

Graph Theory

1 Diameter

性质：对于任意一个点 u ，距离 u 最远的点一定是直径的一个端点。

但是路径不满足这个性质。hack: 考虑路径是直径本身。涉及到路径的时候，需要更细致的讨论。

1.1

前提条件： $w \geq 0$ ，但是这个方法找到直径比较方便。

```
vector<i64> dep(n + 1);
auto dfs = [&](auto&& dfs, int u, int fa) -> void {
    for (auto v : adj[u]) {
        if (v == fa) continue;
        dep[v] = dep[u] + w;
        dfs(dfs, v, u);
    }
};

dep[1] = 0, dfs(dfs, 1, 0);
int x = max_element(dep.begin() + 1, dep.end()) - dep.begin();
dep[x] = 0, dfs(dfs, x, 0);
```

```
i64 ret = *max_element(dep.begin() + 1, dep.end());
```

1.2

```
i64 ret = 0;
vector<array<i64, 2>> dp(n + 1);

auto dfs = [&](auto&& dfs, int u, int fa) -> void {
    dp[u] = { 0, 0 };
    for (auto v : adj[u]) {
        if (v == fa) continue;
        dfs(dfs, v, u);

        i64 get = dp[v][0] + w;
        if (get > dp[u][0]) {
            dp[u][1] = dp[u][0];
            dp[u][0] = get;
        } else if (get > dp[u][1]) {
            dp[u][1] = get;
        }
    }
    ret = max(ret, dp[u][0] + dp[u][1]);
}; dfs(dfs, 1, 0);
```

1.3 例题

给你一棵树，选择一条长度不超过 k 的路径，最小化树上任意一点到这条路径的距离的最大值。

通过上面的性质，合理地放缩，不容易证明：选择的路径一定是直径的一部分。于是双指针、单调队列一下就做完了。你也可以使用 ST 表和二分大力草过去。

2 Center

感觉这个东西屁用没有啊

最小化 $\max \text{dep}_{\text{center}}(u)$ 的点。

性质：

- 中心最多有两个。如果有两个，他们相邻（和重心一样）。
- 中心一定位于树的直径上
- 对于任意一个点 u ，其最长路径一定经过中心。
- 当通过在两棵树间连一条边以合并为一棵树时，连接两棵树的中心可以使新树的直径最小。
- 树的中心到其他任意节点的距离不超过树直径的一半。

换根 dp 维护最大值和次大值即可实现。

```
vector<array<i64, 2>> dp(n + 1);
auto dfs1 = [&](auto&& dfs, int u, int fa) -> void {
    dp[u] = { 0, 0 };
    for (auto [v, w] : adj[u]) {
        if (v == fa) continue;
        dfs(dfs, v, u);

        i64 get = dp[v][0] + w;
        if (get > dp[u][0]) {
            dp[u][1] = dp[u][0];
            dp[u][0] = get;
        } else if (get > dp[u][1]) {
            dp[u][1] = get;
        }
    }
}; dfs1(dfs1, 1, 0);

auto dfs2 = [&](auto&& dfs, int u, int fa) -> void {
    for (auto [v, w] : adj[u]) {
        if (v == fa) continue;
        i64 get;
        if (dp[v][0] + w == dp[u][0]) {
            get = dp[u][1] + w;
        } else get = dp[u][0] + w;

        if (get > dp[v][0]) {
            dp[v][1] = dp[v][0];
            dp[v][0] = get;
        } else if (get > dp[v][1]) {
            dp[v][1] = get;
        }

        dfs(dfs, v, u);
    }
}; dfs2(dfs2, 1, 0);
```

3 Centroid

感觉这个比较重要。这是把分治思想应用到树上的基础，性质也比较多。

性质：

- 删除重心后，任意一棵子树的大小小于等于 $\lfloor \frac{n}{2} \rfloor$
- 考虑以重心为根，那么节点的**深度和最小**
- 最多存在两个重心。如果存在，两个重心相邻，且**删掉这条边后连通块大小相等**
- 删除一个点，如果删的是重心，那么剩余的子树的最大值最小
- 两棵树合并的时候，新树的重心一定在两棵树重心的路径上
- 重心一定在根节点所在重链上。根节点的重子树的重心一定是整棵树重心的后代。

实现代码参考**点分治**部分的 `get_root()` 部分

3.1 例题

其实我也不知道放什么题了。

对于所有 u 求 u 的子树的重心。 u 的子树的重心一定是 u 的重儿子的重心的祖先，那么考虑从重儿子的重心暴力往上跳。

注意到跳的边一定是重边，而重边最多也就 $n - 1$ 条，所以时间复杂度是 $O(n)$ 的。

```
vector<int> ans(n + 1), son(n + 1), siz(n + 1);
auto dfs = [&](auto&& dfs, int u) -> void {
    siz[u] = 1;
    for (auto v : adj[u]) {
        dfs(dfs, v);
        siz[u] += siz[v];
        if (siz[v] > siz[son[u]]) {
            son[u] = v;
        }
    }
    if (son[u] == 0) {
        assert(adj[u].empty());
        ans[u] = u;
        return;
    }

    ans[u] = ans[son[u]];
    for (int& x = ans[u]; x != u; x = fa[x]) {
        int y = fa[x];
        int now = max(siz[son[x]], siz[u] - siz[x]);
        int nxt = max(siz[son[y]], siz[u] - siz[y]);
        if (nxt >= now) break;
    }
}; dfs(dfs, 1);
```

4 点分治

好像确实就是：如果我能够快速求解过某个点的情况，那么就可以用点分治。类似于分治的：如果我能够快速求解过区间中点的答案，那么就可以用分治。

```
vector<int> vis(n + 1), siz(n + 1);
auto get_root = [&](int u, int size) -> int {
    int root = 0, Min = size;
    auto dfs = [&](auto dfs, int u, int fa) -> void {
        siz[u] = 1;
        int Max = 0;
        for (auto [v, w] : adj[u]) {
            if (vis[v] || v == fa) continue;
            dfs(dfs, v, u);
            siz[u] += siz[v];
            Max = max(Max, siz[v]);
        }
        Max = max(Max, size - siz[u]);
        if (Max < Min) {
            Min = Max, root = u;
        }
    }; dfs(dfs, u, 0);
    return root;
};

auto dfz = [&](auto&& dfz, int u, int size) -> void {
    u = get_root(u, size);

    // 这段是求答案的
    // 维护每条链的长度以及它是从哪个儿子出发的，就可以  $O(n\log n)$  求解
    // 时间复杂度  $O(n\log^2 n)$ 
    // Map<int, vector<int>> d;
    // d[0].push_back(u);
    // auto dfs = [&](auto&& dfs, int u, int fa, int dep, int from) -> void {
    //     d[dep].push_back(from);
    //     for (auto [v, w] : adj[u]) {
    //         if (vis[v] || v == fa) continue;
    //         dfs(dfs, v, u, dep + w, from);
    //     }
    // };
    // for (auto [v, w] : adj[u]) {
    //     if (vis[v]) continue;
    //     dfs(dfs, v, u, w, v);
    // }
    // for (auto& [x, pos] : d) {
    //     sort(pos.begin(), pos.end());
    //     pos.erase(unique(pos.begin(), pos.end()), pos.end());
    // }
    // for (int i = 1; i <= m; ++i) {
    //     if (ans[i]) continue;
    //     for (auto& [x, pos] : d) {
    //         auto it = d.find(query[i] - x);
    //         if (it == d.end()) continue;
    //         ans[i] |= pos.size() > 1 | (it->second).size() > 1 | pos[0] != (it->second)[0];
    //     }
    // }
```

```

vis[u] = 1;
for (auto [v, w] : adj[u]) {
    if (vis[v]) continue;
    if (siz[v] < siz[u]) {
        dfz(dfz, v, siz[v]);
    } else {
        dfz(dfz, v, size - siz[u]);
    }
}
}; dfz(dfz, 1, n);

```

5 LCA

5.1 倍增

```

void solve() {
    int n, q, rt;
    std::vector<std::vector<int>> adj(n + 1);

    std::vector<int> dep(n + 1);
    std::vector fa(std::__lg(n) + 1, std::vector<int> (n + 1));

    auto dfs = [&](auto&& dfs, int u) -> void {
        dep[u] = dep[fa[0][u]] + 1;
        for (auto& v : adj[u]) {
            if (v == fa[0][u]) {
                continue;
            }
            fa[0][v] = u;
            dfs(dfs, v);
        }
    }; dfs(dfs, rt);

    for (int i = 1; i <= std::__lg(n); ++i) {
        for (int u = 1; u <= n; ++u) {
            fa[i][u] = fa[i - 1][fa[i - 1][u]];
        }
    }

    auto lca = [&](int u, int v) -> int {
        if (dep[u] < dep[v]) std::swap(u, v);
        while (dep[u] != dep[v]) {
            u = fa[std::__lg(dep[u] - dep[v])][u];
        }
        if (u == v) return u;
        for (int i = std::__lg(n); i >= 0; --i) {
            if (fa[i][u] != fa[i][v]) {
                u = fa[i][u], v = fa[i][v];
            }
        }
        return fa[0][u];
    };
}

```

5.2 树剖

```

struct HLD {
    int n, cur;
    std::vector<std::vector<int>> adj;
    std::vector<int> dfn, idfn, siz, fa, top, dep;

    HLD() = default;
    HLD(int n) { init(n); }

    void init(int n) {
        this -> n = n;
        cur = 0;
        adj.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        idfn.assign(n + 1, 0);
        siz.assign(n + 1, 0);
        fa.assign(n + 1, 0);
        top.assign(n + 1, 0);
        dep.assign(n + 1, 0);
    }

    void add(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void work(int rt = 1) {
        dfs1(rt), dfs2(rt);
    }

    void dfs1(int u) {
        if (fa[u] != 0) {
            adj[u].erase(find(adj[u].begin(), adj[u].end(), fa[u]));
        }
        siz[u] = 1;
        for (auto& v : adj[u]) {
            dep[v] = dep[fa[v] = u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                std::swap(v, adj[u][0]);
            }
        }
    }

    void dfs2(int u) {
        dfn[u] = ++cur;
        idfn[cur] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
    }
}

```

```

int lca(int u, int v) {
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]]) {
            u = fa[top[u]];
        } else {
            v = fa[top[v]];
        }
    }
    return dep[u] < dep[v] ? u : v;
}

int jump(int u, int k) {
    assert(dep[u] >= k);
    int d = dep[u] - k;
    while (dep[top[u]] > d)
        u = fa[top[u]];
    return idfn[dfn[u] - dep[u] + d];
}
};

```

5.3 O(1) LCA

```

struct LCA {
    int n, cur;
    std::vector<std::vector<int>> adj, st;
    std::vector<int> fa, dep, dfn, siz;

    LCA() = default;
    LCA (int n) { init(n); }

    void init(int n) {
        this -> n = n;
        cur = 0;
        adj.assign(n + 1, {});
        st.assign(std::__lg(n) + 1, std::vector<int> (n + 1));
        fa.assign(n + 1, 0);
        dep.assign(n + 1, 0);
        dfn.assign(n + 1, 0);
        siz.assign(n + 1, 0);
    }

    void add(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int u) {
        if(fa[u] != 0) adj[u].erase(find(adj[u].begin(), adj[u].end(), fa[u]));
        siz[u] = 1;
        st[0][dfn[u] = ++cur] = u;
        for(auto v : adj[u]) {
            dep[v] = dep[fa[v] = u] + 1;
            dfs(v);
            siz[u] += siz[v];
        }
    }
};

```

```

int merge(int x, int y) {
    return dep[x] < dep[y] ? x : y;
}

void work(int rt = 1) {
    dep[rt] = 1; dfs(rt);
    for(int i = 1; i <= std::__lg(n); ++i) {
        for(int j = 1; j + (1 << i) - 1 <= n; ++j) {
            st[i][j] = merge(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
        }
    }
}

int lca(int u, int v) {
    if (u == v) return u;
    u = dfn[u], v = dfn[v];
    if (u > v) std::swap(u, v);
    int k = std::__lg(v - u);
    return fa[merge(st[k][u + 1], st[k][v - (1 << k) + 1])];
}
};

```

6 rooted functions

```

bool isAncestor(int f, int u) {
    return dfn[f] <= dfn[u] && dfn[u] <= dfn[f] + siz[f] - 1;
}

int rootedParent(int rt, int u) {
    if (rt == u) return rt;
    if (!isAncestor(u, rt)) return fa[u];
    // 需要保证 adj[u] 里面只有子节点, 不能包含父节点
    auto it = std::upper_bound(adj[u].begin(), adj[u].end(), rt, [&](int x, int y) {
        return dfn[x] < dfn[y];
    }) - 1;
    return *it;
}

int rootedSize(int rt, int u) {
    if (rt == u) return n;
    if (!isAncestor(u, rt)) return siz[u];
    return n - siz[rootedParent(rt, u)];
}

int rootedLca(int rt, int u, int v) {
    return lca(rt, u) ^ lca(u, v) ^ lca(v, rt);
}

```


7 Tarjan

7.1 强连通分量

```

struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    SCC() = default;
    SCC(int n) { init(n); }

    void init(int n) {
        this -> n = n;
        adj.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        low.assign(n + 1, 0);
        bel.assign(n + 1, 0);
        stk.clear();
        cur = cnt = 0;
    }

    void add(int u, int v) {
        adj[u].push_back(v);
    }

    void dfs(int u) {
        dfn[u] = low[u] = ++cur;
        stk.push_back(u);

        for (auto v : adj[u]) {
            if (dfn[v] == 0) {
                dfs(v);
                low[u] = std::min(low[u], low[v]);
            } else if (bel[v] == 0) {
                low[u] = std::min(low[u], dfn[v]);
            }
        }

        if (dfn[u] == low[u]) {
            int x = cnt++;
            do {
                bel[x = stk.back()] = cnt;
                stk.pop_back();
            } while (x != u);
        }
    }

    void work() {
        for (int i = 1; i <= n; i++) {
            if (dfn[i] == 0) dfs(i);
        }
    }
}

```

```

auto getGraph() {
    work();
    std::vector<std::vector<int>> adj(cnt + 1);
    for (int u = 1; u <= n; ++u) {
        for (auto v : adj[u]) {
            if (bel[u] != bel[v]) {
                adj[bel[u]].push_back(bel[v]);
            }
        }
    }
    return std::pair { cnt, std::move(adj) };
}
};

```

7.2 边双连通分量

```

struct EDCC {
    int n;
    int cur, cnt, edges;
    std::vector<int> stk, dfn, low, bel;
    std::vector<std::array<int, 2>> cut;
    std::vector<std::vector<std::array<int, 2>>> adj;

    EDCC() = default;
    EDCC(int n) { init(n); }

    void init(int n) {
        this->n = n;
        adj.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        low.assign(n + 1, 0);
        bel.assign(n + 1, 0);
        stk.clear();
        cut.clear();
        cur = cnt = edges = 0;
    }
    void add(int u, int v) {
        ++edges;
        adj[u].push_back({ v, edges });
        adj[v].push_back({ u, edges });
    }
    void dfs(int u, int fa) {
        dfn[u] = low[u] = ++cur;
        stk.push_back(u);
        for (auto [v, w] : adj[u]) {
            if (w == fa) continue;
            if (dfn[v] == 0) {
                dfs(v, w);
                low[u] = std::min(low[u], low[v]);
                if (dfn[u] < low[v]) {
                    cut.push_back({ u, v });
                }
            } else {
                low[u] = std::min(low[u], dfn[v]);
            }
        }
    }
}

```

```

    if (low[u] == dfn[u]) {
        int x = ++cnt;
        do {
            bel[x = stk.back()] = cnt;
            stk.pop_back();
        } while (x != u);
    }
}

void work() {
    for (int i = 1; i <= n; ++i) {
        if (dfn[i] == 0) dfs(i, 0);
    }
}

auto getTree() {
    work();
    std::vector<std::vector<int>> adj(cnt + 1);
    for (auto [u, v] : cut) {
        adj[bel[u]].push_back(bel[v]);
        adj[bel[v]].push_back(bel[u]);
    }
    return std::pair { cnt, std::move(adj) };
}
};

```

7.3 点双连通分量

```

struct VDCC {
    int n, cur;
    std::vector<std::vector<int>> adj, vdcc;
    std::vector<int> dfn, low, stk, cut;

    VDCC() = default;
    VDCC (int n) { init(n); }

    void init(int n) {
        this -> n = n;
        cur = 0;
        adj.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        low.assign(n + 1, 0);
        cut.assign(n + 1, 0);
        vdcc.clear();
        stk.clear();
    }

    void add(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
}

```

```

void dfs(int u, int fa) {
    int son = 0;
    stk.push_back(u);
    low[u] = dfn[u] = ++cur;
    for (auto v : adj[u]) {
        if (dfn[v] == 0) {
            son += 1;
            dfs(v, u);
            low[u] = std::min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                if (fa != 0) cut[u] = 1;
                std::vector<int> cc { u };
                int x;
                do {
                    cc.push_back(x = stk.back());
                    stk.pop_back();
                } while (x != v);
                vdcc.push_back(cc);
            }
        } else if (v != fa) {
            low[u] = std::min(low[u], dfn[v]);
        }
    }

    if (fa == 0 && son == 0) vdcc.push_back({u});
    if (fa == 0 && son >= 2) cut[u] = 1;
}

void work() {
    for (int i = 1; i <= n; ++i) {
        if (dfn[i] == 0) dfs(i, 0);
    }
}

auto getTree() {
    work();
    int m = vdcc.size();
    std::vector<std::vector<int>> adj(n + m + 1);
    for (int i = 1; i <= m; ++i) {
        for (auto x : vdcc[i - 1]) {
            adj[x].push_back(i + n);
            adj[i + n].push_back(x);
        }
    }
    return std::pair { m, std::move(adj) };
}
};

```

8 Virtual Tree

```

struct VirtualTree {
    int n, tot;
    std::vector<std::vector<int>> adj, st;
    std::vector<int> fa, dep, dfn, siz;

    VirtualTree (int n) { init(n); }

    void init (int n) {
        this -> n = n;
        tot = 0;
        adj.assign(n + 1, {});
        st.assign(std::__lg(n) + 1, std::vector<int> (n + 1));
        fa.assign(n + 1, 0);
        dep.assign(n + 1, 0);
        dfn.assign(n + 1, 0);
        siz.assign(n + 1, 0);
    }

    void add (int u, int v) {
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }

    void dfs (int u) {
        if(fa[u] != 0) adj[u].erase(find(adj[u].begin(), adj[u].end(), fa[u]));
        siz[u] = 1;
        st[0][dfn[u] = ++tot] = u;
        for(auto v : adj[u]) {
            dep[v] = dep[fa[v] = u] + 1;
            dfs(v);
            siz[u] += siz[v];
        }
    }

    int merge (int x, int y) {
        return dep[x] < dep[y] ? x : y;
    }

    void work (int rt = 1) {
        dep[rt] = 1; dfs(rt);
        for(int i = 1; i <= std::__lg(n); ++i) {
            for(int j = 1; j + (1 << i) - 1 <= n; ++j) {
                st[i][j] = merge(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
            }
        }
    }

    int lca (int u, int v) {
        if(u == v) return u;
        u = dfn[u], v = dfn[v];
        if(u > v) std::swap(u, v);
        int len = std::__lg(v - u);
        return fa[merge(st[len][u + 1], st[len][v - (1 << len) + 1])];
    }
}

```

```

template<typename Iter, typename Func>
int build(Iter l, Iter r, Func&& link) {
    std::vector p(l, r);
    std::sort(p.begin(), p.end(), [&](int x, int y) {
        return dfn[x] < dfn[y];
    });

    std::vector<int> stk;
    stk.push_back(1);

    int len = p.size(), x = p[0] == 1;
    for (int i = p[0] == 1; i < len; ++i) {
        int u = p[i];
        int w = lca(u, stk.back());
        if (w != stk.back()) {
            while (stk.size() >= 2 && dep[w] <= dep[stk[stk.size() - 2]]) {
                link(stk[stk.size() - 2], stk.back());
                x |= stk[stk.size() - 2] == 1;
                stk.pop_back();
            }

            if (stk.back() != w) {
                link(w, stk.back());
                stk.pop_back();
                stk.push_back(w);
            }
        }
        stk.push_back(u);
    }

    len = stk.size();
    for (int i = !x; i < len - 1; ++i) {
        link(stk[i], stk[i + 1]);
    }
    return stk[!x];
}
};

```

9 Kruskal 重构树