Beijing Normal University
School of Mathematics

Template

app1eDog

2024 年 10 月 30 日

目录

1 hpp

1.1 heading

```cpp
#include <bits/stdc++.h>

// using namespace std;

#define typet typename T
#define typeu typename U
#define types typename... Ts
#define tempt template <typet>
#define tempu template <typeu>
#define temps template <types>
#define tandu template <typet, typeu>

using LL = long long;
using i128 = __int128;
using PII = std::pair<int, int>;
/*
using UI = unsigned int;
using ULL = unsigned long long;
using ULL = unsigned long long;
using PIL = std::pair<int, LL>;
using PLI = std::pair<LL, int>;
using PLL = std::pair<LL, LL>;
*/
using vi = std::vector<int>;
using vvi = std::vector<vi>;
using vl = std::vector<LL>;
using vvl = std::vector<vl>;
using vpi = std::vector<PII>;

#define ff first
#define ss second
#define all(v) v.begin(), v.end()
#define rall(v) v.rbegin(), v.rend()

#ifdef LOCAL
#include "../debug.h"
#else
#define debug(...) \
    do {           \
    } while (false)
#endif

constexpr int mod = 998244353;
constexpr int inv2 = (mod + 1) / 2;
constexpr int inf = 0x3f3f3f3f;
constexpr LL INF = 1e18;
constexpr double pi = 3.141592653589793;
constexpr double eps = 1e-6;

constexpr int lowbit(int x) { return x & -x; }
/*
constexpr int add(int x, int y) { return x + y < mod ? x + y : x - mod + y; }
constexpr int sub(int x, int y) { return x < y ? mod + x - y : x - y; }
constexpr int mul(LL x, int y) { return x * y % mod; }
constexpr void Add(int& x, int y) { x = add(x, y); }
constexpr void Sub(int& x, int y) { x = sub(x, y); }
constexpr void Mul(int& x, int y) { x = mul(x, y); }
constexpr int pow(int x, int y, int z = 1) {
    for (; y; y /= 2) {
        if (y & 1) Mul(z, x);
        Mul(x, x);
    }
    return z;
}
temps constexpr int add(Ts... x) {
    int y = 0;
    (..., Add(y, x));
    return y;
}
temps constexpr int mul(Ts... x) {
    int y = 1;
    (..., Mul(y, x));
    return y;
}
*/

tandu bool Max(T& x, const U& y) { return x < y ? x = y, true : false; }
tandu bool Min(T& x, const U& y) { return x > y ? x = y, true : false; }

```

```cpp
void solut() {

}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    int t = 1;
    std::cin >> t;
    while (t--) {
        solut();
    }
    return 0;
}
```

1.2 debug.h

```cpp
template <typename T, typename U>
std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p) {
    return os << '<' << p.first << ',' << p.second << '>';
}

template <
    typename T, typename = decltype(std::begin(std::declval<T>())),
    typename = std::enable_if_t<!std::is_same_v<T, std::string>>>
std::ostream& operator<<(std::ostream& os, const T& c) {
    auto it = std::begin(c);
    if (it == std::end(c)) return os << "{}";
    for (os << '{' << *it; ++it != std::end(c); os << ',' << *it);
    return os << '}';
}

#define debug(arg...)                \
    do {                             \
        std::cerr << "[" #arg "] :"; \
        dbg(arg);                    \
    } while (false)

template <typename... Ts>
void dbg(Ts... args) {
    (..., (std::cerr << ' ' << args));
    std::cerr << std::endl;
}
```

1.3 mod int

```cpp
template <int P>
struct Mint {
    int v = 0;

    // reflection
    template <typet = int>
    constexpr operator T() const {
        return v;
    }

    // constructor //
    constexpr Mint() = default;
    template <typet>
    constexpr Mint(T x) : v(x % P) {}
    constexpr int val() const { return v; }
    constexpr int mod() { return P; }

    // io //
    friend std::istream& operator>>(std::istream& is, Mint& x) {
        LL y;
        is >> y;
        x = y;
        return is;
    }
    friend std::ostream& operator<<(std::ostream& os, Mint x) { return os << x.v; }

    // comparision //
    friend constexpr bool operator==(const Mint& lhs, const Mint& rhs) { return lhs.v == rhs.v; }
    friend constexpr bool operator!=(const Mint& lhs, const Mint& rhs) { return lhs.v != rhs.v; }
    friend constexpr bool operator<(const Mint& lhs, const Mint& rhs) { return lhs.v < rhs.v; }
```

```
    friend constexpr bool operator<=(const Mint& lhs, const Mint& rhs) { return lhs.v <= rhs.v; }
    friend constexpr bool operator>(const Mint& lhs, const Mint& rhs) { return lhs.v > rhs.v; }
    friend constexpr bool operator>=(const Mint& lhs, const Mint& rhs) { return lhs.v >= rhs.v; }

    // arithmetic //
    template <typet>
    friend constexpr Mint power(Mint a, T n) {
        Mint ans = 1;
        while (n) {
            if (n & 1) ans *= a;
            a *= a;
            n >>= 1;
        }
        return ans;
    }
    friend constexpr Mint inv(const Mint& rhs) { return power(rhs, P - 2); }
    friend constexpr Mint operator+(const Mint& lhs, const Mint& rhs) {
        return lhs.val() + rhs.val() < P ? lhs.val() + rhs.val() : lhs.val() - P + rhs.val();
    }
    friend constexpr Mint operator-(const Mint& lhs, const Mint& rhs) {
        return lhs.val() < rhs.val() ? lhs.val() + P - rhs.val() : lhs.val() - rhs.val();
    }
    friend constexpr Mint operator*(const Mint& lhs, const Mint& rhs) {
        return static_cast<LL>(lhs.val()) * rhs.val() % P;
    }
    friend constexpr Mint operator/(const Mint& lhs, const Mint& rhs) { return lhs * inv(rhs); }
    Mint operator+() const { return *this; }
    Mint operator-() const { return Mint() - *this; }
    constexpr Mint& operator++() {
        v++;
        if (v == P) v = 0;
        return *this;
    }
    constexpr Mint& operator--() {
        if (v == 0) v = P;
        v--;
        return *this;
    }
    constexpr Mint& operator++(int) {
        Mint ans = *this;
        ++*this;
        return ans;
    }
    constexpr Mint operator--(int) {
        Mint ans = *this;
        --*this;
        return ans;
    }
    constexpr Mint& operator+=(const Mint& rhs) {
        v = v + rhs;
        return *this;
    }
    constexpr Mint& operator-=(const Mint& rhs) {
        v = v - rhs;
        return *this;
    }
    constexpr Mint& operator*=(const Mint& rhs) {
        v = v * rhs;
        return *this;
    }
    constexpr Mint& operator/=(const Mint& rhs) {
        v = v / rhs;
        return *this;
    }
};
using Z = Mint<998244353>;
```

2 Shell Scripts

2.1 checker.sh

Linux version.

```
#!/bin/bash

cd "$1"

g++ -o main -O2 -std=c++17 -DLOCAL main.cpp -ftrapv -fsanitize=address,undefined

for input in *.in; do
    output=${input%.*}.out
    answer=${input%.*}.ans

    ./main < $input > $ouput

    echo "case ${input%.*}: "
    echo "My: "
    cat $output
    echo "Answer: "
    cat $answer

done
```

Windows version.

```
@echo off

cd %1

del .\main.exe

g++ -o main.exe main.cpp -DLOCAL -std=c++17 -ftrapv

for %%i in (*.in) do (
    main.exe < %%i > %%~ni.out
    echo case %%~ni:
    echo My:
    type %%~ni.out
    echo Answer:
    type %%~ni.ans
)

cd ../shell
```

3 data structure

3.1 cartesian tree

3.2 segment tree

```
/* segment tree */
struct Info {
    /* 重载 operator+ */
};

struct Tag {
    /* 重载 operator== */
};

void infoApply(Info& a, int l, int r, const Tag& tag) {}

void tagApply(Tag& a, int l, int r, const Tag& tag) {}

template <class Info, class Tag>
class segTree {
#define ls i << 1
#define rs i << 1 | 1
#define mid ((l + r) >> 1)
#define lson ls, l, mid
```

```
#define rson rs, mid + 1, r

    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;

  public:
    segTree(const std::vector<Info>& init) : n(init.size() - 1) {
        assert(n > 0);
        info.resize(4 << std::__lg(n));
        tag.resize(4 << std::__lg(n));
        auto build = [&](auto dfs, int i, int l, int r) {
            if (l == r) {
                info[i] = init[l];
                return;
            }
            dfs(dfs, lson);
            dfs(dfs, rson);
            push_up(i);
        };
        build(build, 1, 1, n);
    }


  private:
    void push_up(int i) { info[i] = info[ls] + info[rs]; }


    template <class... T>
    void apply(int i, int l, int r, const T&... val) {
        ::infoApply(info[i], l, r, val...);
        ::tagApply(tag[i], l, r, val...);
    }

    void push_down(int i, int l, int r) {
        if (tag[i] == Tag{}) return;
        apply(lson, tag[i]);
        apply(rson, tag[i]);
        tag[i] = {};
    }

  public:
    template <class... T>
    void rangeApply(int ql, int qr, const T&... val) {
        auto dfs = [&](auto dfs, int i, int l, int r) {
            if (qr < l or r < ql) return;
            if (ql <= l and r <= qr) {
                apply(i, l, r, val...);
                return;
            }
            push_down(i, l, r);
            dfs(dfs, lson);
            dfs(dfs, rson);
            push_up(i);
        };
        dfs(dfs, 1, 1, n);
    }

    Info rangeAsk(int ql, int qr) {
        Info res{};
        auto dfs = [&](auto dfs, int i, int l, int r) {
            if (qr < l or r < ql) return;
            if (ql <= l and r <= qr) {
                res = res + info[i];
                return;
            }
            push_down(i, l, r);
            dfs(dfs, lson);
            dfs(dfs, rson);
        };
        dfs(dfs, 1, 1, n);
        return res;
    }

#undef rson
#undef lson
#undef mid
#undef rs
#undef ls
};
```

```
/* Problem: P3369 【模板】普通平衡树 */
struct node {
    int id, l, r;
```

```
    int ls, rs;
    int sum;

    node(int _id, int _l, int _r) : id(_id), l(_l), r(_r) {
        ls = rs = 0;
        sum = 0;
    }
};

/* segment tree */
int idx = 1;
std::vector<node> tree = {node{0, 0, 0}};

auto new_node = [&](int l, int r) -> int {
    tree.push_back(node(idx, l, r));
    return idx++;
};

auto push_up = [&](int u) -> void {
    tree[u].sum = 0;
    if (tree[u].ls) tree[u].sum += tree[tree[u].ls].sum;
    if (tree[u].rs) tree[u].sum += tree[tree[u].rs].sum;
};

auto build = [&]() { new_node(-inf, inf); };

std::function<void(int, int, int, int)> insert = [&](int u, int l, int r, int x) {
    if (l == r) {
        tree[u].sum++;
        return;
    }
    int mid = (l + r - 1) / 2;
    if (x <= mid) {
        if (!tree[u].ls) tree[u].ls = new_node(l, mid);
        insert(tree[u].ls, l, mid, x);
    } else {
        if (!tree[u].rs) tree[u].rs = new_node(mid + 1, r);
        insert(tree[u].rs, mid + 1, r, x);
    }
    push_up(u);
};

std::function<void(int, int, int, int)> remove = [&](int u, int l, int r, int x) {
    if (l == r) {
        if (tree[u].sum) tree[u].sum--;
        return;
    }
    int mid = (l + r - 1) / 2;
    if (x <= mid) {
        if (!tree[u].ls) return;
        remove(tree[u].ls, l, mid, x);
    } else {
        if (!tree[u].rs) return;
        remove(tree[u].rs, mid + 1, r, x);
    }
    push_up(u);
};

std::function<int(int, int, int, int)> get_rank_by_key = [&](int u, int l, int r, int x) -> int {
    if (l == r) {
        return 1;
    }
    int mid = (l + r - 1) / 2;
    int ans = 0;
    if (x <= mid) {
        if (!tree[u].ls) return 1;
        ans = get_rank_by_key(tree[u].ls, l, mid, x);
    } else {
        if (!tree[u].rs) return tree[tree[u].ls].sum + 1;
        if (!tree[u].ls) {
            ans = get_rank_by_key(tree[u].rs, mid + 1, r, x);
        } else {
            ans = get_rank_by_key(tree[u].rs, mid + 1, r, x) + tree[tree[u].ls].sum;
        }
    }
    return ans;
};

std::function<int(int, int, int, int)> get_key_by_rank = [&](int u, int l, int r, int x) -> int {
    if (l == r) {
        return l;
    }
    int mid = (l + r - 1) / 2;
    if (tree[u].ls) {
        if (x <= tree[tree[u].ls].sum) {
            return get_key_by_rank(tree[u].ls, l, mid, x);
        } else {
```

```
            return get_key_by_rank(tree[u].rs, mid + 1, r, x - tree[tree[u].ls].sum);
        }
    } else {
        return get_key_by_rank(tree[u].rs, mid + 1, r, x);
    }
};

std::function<int(int)> get_prev = [&](int x) -> int {
    int rank = get_rank_by_key(1, -inf, inf, x) - 1;
    debug(rank);
    return get_key_by_rank(1, -inf, inf, rank);
};

std::function<int(int)> get_next = [&](int x) -> int {
    debug(x + 1);
    int rank = get_rank_by_key(1, -inf, inf, x + 1);
    debug(rank);
    return get_key_by_rank(1, -inf, inf, rank);
};
```

```
/* Problem: P4556 [Vani有约会]雨天的尾巴 /【模板】线段树合并 */
struct node {
    int l, r, id;
    int ls, rs;
    int cnt, ans;

    node(int _id, int _l, int _r) : id(_id), l(_l), r(_r) {
        ls = rs = 0;
        cnt = ans = 0;
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    int n, m;
    std::cin >> n >> m;
    vvi e(n + 1);
    vi ans(n + 1);
    for (int i = 1; i < n; i++) {
        int u, v;
        std::cin >> u >> v;
        e[u].push_back(v);
        e[v].push_back(u);
    }

    // Segment tree //
    int idx = 1;
    vi rt(n + 1);
    std::vector<node> tree = {node{0, 0, 0}};

    auto new_node = [&](int l, int r) -> int {
        tree.push_back(node(idx, l, r));
        return idx++;
    };

    auto push_up = [&](int u) -> void {
        if (!tree[u].ls) {
            tree[u].cnt = tree[tree[u].rs].cnt;
            tree[u].ans = tree[tree[u].rs].ans;
        } else if (!tree[u].rs) {
            tree[u].cnt = tree[tree[u].ls].cnt;
            tree[u].ans = tree[tree[u].ls].ans;
        } else {
            if (tree[tree[u].rs].cnt > tree[tree[u].ls].cnt) {
                tree[u].cnt = tree[tree[u].rs].cnt;
                tree[u].ans = tree[tree[u].rs].ans;
            } else {
                tree[u].cnt = tree[tree[u].ls].cnt;
                tree[u].ans = tree[tree[u].ls].ans;
            }
        }
    };

    std::function<void(int, int, int, int, int)> modify = [&](int u, int l, int r, int x, int k) {
        if (l == r) {
            tree[u].cnt += k;
            tree[u].ans = l;
            return;
        }
        int mid = (l + r) >> 1;
        if (x <= mid) {
            if (!tree[u].ls) tree[u].ls = new_node(l, mid);
            modify(tree[u].ls, l, mid, x, k);
```

```
        } else {
            if (!tree[u].rs) tree[u].rs = new_node(mid + 1, r);
            modify(tree[u].rs, mid + 1, r, x, k);
        }
        push_up(u);
    };

    std::function<int(int, int, int, int)> merge = [&](int u, int v, int l, int r) -> int {
        // v 的信息传递给 u //
        if (!u) return v;
        if (!v) return u;
        if (l == r) {
            tree[u].cnt += tree[v].cnt;
            return u;
        }
        int mid = (l + r) >> 1;
        tree[u].ls = merge(tree[u].ls, tree[v].ls, l, mid);
        tree[u].rs = merge(tree[u].rs, tree[v].rs, mid + 1, r);
        push_up(u);
        return u;
    };

    // LCA //

    for (int i = 1; i <= n; i++) {
        rt[i] = idx;
        new_node(1, 100000);
    }

    for (int i = 1; i <= m; i++) {
        int u, v, w;
        std::cin >> u >> v >> w;
        int lca = LCA(u, v);
        modify(rt[u], 1, 100000, w, 1);
        modify(rt[v], 1, 100000, w, 1);
        modify(rt[lca], 1, 100000, w, -1);
        if (father[lca][0]) {
            modify(rt[father[lca][0]], 1, 100000, w, -1);
        }
    }

    // dfs //
    std::function<void(int, int)> Dfs = [&](int u, int fa) {
        for (auto v : e[u]) {
            if (v == fa) continue;
            Dfs(v, u);
            merge(rt[u], rt[v], 1, 100000);
        }
        ans[u] = tree[rt[u]].ans;
        if (tree[rt[u]].cnt == 0) ans[u] = 0;
    };

    Dfs(1, 0);

    for (int i = 1; i <= n; i++) {
        std::cout << ans[i] << endl;
    }

    return 0;
}
```

3.3 hjt segment tree

```
/* 洛谷 P3919 【模板】可持久化线段树 1（可持久化数组）*/
struct node {
    int l, r, key;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    int n, m;
    std::cin >> n >> m;
    vi a(n + 1);
    for (int i = 1; i <= n; i++) {
        std::cin >> a[i];
    }

    // hjt segment tree //
    int idx = 0;
```

```
    vi root(m + 1);
    std::vector<node> tr(n * 25);

    std::function<int(int, int)> build = [&](int l, int r) -> int {
        int p = ++idx;
        if (l == r) {
            tr[p].key = a[l];
            return p;
        }
        int mid = (l + r) >> 1;
        tr[p].l = build(l, mid);
        tr[p].r = build(mid + 1, r);
        return p;
    };

    std::function<int(int, int, int, int, int)> modify = [&](int p, int l, int r, int k,
                                                             int x) -> int {
        int q = ++idx;
        tr[q].l = tr[p].l, tr[q].r = tr[p].r;
        if (tr[q].l == tr[q].r) {
            tr[q].key = x;
            return q;
        }
        int mid = (l + r) >> 1;
        if (k <= mid) {
            tr[q].l = modify(tr[q].l, l, mid, k, x);
        } else {
            tr[q].r = modify(tr[q].r, mid + 1, r, k, x);
        }
        return q;
    };

    std::function<int(int, int, int, int)> query = [&](int p, int l, int r, int k) -> int {
        if (tr[p].l == tr[p].r) {
            return tr[p].key;
        }
        int mid = (l + r) >> 1;
        if (k <= mid) {
            return query(tr[p].l, l, mid, k);
        } else {
            return query(tr[p].r, mid + 1, r, k);
        }
    };
    root[0] = build(1, n);

    for (int i = 1; i <= m; i++) {
        int op, ver, k, x;
        std::cin >> ver >> op;
        if (op == 1) {
            std::cin >> k >> x;
            root[i] = modify(root[ver], 1, n, k, x);
        } else {
            std::cin >> k;
            root[i] = root[ver];
            std::cout << query(root[ver], 1, n, k) << endl;
        }
    }
    return 0;
}
```

```
/* 洛谷P3834 【模板】可持久化线段树 2 */

struct node {
    int l, r, cnt;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    int n, m;
    std::cin >> n >> m;
    vi a(n + 1), v;
    for (int i = 1; i <= n; i++) {
        std::cin >> a[i];
        v.push_back(a[i]);
    }
    std::sort(all(v));
    v.erase(unique(all(v)), v.end());
    auto find = [&](int x) -> int { return std::lower_bound(all(v), x) - v.begin() + 1; };

    // hjt segment tree //
    std::vector<node>(n * 25);
    vi root(n + 1);
    int idx = 0;
```

```
    std::function<int(int, int)> build = [&](int l, int r) -> int {
        int p = ++idx;
        if (l == r) return p;
        int mid = (l + r) >> 1;
        tr[p].l = build(l, mid), tr[p].r = build(mid + 1, r);
        return p;
    };

    std::function<int(int, int, int, int)> modify = [&](int p, int l, int r, int x) -> int {
        int q = ++idx;
        tr[q] = tr[p];
        if (tr[q].l == tr[q].r) {
            tr[q].cnt++;
            return q;
        }
        int mid = (l + r) >> 1;
        if (x <= mid) {
            tr[q].l = modify(tr[q].l, l, mid, x);
        } else {
            tr[q].r = modify(tr[q].r, mid + 1, r, x);
        }
        tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
        return q;
    };

    std::function<int(int, int, int, int, int)> query = [&](int p, int q, int l, int r,
                                                             int x) -> int {
        if (l == r) return l;
        int cnt = tr[tr[p].l].cnt - tr[tr[q].l].cnt;
        int mid = (l + r) >> 1;
        if (x <= cnt) {
            return query(tr[p].l, tr[q].l, l, mid, x);
        } else {
            return query(tr[p].r, tr[q].r, mid + 1, r, x - cnt);
        }
    };
    root[0] = build(1, v.size());

    for (int i = 1; i <= n; i++) {
        root[i] = modify(root[i - 1], 1, v.size(), find(a[i]));
    }
    for (int i = 1; i <= m; i++) {
        int l, r, k;
        std::cin >> l >> r >> k;
        std::cout << v[query(root[r], root[l - 1], 1, v.size(), k) - 1] << endl;
    }
    return 0;
}
```

3.4 LiChao tree

3.5 treap

```
/* fhq treap */
struct node {
    node *ch[2];
    int key, val;
    int cnt, size;

    node(int _key) : key(_key), cnt(1), size(1) {
        ch[0] = ch[1] = nullptr;
        val = rand();
    }

    // node(node *_node) {
    // key = _node->key, val = _node->val, cnt = _node->cnt, size = _node->size;
    // }

    inline void push_up() {
        size = cnt;
        if (ch[0] != nullptr) size += ch[0]->size;
        if (ch[1] != nullptr) size += ch[1]->size;
    }
};

struct treap {
#define _2 second.first
#define _3 second.second

    node *root;
```

```
28 |
29 |     pair<node *, node *> split(node *p, int key) {
30 |         if (p == nullptr) return {nullptr, nullptr};
31 |         if (p->key <= key) {
32 |             auto temp = split(p->ch[1], key);
33 |             p->ch[1] = temp.first;
34 |             p->push_up();
35 |             return {p, temp.second};
36 |         } else {
37 |             auto temp = split(p->ch[0], key);
38 |             p->ch[0] = temp.second;
39 |             p->push_up();
40 |             return {temp.first, p};
41 |         }
42 |     }
43 |
44 |     pair<node *, pair<node *, node *> > split_by_rank(node *p, int rank) {
45 |         if (p == nullptr) return {nullptr, {nullptr, nullptr}};
46 |         int ls_size = p->ch[0] == nullptr ? 0 : p->ch[0]->size;
47 |         if (rank <= ls_size) {
48 |             auto temp = split_by_rank(p->ch[0], rank);
49 |             p->ch[0] = temp._3;
50 |             p->push_up();
51 |             return {temp.first, {temp._2, p}};
52 |         } else if (rank <= ls_size + p->cnt) {
53 |             node *lt = p->ch[0];
54 |             node *rt = p->ch[1];
55 |             p->ch[0] = p->ch[1] = nullptr;
56 |             return {lt, {p, rt}};
57 |         } else {
58 |             auto temp = split_by_rank(p->ch[1], rank - ls_size - p->cnt);
59 |             p->ch[1] = temp.first;
60 |             p->push_up();
61 |             return {p, {temp._2, temp._3}};
62 |         }
63 |     }
64 |
65 |     node *merge(node *u, node *v) {
66 |         if (u == nullptr && v == nullptr) return nullptr;
67 |         if (u != nullptr && v == nullptr) return u;
68 |         if (v != nullptr && u == nullptr) return v;
69 |         if (u->val < v->val) {
70 |             u->ch[1] = merge(u->ch[1], v);
71 |             u->push_up();
72 |             return u;
73 |         } else {
74 |             v->ch[0] = merge(u, v->ch[0]);
75 |             v->push_up();
76 |             return v;
77 |         }
78 |     }
79 |
80 |     void insert(int key) {
81 |         auto temp = split(root, key);
82 |         auto l_tr = split(temp.first, key - 1);
83 |         node *new_node;
84 |         if (l_tr.second == nullptr) {
85 |             new_node = new node(key);
86 |         } else {
87 |             l_tr.second->cnt++;
88 |             l_tr.second->push_up();
89 |         }
90 |         node *l_tr_combined = merge(l_tr.first, l_tr.second == nullptr ? new_node : l_tr.second);
91 |         root = merge(l_tr_combined, temp.second);
92 |     }
93 |
94 |     void remove(int key) {
95 |         auto temp = split(root, key);
96 |         auto l_tr = split(temp.first, key - 1);
97 |         if (l_tr.second->cnt > 1) {
98 |             l_tr.second->cnt--;
99 |             l_tr.second->push_up();
100|             l_tr.first = merge(l_tr.first, l_tr.second);
101|         } else {
102|             if (temp.first == l_tr.second) temp.first = nullptr;
103|             delete l_tr.second;
104|             l_tr.second = nullptr;
105|         }
106|         root = merge(l_tr.first, temp.second);
107|     }
108|
109|     int get_rank_by_key(node *p, int key) {
110|         auto temp = split(p, key - 1);
111|         int ret = (temp.first == nullptr ? 0 : temp.first->size) + 1;
112|         root = merge(temp.first, temp.second);
113|         return ret;
114|     }
```

```

    int get_key_by_rank(node *p, int rank) {
        auto temp = split_by_rank(p, rank);
        int ret = temp._2->key;
        root = merge(temp.first, merge(temp._2, temp._3));
        return ret;
    }

    int get_prev(int key) {
        auto temp = split(root, key - 1);
        int ret = get_key_by_rank(temp.first, temp.first->size);
        root = merge(temp.first, temp.second);
        return ret;
    }

    int get_nex(int key) {
        auto temp = split(root, key);
        int ret = get_key_by_rank(temp.second, 1);
        root = merge(temp.first, temp.second);
        return ret;
    }
} tr;
```

3.6 splay

```
/* 洛谷 P3391 【模板】文艺平衡树 */
struct node {
    int ch[2], fa, key;
    int siz, flag;

    void init(int _fa, int _key) { fa = _fa, key = _key, siz = 1; }
};

struct splay {
    node tr[N];
    int n, root, idx;

    bool get(int u) { return u == tr[tr[u].fa].ch[1]; }

    void pushup(int u) { tr[u].siz = tr[tr[u].ch[0]].siz + tr[tr[u].ch[1]].siz + 1; }

    void pushdown(int u) {
        if (tr[u].flag) {
            std::swap(tr[u].ch[0], tr[u].ch[1]);
            tr[tr[u].ch[0]].flag ^= 1, tr[tr[u].ch[1]].flag ^= 1;
            tr[u].flag = 0;
        }
    }

    void rotate(int x) {
        int y = tr[x].fa, z = tr[y].fa;
        int op = get(x);
        tr[y].ch[op] = tr[x].ch[op ^ 1];
        if (tr[x].ch[op ^ 1]) tr[tr[x].ch[op ^ 1]].fa = y;
        tr[x].ch[op ^ 1] = y;
        tr[y].fa = x, tr[x].fa = z;
        if (z) tr[z].ch[y == tr[z].ch[1]] = x;
        pushup(y), pushup(x);
    }

    void opt(int u, int k) {
        for (int f = tr[u].fa; f = tr[u].fa, f != k; rotate(u)) {
            if (tr[f].fa != k) rotate(get(u) == get(f) ? f : u);
        }
        if (k == 0) root = u;
    }

    void output(int u) {
        pushdown(u);
        if (tr[u].ch[0]) output(tr[u].ch[0]);
        if (tr[u].key >= 1 && tr[u].key <= n) {
            std::cout << tr[u].key << ' ';
        }
        if (tr[u].ch[1]) output(tr[u].ch[1]);
    }

    void insert(int key) {
        idx++;
        tr[idx].ch[0] = root;
        tr[idx].init(0, key);
        tr[root].fa = idx;
        root = idx;
```

```
        pushup(idx);
    }

    int kth(int k) {
        int u = root;
        while (1) {
            pushdown(u);
            if (tr[u].ch[0] && k <= tr[tr[u].ch[0]].siz) {
                u = tr[u].ch[0];
            } else {
                k -= tr[tr[u].ch[0]].siz + 1;
                if (k <= 0) {
                    opt(u, 0);
                    return u;
                } else {
                    u = tr[u].ch[1];
                }
            }
        }
    }

} splay;

int n, m, l, r;
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    std::cin >> n >> m;
    splay.n = n;
    splay.insert(-inf);
    for (int i = 1; i <= n; i++) splay.insert(i);
    splay.insert(inf);
    for (int i = 1; i <= m; i++) {
        std::cin >> l >> r;
        l = splay.kth(l), r = splay.kth(r + 2);
        splay.opt(l, 0), splay.opt(r, l);
        splay.tr[splay.tr[r].ch[0]].flag ^= 1;
    }
    splay.output(splay.root);
    return 0;
}
```

```
/* 洛谷 P3369 【模板】普通平衡树 */

struct node {
    int ch[2], fa, key, siz, cnt;

    void init(int _fa, int _key) { fa = _fa, key = _key, siz = cnt = 1; }

    void clear() { ch[0] = ch[1] = fa = key = siz = cnt = 0; }
};

struct splay {
    node tr[N];
    int n, root, idx;

    bool get(int u) { return u == tr[tr[u].fa].ch[1]; }

    void pushup(int u) { tr[u].siz = tr[tr[u].ch[0]].siz + tr[tr[u].ch[1]].siz + tr[u].cnt; }

    void rotate(int x) {
        int y = tr[x].fa, z = tr[y].fa;
        int op = get(x);
        tr[y].ch[op] = tr[x].ch[op ^ 1];
        if (tr[x].ch[op ^ 1]) tr[tr[x].ch[op ^ 1]].fa = y;
        tr[x].ch[op ^ 1] = y;
        tr[y].fa = x, tr[x].fa = z;
        if (z) tr[z].ch[y == tr[z].ch[1]] = x;
        pushup(y), pushup(x);
    }

    void opt(int u, int k) {
        for (int f = tr[u].fa; f = tr[u].fa, f != k; rotate(u)) {
            if (tr[f].fa != k) {
                rotate(get(u) == get(f) ? f : u);
            }
        }
        if (k == 0) root = u;
    }

    void insert(int key) {
        if (!root) {
            idx++;
            tr[idx].init(0, key);
```

```
            root = idx;
            return;
        }
        int u = root, f = 0;
        while (1) {
            if (tr[u].key == key) {
                tr[u].cnt++;
                pushup(u), pushup(f);
                opt(u, 0);
                break;
            }
            f = u, u = tr[u].ch[tr[u].key < key];
            if (!u) {
                idx++;
                tr[idx].init(f, key);
                tr[f].ch[tr[f].key < key] = idx;
                pushup(idx), pushup(f);
                opt(idx, 0);
                break;
            }
        }
    }

    // 返回节点编号 //
    int kth(int rank) {
        int u = root;
        while (1) {
            if (tr[u].ch[0] && rank <= tr[tr[u].ch[0]].siz) {
                u = tr[u].ch[0];
            } else {
                rank -= tr[tr[u].ch[0]].siz + tr[u].cnt;
                if (rank <= 0) {
                    opt(u, 0);
                    return u;
                } else {
                    u = tr[u].ch[1];
                }
            }
        }
    }

    // 返回排名 //
    int nlt(int key) {
        int rank = 0, u = root;
        while (1) {
            if (tr[u].key > key) {
                u = tr[u].ch[0];
            } else {
                rank += tr[tr[u].ch[0]].siz;
                if (tr[u].key == key) {
                    opt(u, 0);
                    return rank + 1;
                }
                rank += tr[u].cnt;
                if (tr[u].ch[1]) {
                    u = tr[u].ch[1];
                } else {
                    return rank + 1;
                }
            }
        }
    }

    int get_prev(int key) { return kth(nlt(key) - 1); }

    int get_next(int key) { return kth(nlt(key + 1)); }

    void remove(int key) {
        nlt(key);
        if (tr[root].cnt > 1) {
            tr[root].cnt--;
            pushup(root);
            return;
        }
        int u = root, l = get_prev(key);
        tr[tr[u].ch[1]].fa = l;
        tr[l].ch[1] = tr[u].ch[1];
        tr[u].clear();
        pushup(root);
    }

    void output(int u) {
        if (tr[u].ch[0]) output(tr[u].ch[0]);
        std::cout << tr[u].key << ' ';
        if (tr[u].ch[1]) output(tr[u].ch[1]);
    }

```

```
} splay;

int n, op, x;
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    splay.insert(-inf), splay.insert(inf);

    std::cin >> n;
    for (int i = 1; i <= n; i++) {
        std::cin >> op >> x;
        if (op == 1) {
            splay.insert(x);
        } else if (op == 2) {
            splay.remove(x);
        } else if (op == 3) {
            std::cout << splay.nlt(x) - 1 << '\n';
        } else if (op == 4) {
            std::cout << splay.tr[splay.kth(x + 1)].key << '\n';
        } else if (op == 5) {
            std::cout << splay.tr[splay.get_prev(x)].key << '\n';
        } else if (op == 6) {
            std::cout << splay.tr[splay.get_next(x)].key << '\n';
        }
    }
    return 0;
}
```

3.7 Link Cut Tree

3.8 ODT

3.9 tree in tree

4 string

4.1 KMP

```
auto getNext = [&](const std::string& s) -> vi {
    int n = s.length();
    vi next(n);
    for (int i = 1; i < n; i++) {
        int j = next[i - 1];
        while (j > 0 and s[i] != s[j]) j = next[j - 1];
        if (s[i] == s[j]) j++;
        next[i] = j;
    }
    return next;
};
```

4.2 z-function

```
auto zFunction = [&](const std::string& s) -> vi {
    int n = s.size();
    vi z(n);
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r and z[i - l] < r - i + 1) {
            z[i] = z[i - l];
        } else {
            z[i] = std::max(0, r - i + 1);
            while (z[i] + i < n and s[z[i]] == s[z[i] + i]) z[i]++;
        }
        if (z[i] + i - 1 > r) {
            l = i;
            r = z[i] + i - 1;
        }
    }
    return z;
```

```
};
```

4.3 Manacher

```
```

4.4 AC automaton

4.5 PAM

```
/* PAM @ ddl */
std::vector<node> tr;
std::vector<int> stk;
auto newnode = [&](int len) {
    tr.emplace_back();
    tr.back().len = len;
    return (int) tr.size() - 1;
};
auto PAMinit = [&]() {
    newnode(0), tr.back().fail = 1;
    newnode(-1), tr.back().fail = 0;
    stk.push_back(-1);
};
PAMinit();
auto getfail = [&](int v) {
    while (stk.end()[-2 - tr[v].len] != stk.back()) {
        v = tr[v].fail;
    }
    return v;
};
auto insert = [&](int last, int c, int cnt) {
    stk.emplace_back(c);
    int x = getfail(last);
    if (!tr[x].ch[c]) {
        int u = newnode(tr[x].len + 2);
        tr[u].fail = tr[getfail(tr[x].fail)].ch[c];
        tr[x].ch[c] = u;
        /* tr[u].size = tr[tr[u].fail].size + 1; */
        /* Can be used to count the number of types of palindromic strings ending at the current
         * position */
    }
    tr[tr[x].ch[c]].size += cnt;
    return tr[x].ch[c];
};
auto build = [&]() {  /* DP on fail tree */
    int ans = 0;
    for (int i = (int) tr.size() - 1; i > 1; i--) {
        tr[tr[i].fail].size += tr[i].size;
        /* options */
    }
    return ans;
};
/* PAM */
int ans = 0, last = 0;
for (int i = 0; i < n; i++) {
    last = insert(last, s[i] - 'a', 1);
}
```

4.6 suffix array

```
/* suffix array and ST table @ jiangly */
auto suffixArray = [&](const std::string& s) {
    int n = s.length();
    vi sa(n), rk(n);
    std::iota(all(sa), 0);
    std::sort(all(sa), [&](int a, int b) { return s[a] < s[b]; });
    rk[sa[0]] = 0;
    for (int i = 1; i < n; ++i) {
        rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
    }
    int k = 1;
    vi tmp(n), cnt(n);
```

```
    tmp.reserve(n);
    while (rk[sa[n - 1]] < n - 1) {
        tmp.clear();
        for (int i = 0; i < k; ++i) tmp.push_back(n - k + i);
        for (const auto& i : sa) {
            if (i >= k) tmp.push_back(i - k);
        }
        std::fill(all(cnt), 0);
        for (int i = 0; i < n; i++) cnt[rk[i]]++;
        for (int i = 1; i < n; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--) sa[--cnt[rk[tmp[i]]]] = tmp[i];
        std::swap(rk, tmp);
        rk[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] or sa[i - 1] + k == n or
                                         tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
        }
        k *= 2;
    }
    vi height(n);
    for (int i = 0, j = 0; i < n; ++i) {
        if (rk[i] == 0) continue;
        if (j) --j;
        while (s[i + j] == s[sa[rk[i] - 1] + j]) ++j;
        height[rk[i]] = j;
    }
    return std::make_tuple(sa, rk, height);
};
auto [sa, rk, height] = suffixArray(s);
vvi f(n, vi(30, inf));
vi Log2(n);
auto init = [&]() -> void {
    for (int i = 0; i < n; i++) {
        f[i][0] = height[i];
        if (i > 1) Log2[i] = Log2[i / 2] + 1;
    };
    int t = Log2.back();
    for (int j = 1; j <= t; j++) {
        for (int i = 0; i <= n - (1 << j); i++) {
            f[i][j] = std::min(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
        }
    }
};
init();
auto query = [&](int l, int r) -> int {
    int t = Log2[r - l + 1];
    return std::min(f[l][t], f[r - (1 << t) + 1][t]);
};
auto lcp = [&](int i, int j) {
    i = rk[i], j = rk[j];
    if (i > j) std::swap(i, j);
    return query(i + 1, j);
};
```

5 graph