    // ...
}
```

树上倍增

用树上倍增求 LCA

类似地，从最大可跳的步数开始跳，如果跳完之后 u 和 v 不相同才跳。最后，在向上跳一级，就是这两个节点的 LCA。

```
int lca(int u, int v) {
    // Step 1:
    // ......
    // Step 2:
    for (int i = MAXM - 1; i >= 0; i--)
        if (fa[u][i] != fa[v][i]) {
            u = fa[u][i];
            v = fa[v][i];
        }
    return p[u];
}
```

树上倍增

用树上倍增求 LCA

由于使用了倍增表，我们可以在 $O(\log n)$ 时间内完成跳转，相比 $O(n)$ 的暴力跳转，这是一个很大的改进。

树上倍增

用树上倍增求 LCA

由于使用了倍增表，我们可以在 $O(\log n)$ 时间内完成跳转，相比 $O(n)$ 的暴力跳转，这是一个很大的改进。

这样，我们得到了一个 $O(n)$ 预处理，单次询问 $O(\log n)$ 回答的 LCA 算法。

树上倍增

用树上倍增求 LCA

练习：洛谷 P3379

请使用树上倍增的方法求出 LCA。

树上倍增

用树上倍增求 LCA

花了这么长时间讨论 LCA 的求法，那么 LCA 到底有什么实际应用呢？

树上倍增

用树上倍增求 LCA

花了这么长时间讨论 LCA 的求法，那么 LCA 到底有什么实际应用呢？

树中的任意两个节点 u, v 必然存在唯一一条简单路径，而这条简单路径又可以以 $\mathrm{LCA}(u, v)$ 为分隔点，分成两条垂直路径！

在求倍增表的过程中，我们可以维护一些额外的信息，这样就可以查询两点之间路径的相关信息！

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

Solution

先考虑部分分的做法。

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

Solution

先考虑部分分的做法。

对于每个询问 u, v，从大到小依次加入每一条边，直到 u, v 连通为止。最后加入的边的权值就是答案。

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

Solution

先考虑部分分的做法。

对于每个询问 u, v，从大到小依次加入每一条边，直到 u, v 连通为止。最后加入的边的权值就是答案。

可以用并查集维护连通性。

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

Solution

其实有个结论：权值最小的边权值最大的路径，一定在最大生成树上。

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

Solution

其实有个结论：权值最小的边权值最大的路径，一定在最大生成树上。

这样问题就变成了，给定一棵边带权的树，每次给定两个点 u, v，查询树上这两个点之间路径中，权值最小边的权值。

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

Solution

其实有个结论：权值最小的边权值最大的路径，一定在最大生成树上。

这样问题就变成了，给定一棵边带权的树，每次给定两个点 u, v，查询树上这两个点之间路径中，权值最小边的权值。

在构造倍增表时，额外维护一个数组 $minv[x][k]$，表示节点 x 到它的 2^k 级祖先之间的路径中，边权的最小值。

树上倍增

洛谷 P1967：货车运输

给定一张带权无向图，需要回答若干个询问。每个询问给出两个节点 u, v，求 u, v 之间的路径，使得这条路径中权值最小的边的权值最大。

Solution

其实有个结论：权值最小的边权值最大的路径，一定在最大生成树上。

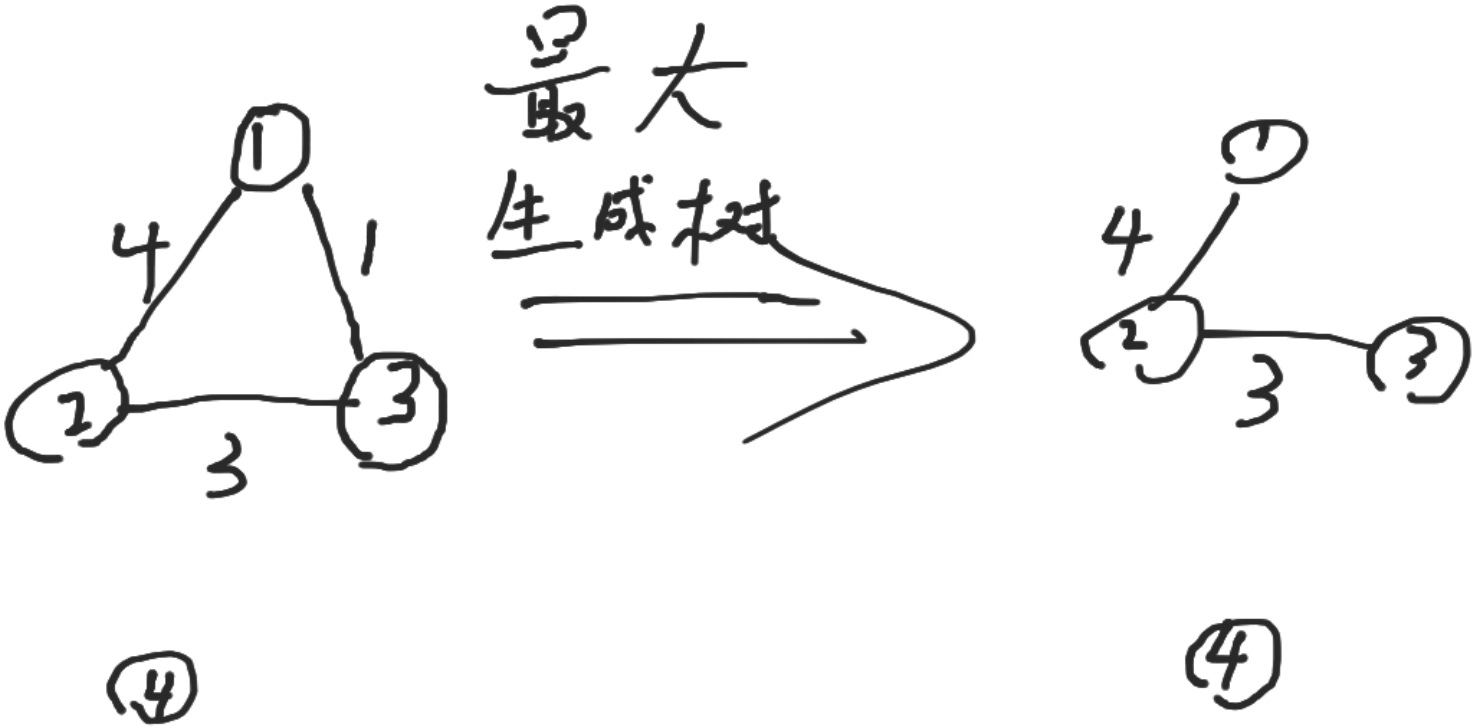
这样问题就变成了，给定一棵边带权的树，每次给定两个点 u, v，查询树上这两个点之间路径中，权值最小边的权值。

在构造倍增表时，额外维护一个数组 $minv[x][k]$，表示节点 x 到它的 2^k 级祖先之间的路径中，边权的最小值。

只需要对求 LCA 的代码稍加改动即可。

树上倍增

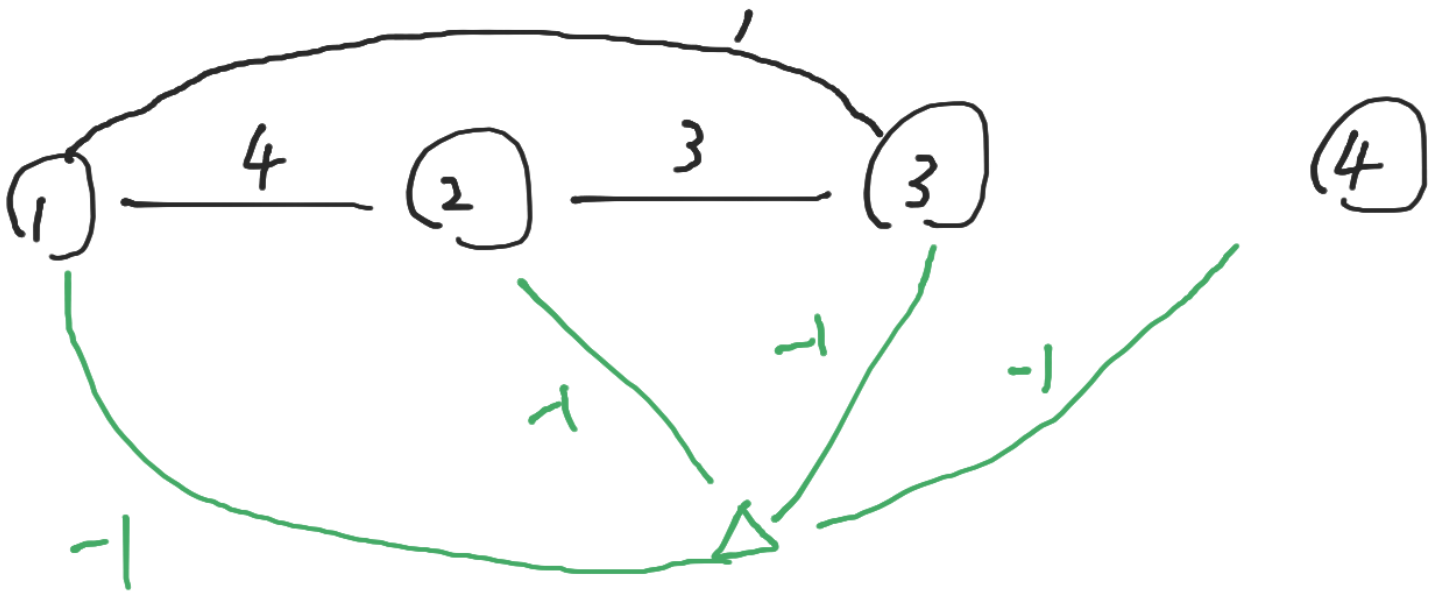
洛谷 P1967：货车运输



树上倍增

洛谷 P1967：货车运输

为避免原图不连通可能带来的麻烦，我们可以建立一个特殊节点，这个特殊节点和原图中其他每个节点之间连一条权值为 -1 的边。如果询问的两个点在原图中不连通，就必然为经过特殊节点，答案必然是 -1。



树上倍增

洛谷 P1967：货车运输

首先求最大生成树。

```
int ufs[MAXN];
int find(int x) { /* ... */ }
void kruskal() {
    for (int i = 1; i <= n; i++) ufs[i] = i;
    sort(edges.begin(), edges.end());
    for (int i = 0; i < edges.size(); i++) {
        int u = edges[i].u, v = edges[i].v, w = edges[i].w;
        int uu = find(u), vv = find(v);
        if (uu != vv) {
            ufs[uu] = vv;
            adj[u].push_back(make_pair(v, w));
            adj[v].push_back(make_pair(u, w));
        }
    }
}
```

树上倍增

洛谷 P1967：货车运输

dfs 时需要额外记录到父亲节点的距离。

```
int depth[MAXN], p[MAXN], d[MAXN];

void dfs(int u, int par, int dist, int dep) {
    depth[u] = dep; p[u] = par; d[u] = dist;
    for (int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i].first, w = adj[u][i].second;
        if (v == par) continue;
        dfs(v, u, w, dep + 1);
    }
}
```

树上倍增

洛谷 P1967：货车运输

构造倍增表时，额外计算 minv 数组。

```
int fa[MAXN][MAXM], minv[MAXN][MAXM];

void build() {
    for (int i = 1; i <= n; i++) {
        fa[i][0] = p[i];
        minv[i][0] = d[i];
    }
    for (int j = 1; j < MAXM; j++)
        for (int i = 1; i <= n; i++) {
            fa[i][j] = fa[fa[i][j-1]][j-1];
            minv[i][j] = min(minv[i][j-1], minv[fa[i][j-1]][j
              -1]);
        }
}
```

树上倍增

洛谷 P1967：货车运输

```
int query(int u, int v) {
    // Step 1:
    int ans = INT_MAX;
    if (depth[u] < depth[v]) swap(u, v);
    if (depth[u] > depth[v]) {
        for (int i = MAXM - 1; i >= 0; i--)
            if (depth[fa[u][i]] > depth[v]) {
                ans = min(ans, minv[u][i]);
                u = fa[u][i];
            }

        ans = min(ans, d[u]);
        u = p[u];
    }
    if (u == v) return ans;
    // Step 2:
    // ...
}
```

树上倍增

洛谷 P1967：货车运输

```
int query(int u, int v) {
    // Step 1:
    // ...
    // Step 2:
    for (int i = MAXM - 1; i >= 0; i--)
        if (fa[u][i] != fa[v][i]) {
            ans = min(ans, minv[u][i]); u = fa[u][i];
            ans = min(ans, minv[v][i]); v = fa[v][i];
        }
    return min(ans, min(d[u], d[v]));
}
```

图论选讲

- 洛谷 P2661：信息传递
- 洛谷 P1351：联合权值
- 洛谷 P2296：寻找道路
- 洛谷 P1137：旅行计划