Beijing Normal University
School of Mathematics

Template

app1eDog

2024 年 11 月 1 日

目录

1 hpp

1.1 heading

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

// using namespace std;

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 inf = 0x3f3f3f3f;
constexpr LL INF = 1e18;
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; }

void solut() {
    ;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    int t = 1;
    std::cin >> t;
    while (t--) {
        solut();
    }
    return 0;
}
```

1.2 debug.h

```
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;
}
```

2 shell scripts

2.1 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
```

2.2 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 stack

```
vi stk;
for (int i = 1; i <= n; i++){
    while (!stk.empty() and stk.back() > a[i]) {
        stk.pop_back();
    }
    stk.pop_back(a[i]);
}
```

3.2 queue

```
std::deque<int> q;
for (int i = 1; i <= n; i++) {
    while (!q.empty and a[q.back()] >= a[i]) p.pop_back();
    if (!q.empty() and i - q.front() >= k) q.pop_front();
    q.push_back(i);
}
```

3.3 DSU

```
/* DSU */
vi fa(n + 1);
std::iota(all(fa), 0);
std::function<int(int)> find = [&] (int x) -> int{
    return x == fa[x] ? x : fa[x] = find(fa[x]);
};
auto merge = [&] (int x, int y) -> void{
    x = find(x), y = find(y);
    if (x == y) return;
    // operations //
    fa[y] = x;
};
```

3.4 spare table

一维

```
/* spare table */
int B = 30;
vvi f(n + 1, vi(B));
vi Log2(n + 1);
auto init = [&]() -> void {
    for (int i = 1; i <= n; i++) {
        f[i][0] = a[i];
        if (i > 1) Log2[i] = Log2[i / 2] + 1;
    };
    int t = Log2[n];
    for (int j = 1; j <= t; j++) {
        for (int i = 1; i <= n - (1 << j) + 1; i++) {
            f[i][j] = std::max(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::max(f[l][t], f[r - (1 << t) + 1][t]);
};
```

二维

```
/* spare table */
intB = 30;
std::vector f(n + 1, std::vector<std::array<std::array<int, B>, B>>(m + 1));
vi Log2(n + 1);
auto init = [&]() -> void {
    for (int i = 2; i <= std::max(n, m); i++) {
        Log2[i] = Log2[i / 2] + 1;
    }
    for (int i = 2; i <= n; i++) {
        for (int j = 2; j <= m; j++) {
            f[i][j][0][0] = a[i][j];
        }
    }
    for (int ki = 0; ki <= Log2[n]; ki++) {
        for (int kj = 0; kj <= Log2[n]; kj++) {
            if (!ki && !kj) continue;
            for (int i = 1; i <= n - (1 << ki) + 1; i++) {
                for (int j = 1; j <= m - (1 << kj) + 1; j++) {
                    if (ki) {
                        f[i][j][ki][kj] =
                            std::max(f[i][j][ki - 1][kj], f[i + (1 << (ki - 1))][j][ki - 1][kj]);
                    } else {
                        f[i][j][ki][kj] =
                            std::max(f[i][j][ki][kj - 1], f[i][j + (1 << (kj - 1))][ki][kj - 1]);
                    }
                }
            }
        }
    }
};
init();
auto query = [&](int x1, int y1, int x2, int y2) -> int {
    int ki = Log2[x2 - x1 + 1], kj = Log2[y2 - y1 + 1];
    int t1 = f[x1][y1][ki][kj];
    int t2 = f[x2 - (1 << ki) + 1][y1][ki][kj];
    int t3 = f[x1][y2 - (1 << kj) + 1][ki][kj];
    int t4 = f[x2 - (1 << ki) + 1][y2 - (1 << kj) + 1][ki][kj];
    return std::max({t1, t2, t3, t4});
};
```

3.5 Cartesian tree

一种特殊的平衡树, 用元素的值作为平衡点节点的 val, 元素的下标作为 key.

```
/* cartesian tree */
vi ls(n + 1), rs(n + 1), stk(n + 1);
int top = 1;
for (int i = 1; i <= n; i++) {
    int k = top;
    while (k and a[stk[k]] > a[i]) k--;
    if (k) rs[stk[k]] = i;
    if (k < top) ls[i] = stk[k + 1];
    stk[++k] = i;
    top = k;
}
```

3.6 segment tree

TODO

3.7 persistent segment tree

单点修改, 版本拷贝

n 个数, m 次操作, 操作分别为

1. v_i 1 loc_i $value_i$: 将第 v_i 个版本的 $a[loc_i]$ 修改为 $value_i$,
2. v_i 2 loc_i: 拷贝第 v_i 个版本, 并查询该版本的 $a[loc_i]$.

```cpp
// 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) << '\n';
        }
    }

    return 0;
}
```

区间第 k 小

长度为 n 的序列 a, m 次查询, 每次查询 $[l,r]$ 中的第 k 小值.

```cpp
// 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] << '\n';
    }

    return 0;
}
```

3.8 treap

fhq treap

n 次操作, 操作分为如下 6 种:

1. 插入数 x;
2. 删除数 x (若有多个相同的数，只删除一个);
3. 查询数 x 的排名 (排名定义为小于 x 的数的个数 + 1);
4. 查询排名为 x 的数;
5. 求 x 的前驱 (前驱定义为小于 x 的最大数);
6. 求 x 的后继 (后继定义为大于 x 的最小数).

```
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;

    pair<node *, node *> split(node *p, int key) {
        if (p == nullptr) return {nullptr, nullptr};
        if (p->key <= key) {
            auto temp = split(p->ch[1], key);
            p->ch[1] = temp.first;
            p->push_up();
            return {p, temp.second};
        } else {
            auto temp = split(p->ch[0], key);
            p->ch[0] = temp.second;
            p->push_up();
            return {temp.first, p};
        }
    }

    pair<node *, pair<node *, node *> > split_by_rank(node *p, int rank) {
        if (p == nullptr) return {nullptr, {nullptr, nullptr}};
        int ls_size = p->ch[0] == nullptr ? 0 : p->ch[0]->size;
        if (rank <= ls_size) {
            auto temp = split_by_rank(p->ch[0], rank);
            p->ch[0] = temp._3;
            p->push_up();
            return {temp.first, {temp._2, p}};
        } else if (rank <= ls_size + p->cnt) {
            node *lt = p->ch[0];
            node *rt = p->ch[1];
            p->ch[0] = p->ch[1] = nullptr;
            return {lt, {p, rt}};
        } else {
            auto temp = split_by_rank(p->ch[1], rank - ls_size - p->cnt);
            p->ch[1] = temp.first;
            p->push_up();
```

```cpp
            return {p, {temp._2, temp._3}};
        }
    }

    node *merge(node *u, node *v) {
        if (u == nullptr && v == nullptr) return nullptr;
        if (u != nullptr && v == nullptr) return u;
        if (v != nullptr && u == nullptr) return v;
        if (u->val < v->val) {
            u->ch[1] = merge(u->ch[1], v);
            u->push_up();
            return u;
        } else {
            v->ch[0] = merge(u, v->ch[0]);
            v->push_up();
            return v;
        }
    }

    void insert(int key) {
        auto temp = split(root, key);
        auto l_tr = split(temp.first, key - 1);
        node *new_node;
        if (l_tr.second == nullptr) {
            new_node = new node(key);
        } else {
            l_tr.second->cnt++;
            l_tr.second->push_up();
        }
        node *l_tr_combined = merge(l_tr.first, l_tr.second == nullptr ? new_node : l_tr.second);
        root = merge(l_tr_combined, temp.second);
    }

    void remove(int key) {
        auto temp = split(root, key);
        auto l_tr = split(temp.first, key - 1);
        if (l_tr.second->cnt > 1) {
            l_tr.second->cnt--;
            l_tr.second->push_up();
            l_tr.first = merge(l_tr.first, l_tr.second);
        } else {
            if (temp.first == l_tr.second) temp.first = nullptr;
            delete l_tr.second;
            l_tr.second = nullptr;
        }
        root = merge(l_tr.first, temp.second);
    }

    int get_rank_by_key(node *p, int key) {
        auto temp = split(p, key - 1);
        int ret = (temp.first == nullptr ? 0 : temp.first->size) + 1;
        root = merge(temp.first, temp.second);
        return ret;
    }

    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;
    }
};

treap tr;

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

    srand(time(0));

    int n;
```

```
    cin >> n;
    while (n--) {
        int op, x;
        cin >> op >> x;
        if (op == 1) {
            tr.insert(x);
        } else if (op == 2) {
            tr.remove(x);
        } else if (op == 3) {
            cout << tr.get_rank_by_key(tr.root, x) << '\n';
        } else if (op == 4) {
            cout << tr.get_key_by_rank(tr.root, x) << '\n';
        } else if (op == 5) {
            cout << tr.get_prev(x) << '\n';
        } else {
            cout << tr.get_nex(x) << '\n';
        }
    }
    return 0;
}
```

用 01 trie 实现的一种方式

同样的题目, 注意使用 01 trie 只能存在非负数. 速度能快不少, 但只能单点操作, 而且有点费空间.

```
// 洛谷 P3369 【模板】普通平衡树

struct Treap {
    int id = 1, maxlog = 25;
    int ch[N * 25][2], siz[N * 25];

    int newnode() {
        id++;
        ch[id][0] = ch[id][1] = siz[id] = 0;
        return id;
    }

    void merge(int key, int cnt) {
        int u = 1;
        for (int i = maxlog - 1; i >= 0; i--) {
            int v = (key >> i) & 1;
            if (!ch[u][v]) ch[u][v] = newnode();
            u = ch[u][v];
            siz[u] += cnt;
        }
    }

    int get_key_by_rank(int rank) {
        int u = 1, key = 0;
        for (int i = maxlog - 1; i >= 0; i--) {
            if (siz[ch[u][0]] >= rank) {
                u = ch[u][0];
            } else {
                key |= (1 << i);
                rank -= siz[ch[u][0]];
                u = ch[u][1];
            }
        }
        return key;
    }

    int get_rank_by_key(int rank) {
        int key = 0;
        int u = 1;
        for (int i = maxlog - 1; i >= 0; i--) {
            if ((rank >> i) & 1) {
                key += siz[ch[u][0]];
                u = ch[u][1];
            } else {
                u = ch[u][0];
            }
            if (!u) break;
        }
        return key;
    }

    int get_prev(int x) { return get_key_by_rank(get_rank_by_key(x)); }
    int get_next(int x) { return get_key_by_rank(get_rank_by_key(x + 1) + 1); }
} treap;

const int num = 1e7;
int n, op, x;
```

```cpp

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

    std::cin >> n;
    for (int i = 1; i <= n; i++) {
        std::cin >> op >> x;
        if (op == 1) {
            treap.merge(x + num, 1);
        } else if (op == 2) {
            treap.merge(x + num, -1);
        } else if (op == 3) {
            std::cout << treap.get_rank_by_key(x + num) + 1 << '\n';
        } else if (op == 4) {
            std::cout << treap.get_key_by_rank(x) - num << '\n';
        } else if (op == 5) {
            std::cout << treap.get_prev(x + num) - num << '\n';
        } else if (op == 6) {
            std::cout << treap.get_next(x + num) - num << '\n';
        }
    }
    return 0;
}
```

3.9 splay

文艺平衡树

初始为 1 到 n 的序列, m 次操作, 每次将序列下标为 $[l \sim r]$ 的区间翻转.

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

    return 0;
}
```

普通平衡树

```
// 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;
```

```cpp
        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 << endl;
        } else if (op == 4) {
            std::cout << splay.tr[splay.kth(x + 1)].key << endl;
        } else if (op == 5) {
            std::cout << splay.tr[splay.get_prev(x)].key << endl;
        } else if (op == 6) {
            std::cout << splay.tr[splay.get_next(x)].key << endl;
        }
    }

    return 0;
}
```

4 string

4.1 kmp

```
/* kmp */
auto kmp = [&](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

```
/* exkmp */
auto exkmp = [&](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

TODO

AC automaton

```
/* AC auto */
int cnt = 0;
const int N = 2e5 + 10;
static std::array<std::array<int, 26>, N> tr;
static std::array<int, N> exist, fail, ans, point;
vi order;

auto insert = [&](const auto& s) {
    int p = 0;
    for (const auto& ch : s) {
        int c = ch - 'a';
        if (!tr[p][c]) tr[p][c] = ++cnt;
        p = tr[p][c];
    }
    exist[p]++;
    return p;
};

auto build = [&]() {
    std::queue<int> q;
    for (int i = 0; i < 26; i++) {
        if (tr[0][i]) q.push(tr[0][i]);
    }
    while (!q.empty()) {
        auto u = q.front();
        q.pop();
        order.push_back(u);
        for (int i = 0; i < 26; i++) {
            if (tr[u][i]) {
                fail[tr[u][i]] = tr[fail[u]][i];
                q.push(tr[u][i]);
            } else {
                tr[u][i] = tr[fail[u]][i];
            }
        }
    }
};

auto query = [&](const auto& s) {
    int p = 0;
    for (const auto ch : s) {
        p = tr[p][ch - 'a'];
        ans[p]++;
    }
    return;
};

void solve (){
    for (int i = 0; i < n; i++) {
        point[i] = insert(t);
    }
    build();
    query(s);
    /* fail 树上子树求和 */
    reverse(all(order));
    for (const auto& i : order) ans[fail[i]] += ans[i];
}
```

4.4 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);
}
```

4.5 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]);
```

```cpp
            }
            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);
};
```

4.6 trie

普通字典树 (单词匹配)

```cpp
/* trie */
int cnt;
std::vector<std::array<int, 26>> trie(n + 1);
vi exist(n + 1);
auto insert = [&](const std::string& s) -> void {
    int p = 0;
    for (const auto ch : s) {
        int c = ch - 'a';
        if (!trie[p][c]) trie[p][c] = ++cnt;
        p = trie[p][c];
    }
    exist[p] = true;
};
auto find = [&](const string& s) -> bool {
    int p = 0;
    for (const auto ch : s) {
        int c = ch - 'a';
        if (!trie[p][c]) return false;
        p = trie[p][c];
    }
    return exist[p];
};
```

01 字典树 (求最大异或值)

给定 n 个数, 取两个数进行异或运算, 求最大异或值.

```cpp
/* trie */
int cnt = 0;
std::vector<std::array<int, 2>> trie(N);
auto insert = [&](int x) -> void {
    int p = 0;
    for (int i = 30; i >= 0; i--) {
        int c = (x >> i) & 1;
```

```
        if (!trie[p][c]) trie[p][c] = ++cnt;
        p = trie[p][c];
    }
};
auto find = [&](int x) -> int {
    int sum = 0, p = 0;
    for (int i = 30; i >= 0; i--) {
        int c = (x >> i) & 1;
        if (trie[p][c ^ 1]) {
            p = trie[p][c ^ 1];
            sum += (1 << i);
        } else {
            p = trie[p][c];
        }
    }
    return sum;
};
```

字典树合并

来自浙大城市学院 2023 校赛 E 题.

给定一棵根为 1 的树, 每个点的点权为 w_i. 一共 q 次询问, 每次给出一对 u, v, 询问以 v 为根的子树上的点与 u 的权值最大异或值.

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

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

    // 离线询问 //
    std::cin >> m;
    std::vector<vpi> q(n + 1);
    vi ans(m + 1);
    for (int i = 1; i <= m; i++) {
        int u, v;
        std::cin >> u >> v;
        q[v].emplace_back(u, i);
    }

    // 01 trie //
    std::vector<std::array<int, 2>> tr(1);
    auto new_node = [&]() -> int {
        tr.emplace_back();
        return tr.size() - 1;
    };
    vi id(n + 1);
    auto insert = [&](int root, int x) {
        int p = root;
        for (int i = 29; i >= 0; i--) {
            int c = x >> i & 1;
            if (!tr[p][c]) tr[p][c] = new_node();
            p = tr[p][c];
        }
    };
    auto query = [&](int root, int x) -> int {
        int ans = 0, p = root;
        for (int i = 29; i >= 0; i--) {
            int c = x >> i & 1;
            if (tr[p][c ^ 1]) {
                p = tr[p][c ^ 1];
                ans += (1 << i);
            } else {
                p = tr[p][c];
            }
        }
        return ans;
    };
    std::function<int(int, int)> merge = [&](int a, int b) -> int {
        // b 的信息挪到 a 上 //
        if (!a) return b;
```

```
        if (!b) return a;
        tr[a][0] = merge(tr[a][0], tr[b][0]);
        tr[a][1] = merge(tr[a][1], tr[b][1]);
        return a;
    };
    std::function<void(int, int)> dfs = [&](int u, int fa) {
        id[u] = new_node();
        insert(id[u], w[u]);
        for (auto v : e[u]) {
            if (v == fa) continue;
            dfs(v, u);
            id[u] = merge(id[u], id[v]);
        }
        for (auto [v, i] : q[u]) {
            ans[i] = query(id[u], w[v]);
        }
    };
    dfs(1, 0);
    for (int i = 1; i <= m; i++) std::cout << ans[i] << endl;
    return 0;
}
```

5 math - number theory

5.1 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>;
```

5.2 Eculid

欧几里得算法

```
std::gcd(a, b)
```

扩展欧几里得算法

```
/* exgcd */
auto exgcd = [&](LL a, LL b, LL& x, LL& y) {
    LL x1 = 1, x2 = 0, x3 = 0, x4 = 1;
    while (b != 0) {
        LL c = a / b;
        std::tie(x1, x2, x3, x4, a, b) =
            std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
    }
    x = x1, y = x2;
};
auto exgcd = [&](auto&& self, LL a, LL b, LL& x, LL& y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    LL d = self(self, b, a % b, y, x);
    y -= a / b * x;
    return d;
};
```

```
auto exgcd = [&](auto&& self, LL a, LL b, LL& x, LL& y) {
    if (!b) {
        x = 1, y = 0;
        return;
    }
    self(self, b, a % b, y, x);
    y -= a / b * x;
};
```

```
auto exgcd = [&](auto&& self, LL a, LL b, LL& x, LL& y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    LL d = self(self, b, a % b, y, x);
    y -= a / b * x;
    return d;
};
```

类欧几里得算法

一般形式：求 $f(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$

$f(a, b, c, n) = nm - f(c, c-b-1, a, m-1)$

```
LL f(LL a, LL b, LL c, LL n) {
    if (a == 0) return ((b / c) * (n + 1));
    if (a >= c || b >= c)
        return f(a % c, b % c, c, n) + (a / c) * n * (n + 1) / 2 + (b / c) * (n + 1);
    LL m = (a * n + b) / c;
    LL v = f(c, c - b - 1, a, m - 1);
    return n * m - v;
}
```

更进一步，求：$g(a, b, c, n) = \sum_{i=0}^{n} i\lfloor\frac{ai+b}{c}\rfloor$ 以及 $h(a, b, c, n) = \sum_{i=0}^{n} \lfloor\frac{ai+b}{c}\rfloor^2$

$g(a, b, c, n) = \lfloor\frac{mn(n+1)-f(c,c-b-1,a,m-1)-h(c,c-b-1,a,m-1)}{2}\rfloor$

$h(a, b, c, n) = nm(m+1) - 2f(c, c-b-1, a, m-1) - 2g(c, c-b-1, a, m-1) - f(a, b, c, n)$

```
const int inv2 = 499122177;
const int inv6 = 166374059;

LL f(LL a, LL b, LL c, LL n);
LL g(LL a, LL b, LL c, LL n);
LL h(LL a, LL b, LL c, LL n);

struct data {
    LL f, g, h;
};

data calc(LL a, LL b, LL c, LL n) {
    LL ac = a / c, bc = b / c, m = (a * n + b) / c, n1 = n + 1, n21 = n * 2 + 1;
    data d;
    if (a == 0) {
        d.f = bc * n1 % mod;
        d.g = bc * n % mod * n1 % mod * inv2 % mod;
        d.h = bc * bc % mod * n1 % mod;
        return d;
    }
    if (a >= c || b >= c) {
        d.f = n * n1 % mod * inv2 % mod * ac % mod + bc * n1 % mod;
        d.g =
            ac * n % mod * n1 % mod * n21 % mod * inv6 % mod + bc * n % mod * n1 % mod * inv2 % mod;
        d.h = ac * ac % mod * n % mod * n1 % mod * n21 % mod * inv6 % mod +
              bc * bc % mod * n1 % mod + ac * bc % mod * n % mod * n1 % mod;
        d.f %= mod, d.g %= mod, d.h %= mod;
        data e = calc(a % c, b % c, c, n);
        d.h += e.h + 2 * bc % mod * e.f % mod + 2 * ac % mod * e.g % mod;
        d.g += e.g, d.f += e.f;
        d.f %= mod, d.g %= mod, d.h %= mod;
        return d;
    }
    data e = calc(c, c - b - 1, a, m - 1);
    d.f = n * m % mod - e.f, d.f = (d.f % mod + mod) % mod;
    d.g = m * n % mod * n1 % mod - e.h - e.f, d.g = (d.g * inv2 % mod + mod) % mod;
    d.h = n * m % mod * (m + 1) % mod - 2 * e.g - 2 * e.f - d.f;
    d.h = (d.h % mod + mod) % mod;
    return d;
}
```

5.3 inverse

线性递推

$a^{-1} \equiv -\lfloor\frac{p}{a}\rfloor \times (p\%a)^{-1}$

```
/* inverse */
vi inv(n + 1);
auto sieve_inv = [&](int n) {
    inv[1] = 1;
    for (int i = 2; i <= n; i++) {
        inv[i] = 1ll * (p - p / i) * inv[p % i] % p;
    }
};
```

求 n 个数的逆元

```
/* inverse */
auto inverse =[&](const vi& a) {
    int n = a.size();
    vi b(n), f(n), ivf(n);
    f[0] = a[0];
    for (int i = 1; i < n; i++) {
        f[i] = 1ll * f[i - 1] * a[i] % p;
    }
    ivf.back() = quick_power(f.back(), p - 2, p);
    for (int i = n - 1; i; i--) {
        ivf[i - 1] = 1ll * ivf[i] * a[i] % p;
    }
    b[0] = ivf[0];
    for (int i = 1; i < n; i++) {
        b[i] = 1ll * ivf[i] * f[i - 1] % p;
    }
    return b;
};
```

5.4 sieve

素数

```
vi prime, is_prime(n + 1, 1);
auto Euler_sieve = [&](int n){
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) prime.push_back(i);
        for (auto p : prime) {
            if (i * p > n) break;
            is_prime[i * p] = 0;
            if (i % p == 0) break;
        }
    }
};
```

欧拉函数

```
vi phi(n + 1), prime;
vi is_prime(n + 1, 1);
auto get_phi = [&](int n) {
    int cnt = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
            prime.push_back(i);
            phi[i] = i - 1;
        }
        for (auto p : prime) {
            if (i * p > n) break;
            is_prime[i * p] = 0;
            if (i % p) {
                phi[i * p] = phi[i] * phi[p];
            } else {
                phi[i * p] = phi[i] * p;
                break;
            }
        }
    }
};
```

约数和

```
vi g(n + 1), d(n + 1), prime;
vi is_prime(n + 1, 1);
auto get_d = [&](int n) {
    int tot = 0;
    g[1] = d[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
```

```
            prime.push_back(i);
            d[i] = g[i] = i + 1;
        }
        for (auto p : prime) {
            if (i * p > n) break;
            is_prime[i * p] = 0;
            if (i % p == 0) {
                g[i * p] = g[i] * p + 1;
                d[i * p] = d[i] / g[i] * g[i * p];
                break;
            } else {
                d[i * p] = d[i] * d[p];
                g[i * p] = 1 + p;
            }
        }
    }
};
```

莫比乌斯函数

```
vi mu(n + 1), prime;
vi is_prime(n + 1, 1);
auto get_mu = [&](int n) {
    mu[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
            prime.push_back(i);
            mu[i] = -1;
        }
        for (auto p : prime) {
            if (i * p > n) break;
            is_prime[i * p] = 0;
            if (i % p == 0) {
                mu[i * p] = 0;
                break;
            }
            mu[i * p] = -mu[i];
        }
    }
};
```

杜教筛

```
const int N = 1e7;
vi mu(N + 1), phi(N + 1), prime;
vl sum_phi(N + 1), sum_mu(N + 1);
vi is_prime(N + 1, 1);
std::map<LL, LL> mp_mu;

/* 计算 1 ~ 10^7 的 mu */
auto get_mu = [&](int n) {
    phi[1] = mu[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
            prime.push_back(i);
            phi[i] = i - 1;
            mu[i] = -1;
        }
        for (auto p : prime) {
            if (i * p > n) break;
            is_prime[i * p] = 0;
            if (i % p == 0) {
                phi[i * p] = phi[i] * p;
                mu[i * p] = 0;
                break;
            }
            phi[i * p] = phi[i] * phi[p];
            mu[i * p] = -mu[i];
        }
    }
};
get_mu(N);
for (int i = 1; i <= N; i++) {
    sum_phi[i] = sum_phi[i - 1] + phi[i];
    sum_mu[i] = sum_mu[i - 1] + mu[i];
}

/* 杜教筛: 求 mu 的前缀和 */
```

```cpp
std::function<LL(LL)> S_mu = [&](LL x) -> LL {
    if (x <= N) return sum_mu[x];
    auto it = mp_mu.find(x);
    if (it != mp_mu.end()) return mp_mu[x];
    LL ans = 1;
    for (LL i = 2, j; i <= x; i = j + 1) {
        j = x / (x / i);
        ans -= S_mu(x / i) * (j - i + 1);
    }
    return mp_mu[x] = ans;
};

/* 杜教筛: 求 phi 的前缀和 */
auto S_phi = [&](LL x) -> LL {
    if (x <= N) return sum_phi[x];
    LL ans = 0;
    for (LL i = 1, j; i <= x; i = j + 1) {
        j = x / (x / i);
        ans += 1ll * (S_mu(j) - S_mu(i - 1)) * (x / i) * (x / i);
    }
    return (ans - 1) / 2 + 1;
};
```

5.5 block

分块的逻辑

下取整 $\lfloor \frac{n}{g} \rfloor = k$ 的分块 $()g \leqslant n)$

```cpp
for(int l = 1, r, k; l <= n; l = r + 1){
    k = n / l;
    r = n / (n / l);
    debug(l, r, k);
}
```

$k = \lfloor \frac{n}{g} \rfloor$ 从大到小遍历 $\lfloor \frac{n}{g} \rfloor$ 的所有取值, $[l,\ r]$ 对应的是 g 取值的区间.

```
n = 11
[l, r, k] : 1 1 11
[l, r, k] : 2 2 5
[l, r, k] : 3 3 3
[l, r, k] : 4 5 2
[l, r, k] : 6 11 1
```

上取整 $\lceil \frac{n}{g} \rceil = k$ 的分块 $(g < n)$

```cpp
for(int l = 1, r, k; l < n; l = r + 1){
    k = (n + l - 1) / l;
    r = (n + k - 2) / (k - 1) - 1;
    debug(l, r, k);
}
```

$k = \lceil \frac{n}{g} \rceil$ 从大到小遍历 $\lceil \frac{n}{g} \rceil$ 的所有取值, $[l,\ r]$ 对应的是 g 取值的区间.

```
n = 11
[l, r, k] : 1 1 11
[l, r, k] : 2 2 6
[l, r, k] : 3 3 4
[l, r, k] : 4 5 3
[l, r, k] : 6 10 2
```

一般形式

计算 $\sum_{i=1}^{n} f(i) \lfloor \frac{n}{i} \rfloor$, 设 $s(i)$ 为 $f(i)$ 的前缀和。

```cpp
for (int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    ans += (s[r] - s[l - 1]) * (n / l);
}
```

$\sum_{i=1}^{n} f(i)\lfloor\frac{a}{i}\rfloor\lfloor\frac{b}{i}\rfloor$

```
for (int l = 1, r, r1, r2; l <= n; l = r + 1) {
    if (a / l) {
        r1 = a / (a / l);
    } else {
        r1 = n;
    }
    if (b / l) {
        r2 = b / (b / l);
    } else {
        r2 = n;
    }
    r = min(min(r1, r2), n);
    ans += (s[r] - s[l - 1]) * (a / l) * (b / l);
}
```

5.6 CRT & exCRT

求解

$$\begin{cases} N \equiv a_1 \bmod m_1 \\ N \equiv a_2 \bmod m_2 \\ \dots \\ N \equiv a_n \bmod m_n \end{cases}$$

有 $N \equiv \sum_{i=1}^{k} a_i \times \mathrm{inv}\left(\frac{M}{m_i}, m_i\right) \times \left(\frac{M}{m_i}\right) \bmod M$

```
/* CRT */
auto crt = [&](int n, const vi& a, const vi& m) -> LL{
    LL ans = 0, M = 1;
    for(int i = 1; i <= n; i++) M *= m[i];
    for(int i = 1; i <= n; i++){
        ans = (ans + a[i] * inv(M / m[i], m[i]) * (M / m[i])) % M;
    }
    return (ans % M + M) % M;
};
```

扩展中国剩余定理

```
/* exCRT */
auto excrt = [&](int n, const vi& a, const vi& m) -> LL{
    LL A = a[1], M = m[1];
    for (int i = 2; i <= n; i++) {
        LL x, y, d = std::gcd(M, m[i]);
        exgcd(M, m[i], x, y);
        LL mod = M / d * m[i];
        x = x * (a[i] - A) / d % (m[i] / d);
        A = ((M * x + A) % mod + mod) % mod;
        M = mod;
    }
    return A;
};
```

5.7 BSGS & exBSGS

求解满足 $a^x \equiv b \bmod p$ 的 x

```
/* BSGS */
/* return value = -1e18 means no solution */
auto BSGS = [&](LL a, LL b, LL p) {
    if (1 % p == b % p) return 0ll;
    LL k = std::sqrt(p) + 1;
    std::unordered_map<LL, LL> hash;
    for (LL i = 0, j = b % p; i < k; i++) {
        hash[j] = i;
        j = j * a % p;
    }
    LL ak = 1;
    for (int i = 1; i <= k; i++) ak = ak * a % p;
```

```
    for (int i = 1, j = ak; i <= k; i++) {
        if (hash.count(j)) return 1ll * i * k - hash[j];
        j = 1ll * j * ak % p;
    }
    return -INF;
};
```

$(a,p) \neq 1$ 的情形

```
/* exBSGS */
/* return value < 0 means no solution */
auto exBSGS = [&](auto&& self, LL a, LL b, LL p) {
    b = (b % p + p) % p;
    if (1ll % p == b % p) return 0ll;
    LL x, y, d = std::gcd(a, p);
    exgcd(exgcd, a, p, x, y);
    if (d > 1) {
        if (b % d != 0) return -INF;
        exgcd(exgcd, a / d, p / d, x, y);
        return self(self, a, b / d * x % (p / d), p / d) + 1;
    }
    return BSGS(a, b, p);
};
```

5.8 Miller Rabin

```
/* Miller Rabin */
vl vv = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
auto quick_power = [&](LL a, LL n, LL mod) {
    LL ans = 1;
    while (n) {
        if (n & 1) ans = (i128) ans * a % mod;
        a = (i128) a * a % mod;
        n >>= 1;
    }
    return ans;
};

auto millerRabin = [&](LL n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    int s = __builtin_ctzll(n - 1);
    LL d = n >> s;
    for (auto a : vv) {
        LL p = quick_power(a % n, d, n);
        int i = s;
        while (p != 1 and p != n - 1 and a % n and i--) p = (i128) p * p % n;
        if (p != n - 1 and i != s) return false;
    }
    return true;
};
```

5.9 Pollard Rho

能在 $O(n^{\frac{1}{4}})$ 的时间复杂度随机出一个 n 的非平凡因数.

```
/* pollard rho */
auto pollard_rho = [&](LL x) -> LL{
    LL s = 0, t = 0, val = 1;
    LL c = rand() % (x - 1) + 1;
    for(int goal = 1;; goal <<= 1, s = t, val = 1){
        for(int step = 1; step <= goal; step++){
            t = ((i128) t * t + c) % x;
            val = (i128) val * abs(t - s) % x;
            if(step % 127 == 0){
                LL d = std::gcd(val, x);
                if(d > 1) return d;
            }
        }
        LL d = std::gcd(val, x);
        if(d > 1) return d;
    }
};
```

利用 Miller Rabin 和 Pollard Rho 进行素因数分解

```cpp
auto factorize = [&](LL a) -> vl{
    vl ans, stk;
    for (auto p : prime) {
        if (p > 1000) break;
        while (a % p == 0) {
            ans.push_back(p);
            a /= p;
        }
        if (a == 1) return ans;
    }
    stk.push_back(a);
    while (!stk.empty()) {
        LL b = stk.back();
        stk.pop_back();
        if (miller_rabin(b)) {
            ans.push_back(b);
            continue;
        }
        LL c = b;
        while (c >= b) c = pollard_rho(b);
        stk.push_back(c);
        stk.push_back(b / c);
    }
    return ans;
};
```

5.10 quadratic residu

```cpp
/* cipolla */
auto cipolla = [&](int x) {
    std::srand(time(0));
    auto check = [&](int x) -> bool { return pow(x, (mod - 1) / 2) == 1; };
    if (!x) return 0;
    if (!check(x)) return -1;
    int a, b;
    while (1) {
        a = rand() % mod;
        b = sub(mul(a, a), x);
        if (!check(b)) break;
    }
    PII t = {a, 1};
    PII ans = {1, 0};
    auto mulp = [&](PII x, PII y) -> PII {
        auto [x1, x2] = x;
        auto [y1, y2] = y;
        int c = add(mul(x1, y1), mul(x2, y2, b));
        int d = add(mul(x1, y2), mul(x2, y1));
        return {c, d};
    };
    for (int i = (mod + 1) / 2; i; i >>= 1) {
        if (i & 1) ans = mulp(ans, t);
        t = mulp(t, t);
    }
    return std::min(ans.ff, mod - ans.ff);
}
```

5.11 Lucas

卢卡斯定理

用于求大组合数, 并且模数是一个不大的素数.

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

$\binom{n \bmod p}{m \bmod p}$ 可以直接计算, $\binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor}$ 可以继续使用卢卡斯计算.

递归至 $m = 0$ 的时候, 返回 1.

p 不太大, 一般在 10^5 左右.

```
auto C = [&](LL n, LL m, LL p) -> LL {
    if (n < m) return 0;
    if (m == 0) return 1;
    return fac[n] * inv_fac[m] % p * inv_fac[n - m] % p;
};
/* lucas */
auto lucas = [&](auto&& self, LL n, LL m, LL p) -> LL {
    if (n < m) return 0;
    if (m == 0) return 1;
    return C(n % p, m % p, p) * self(self, n / p, m / p, p) % p;
}
```

素数在组合数中的次数

Legengre 给出一种 $n!$ 中素数 p 的幂次的计算方式为:

$$\sum_{1\leqslant j}\left\lfloor\frac{n}{p^j}\right\rfloor.$$

另一种计算方式利用 p 进制下各位数字和:

$$v_p(n!)=\frac{n-S_p(n)}{p-1}.$$

则有

$$v_p(C_m^n)=\frac{S_p(n)+S_p(m-n)-S_p(m)}{p-1}.$$

扩展卢卡斯定理

计算

$$\begin{pmatrix} n \\ m \end{pmatrix} \bmod p,$$

p 可能为合数.

第一部分: CRT.

原问题变成求

$$\begin{cases} \begin{pmatrix} n \\ m \end{pmatrix} \equiv a_1 \bmod p_1^{\alpha_1} \\ \begin{pmatrix} n \\ m \end{pmatrix} \equiv a_2 \bmod p_2^{\alpha_2} \\ \cdots \\ \begin{pmatrix} n \\ m \end{pmatrix} \equiv a_k \bmod p_k^{\alpha_k} \end{cases}$$

在求出 a_i 之后就可以利用 CRT 求出答案.

第二部分: 移除分子分母中的素数

问题转换成求解

$$\begin{pmatrix} n \\ m \end{pmatrix} \bmod q^k.$$

等价于

$$\frac{\frac{n!}{q^x}}{\frac{m!}{q^y}\frac{(n-m)!}{q^z}}q^{x-y-z} \bmod q^k,$$

其中 x 表示 $n!$ 中 q 的次数, y, z 同理.

第三部分: 威尔逊定理的推论

问题转换为求

$$\frac{n!}{q^x} \bmod q^k.$$

可以利用威尔逊定理的推论.

```
/* exlucas */
auto exLucas = [&](LL n, LL m, LL p) {
    auto inv = [&](LL a, LL p) {
        LL x, y;
        exgcd(a, p, x, y);
        return (x % p + p) % p;
    };

    auto func = [&](auto&& self, LL n, LL pi, LL pk) {
        if (!n) return 1ll;
        LL ans = 1;
        for (LL i = 2; i <= pk; i++) {
            if (i % pi) ans = ans * i % p;
        }
        ans = quick_power(ans, n / pk, pk);
        for (LL i = 2; i <= n % pk; i++) {
            if (i % pi) ans = ans * i % pk;
        }
        ans = ans * self(self, n / pi, pi, pk) % pk;
        return ans;
    };

    auto multiLucas = [&](LL n, LL m, LL pi, LL pk) {
        LL cnt = 0;
        for (LL i = n; i; i /= pi) cnt += i / pi;
        for (LL i = m; i; i /= pi) cnt -= i / pi;
        for (LL i = n - m; i; i /= pi) cnt -= i / pi;
        LL ans = quick_power(pi, cnt, pk) * func(func, n, pi, pk) % pk;
        ans = ans * inv(func(func, m, pi, pk), pk) % pk;
        ans = ans * inv(func(func, n - m, pi, pk), pk) % pk;
        return ans;
    };

    auto crt = [&](const vl& a, const vl& m, int k) {
        LL ans = 0;

        for (int i = 0; i < k; i++) {
            ans = (ans + a[i] * inv(p / m[i], m[i]) * (p / m[i])) % p;
        }
        return (ans % p + p) % p;
    };

    vl a, prime;
    LL pp = p;
    for (int i = 2; i * i <= pp; i++) {
        if (pp % i) continue;
        prime.push_back(1);
        while (pp % i == 0) {
            prime.back() *= i;
            pp /= i;
        }
        a.push_back(multiLucas(n, m, i, prime.back()));
    }
    if (pp > 1) {
        prime.push_back(pp);
        a.push_back(multiLucas(n, m, pp, pp));
    }
    return crt(a, prime, a.size());
};
```

5.12 Wilson

简单结论

对于素数 p 有

$$(p-1)! \equiv -1 \bmod p.$$

推论

令 $(n!)_p$ 表示不大于 n 且不被 p 整除的正整数的乘积.

特殊情形: n 为素数 p 时即为上述结论.

一般结论: 对素数 p 和正整数 q 有

$$((p^q)!)_p \equiv \pm 1 \bmod p^q.$$

详细定义:

$$((p^q)!)_p = \begin{cases} 1 & \text{if } p = 2 \text{ and } q \geqslant 3, \\ -1 & \text{other wise.} \end{cases}$$

5.13 LTE

将素数 p 在整数 n 中的个数记为 $v_p(n)$.

$(n,p) = 1$

对所有素数 p 和满足 $(n,p) = 1$ 的整数 n, 有

1. 若 $p \mid x - y$, 则有
$$v_p(x^n - y^n) = v_p(x - y).$$
2. 若 $p \mid x - y$, 则对奇数 n 有
$$v_p(x^n + y^n) = v_p(x + y).$$

p 是奇素数

对所有奇素数 p 有

1. 若 $p \mid x - y$, 则有
$$v_p(x^n - y^n) = v_p(x - y) + v_p(n).$$
2. 若 $p \mid x - y$, 则对奇数 n 有
$$v_p(x^n + y^n) = v_p(x + y) + v_p(n).$$

$p = 2$

对 $p = 2$ 且 $p \mid x - y$ 有

1. 对奇数 n 有
$$v_2(x^n - y^n) = v_2(x - y).$$
2. 对偶数 n 有
$$v_2(x^n - y^n) = v_2(x - y) + v_2(x + y) + v_2(n) - 1.$$

除此之外, 对上述 x, y, n, 若 $4 \mid x - y$, 有

1. $v_2(x + y) = 1$.
2. $v_2(x^n - y^n) = v_2(x - y) + v_2(n)$.

5.14 Mobius inversion

莫比乌斯函数

$$\mu(n)=\begin{cases}1 & n=1,\\ 0 & n\text{ 含有平方因子，}\\ (-1)^k & k\text{ 为 }n\text{ 的本质不同素因子个数．}\end{cases}$$

性质

$$\sum_{d|n}\mu(d)=\begin{cases}1 & n=1\\ 0 & n\neq 1\end{cases}.$$

$$\varphi(n)=\sum_{d|n}d\cdot\mu(\frac{n}{d}).$$

反演结论

$$[gcd(i,j)=1]=\sum_{d|gcd(i,j)}\mu(d).$$

$O(n\log n)$ 求莫比乌斯函数

```
mu[1] = 1;
for (int i = 1; i <= n; i++){
    for (int j = i + i; j <= n; j += i){
        mu[j] -= mu[i];
    }
}
```

莫比乌斯变换

设 $f(n),F(n)$.

1. $F(n)=\sum_{d|n}f(d)$, 则 $f(n)=\sum_{d|n}\mu(d)F\left(\frac{n}{d}\right)$.
2. $F(n)=\sum_{n|d}f(d)$, 则 $f(n)=\sum_{n|d}\mu\left(\frac{d}{n}\right)F(d)$.

6 math - polynomial

6.1 FTT

FFT 与拆系数 FFT

```
const int sz = 1 << 23;
int rev[sz];
int rev_n;
void set_rev(int n) {
    if (n == rev_n) return;
    for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) * n) / 2;
    rev_n = n;
}
tempt void butterfly(T* a, int n) {
    set_rev(n);
    for (int i = 0; i < n; i++) {
        if (i < rev[i]) std::swap(a[i], a[rev[i]]);
    }
}

namespace Comp {

long double pi = 3.141592653589793238;

tempt struct complex {
    T x, y;
    complex(T x = 0, T y = 0) : x(x), y(y) {}
    complex operator+(const complex& b) const { return complex(x + b.x, y + b.y); }

    complex operator-(const complex& b) const { return complex(x - b.x, y - b.y); }

    complex operator*(const complex& b) const {
        return complex<T>(x * b.x - y * b.y, x * b.y + y * b.x);
    }
    complex operator~() const { return complex(x, -y); }
    static complex unit(long double rad) { return complex(std::cos(rad), std::sin(rad)); }
};

}   // namespace Comp

struct fft_t {
    typedef Comp::complex<double> complex;
    complex wn[sz];

    fft_t() {
        for (int i = 0; i < sz / 2; i++) {
            wn[sz / 2 + i] = complex::unit(2 * Comp::pi * i / sz);
        }
        for (int i = sz / 2 - 1; i; i--) wn[i] = wn[i * 2];
    }

    void operator()(complex* a, int n, int type) {
        if (type == -1) std::reverse(a + 1, a + n);
        butterfly(a, n);
        for (int i = 1; i < n; i *= 2) {
            const complex* w = wn + i;
            for (complex *b = a, t; b != a + n; b += i + 1) {
                t = b[i];
                b[i] = *b - t;
                *b = *b + t;
                for (int j = 1; j < i; j++) {
                    t = (++b)[i] * w[j];
                    b[i] = *b - t;
                    *b = *b + t;
                }
            }
        }
        if (type == 1) return;
        for (int i = 0; i < n * 2; i++) ((double*) a)[i] /= n;
    }
} FFT;

typedef decltype(FFT)::complex complex;

vi fft(const vi& f, const vi& g) {
    static complex ff[sz];
    int n = f.size(), m = g.size();
    vi h(n + m - 1);
    if (std::min(n, m) <= 50) {
        for (int i = 0; i < n; i++) {
```

```
            for (int j = 0; j < m; ++j) {
                h[i + j] += f[i] * g[j];
            }
        }
        return h;
    }
    int c = 1;
    while (c + 1 < n + m) c *= 2;
    std::memset(ff, 0, sizeof(decltype(*(ff))) * (c));
    for (int i = 0; i < n; i++) ff[i].x = f[i];
    for (int i = 0; i < m; i++) ff[i].y = g[i];
    FFT(ff, c, 1);
    for (int i = 0; i < c; i++) ff[i] = ff[i] * ff[i];
    FFT(ff, c, -1);
    for (int i = 0; i + 1 < n + m; i++) h[i] = std::llround(ff[i].y / 2);
    return h;
}

vi mtt(const vi& f, const vi& g) {
    static complex ff[3][sz], gg[2][sz];
    static int s[3] = {1, 31623, 31623 * 31623};
    int n = f.size(), m = g.size();
    vi h(n + m - 1);
    if (std::min(n, m) <= 50) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                Add(h[i + j], mul(f[i], g[j]));
            }
        }
        return h;
    }
    int c = 1;
    while (c + 1 < n + m) c *= 2;
    for (int i = 0; i < 2; ++i) {
        std::memset(ff[i], 0, sizeof(decltype(*(ff[i]))) * (c));
        std::memset(gg[i], 0, sizeof(decltype(*(ff[i]))) * (c));
        for (int j = 0; j < n; ++j) ff[i][j].x = f[j] / s[i] % s[1];
        for (int j = 0; j < m; ++j) gg[i][j].x = g[j] / s[i] % s[1];
        FFT(ff[i], c, 1);
        FFT(gg[i], c, 1);
    }
    for (int i = 0; i < c; ++i) {
        ff[2][i] = ff[1][i] * gg[1][i];
        ff[1][i] = ff[1][i] * gg[0][i];
        gg[1][i] = ff[0][i] * gg[1][i];
        ff[0][i] = ff[0][i] * gg[0][i];
    }
    for (int i = 0; i < 3; ++i) {
        FFT(ff[i], c, -1);
        for (int j = 0; j + 1 < n + m; ++j) {
            Add(h[j], mul(std::llround(ff[i][j].x) % mod, s[i]));
        }
    }
    FFT(gg[1], c, -1);
    for (int i = 0; i + 1 < n + m; ++i) {
        Add(h[i], mul(std::llround(gg[1][i].x) % mod, s[1]));
    }
    return h;
}
```

6.2 FWT

and

$$C_i = \sum_{i=j\&k} A_j B_k$$

分治过程

$$\mathrm{FWT[A]} = merge(\mathrm{FWT[A_0]} + \mathrm{FWT[A_1]}, \mathrm{FWT[A_1]}),$$

$$\mathrm{UFWT[A']} = merge(\mathrm{UFWT[A'_0]} - \mathrm{UFWT[A'_1]}, \mathrm{UFWT[A'_1]}).$$

```
/* mod 998244353 */
     int n = v.size();
    for (int mid = 1; mid < n; mid <<= 1) {
        for (int block = mid << 1, j = 0; j < n; j += block) {
            for (int i = j; i < j + mid; i++) {
```

```
                LL x = v[i], y = v[i + mid];
                if (type == 1) {
                    v[i] = add(x, y);
                } else {
                    v[i] = sub(x, y);
                }
            }
        }
    }
    return v;
};
```

or

$$C_i = \sum_{i=j|k} A_j B_k$$

分治过程

$$\mathrm{FWT[A]} = merge(\mathrm{FWT[A_0]}, \mathrm{FWT[A_0]} + \mathrm{FWT[A_1]}),$$

$$\mathrm{UFWT[A']} = merge(\mathrm{UFWT[A'_0]}, -\mathrm{UFWT[A'_0]} + \mathrm{UFWT[A'_1]}).$$

```
/* mod 998244353 */
auto FWT_or = [&](vi v, int type) -> vi {
    int n = v.size();
    for (int mid = 1; mid < n; mid <<= 1) {
        for (int block = mid << 1, j = 0; j < n; j += block) {
            for (int i = j; i < j + mid; i++) {
                LL x = v[i], y = v[i + mid];
                if (type == 1) {
                    v[i + mid] = add(x, y);
                } else {
                    v[i + mid] = sub(y, x);
                }
            }
        }
    }
    return v;
};
```

xor

$$C_i = \sum_{i=j\oplus k} A_j B_k$$

分治过程

$$\mathrm{FWT[A]} = merge(\mathrm{FWT[A_0]} + \mathrm{FWT[A_1]}, \mathrm{FWT[A_0]} - \mathrm{FWT[A_1]}),$$

$$\mathrm{UFWT[A']} = merge\left(\frac{\mathrm{UFWT[A'_0]} + \mathrm{UFWT[A'_1]}}{2}, \frac{\mathrm{UFWT[A'_0]} - \mathrm{UFWT[A'_1]}}{2}\right).$$

```
/* mod 998244353 */
auto FWT_xor = [&](vi v, int type) -> vi {
    int n = v.size();
    for (int mid = 1; mid < n; mid <<= 1) {
        for (int block = mid << 1, j = 0; j < n; j += block) {
            for (int i = j; i < j + mid; i++) {
                LL x = v[i], y = v[i + mid];
                v[i] = add(x, y);
                v[i + mid] = sub(x, y);
                if (type == -1) {
                    Mul(v[i], inv2);
                    Mul(v[i + mid], inv2);
                }
            }
        }
    }
    return v;
};
```

统一地，

```
a = FWT(a, 1),  b = FWT(b, 1);
for (int i = 0; i < (1 << n); i++) {
    c[i] = mul(a[i], b[i]);
}
c = FWT(c, -1);
```

6.3 class polynomial

```
class polynomial : public vi {
   public:
    polynomial() = default;
    polynomial(const vi& v) : vi(v) {}
    polynomial(vi&& v) : vi(std::move(v)) {}

    int degree() { return size() - 1; }

    void clearzero() {
        while (size() && !back()) pop_back();
    }
};


polynomial& operator+=(polynomial& a, const polynomial& b) {
    a.resize(std::max(a.size(), b.size()), 0);
    for (int i = 0; i < b.size(); i++) {
        Add(a[i], b[i]);
    }
    a.clearzero();
    return a;
}

polynomial operator+(const polynomial& a, const polynomial& b) {
    polynomial ans = a;
    return ans += b;
}

polynomial& operator-=(polynomial& a, const polynomial& b) {
    a.resize(std::max(a.size(), b.size()), 0);
    for (int i = 0; i < b.size(); i++) {
        Sub(a[i], b[i]);
    }
    a.clearzero();
    return a;
}

polynomial operator-(const polynomial& a, const polynomial& b) {
    polynomial ans = a;
    return ans -= b;
}

class ntt_t {
   public:
    static const int maxbit = 22;
    static const int sz = 1 << maxbit;
    static const int mod = 998244353;
    static const int g = 3;

    std::array<int, sz + 10> w;
    std::array<int, maxbit + 10> len_inv;

    ntt_t() {
        int wn = pow(g, (mod - 1) >> maxbit);
        w[0] = 1;
        for (int i = 1; i <= sz; i++) {
            w[i] = mul(w[i - 1], wn);
        }
        len_inv[maxbit] = pow(sz, mod - 2);
        for (int i = maxbit - 1; ~i; i--) {
            len_inv[i] = add(len_inv[i + 1], len_inv[i + 1]);
        }
    }

    void operator()(vi& v, int& n, int type) {
        int bit = 0;
        while ((1 << bit) < n) bit++;
        int tot = (1 << bit);
        v.resize(tot, 0);
        vi rev(tot);
        n = tot;
        for (int i = 0; i < tot; i++) {
```

```
            rev[i] = rev[i >> 1] >> 1;
            if (i & 1) {
                rev[i] |= tot >> 1;
            }
        }
        for (int i = 0; i < tot; i++) {
            if (i < rev[i]) {
                std::swap(v[i], v[rev[i]]);
            }
        }
        for (int midd = 0; (1 << midd) < tot; midd++) {
            int mid = 1 << midd;
            int len = mid << 1;
            for (int i = 0; i < tot; i += len) {
                for (int j = 0; j < mid; j++) {
                    int w0 = v[i + j];
                    int w1 = mul(
                        w[type == 1 ? (j << maxbit - midd - 1) : (len - j << maxbit - midd - 1)],
                        v[i + j + mid]);
                    v[i + j] = add(w0, w1);
                    v[i + j + mid] = sub(w0, w1);
                }
            }
        }
        if (type == -1) {
            for (int i = 0; i < tot; i++) {
                v[i] = mul(v[i], len_inv[bit]);
            }
        }
    }
} NTT;
```

乘法

```
polynomial& operator*=(polynomial& a, const polynomial& b) {
    if (!a.size() || !b.size()) {
        a.resize(0);
        return a;
    }
    polynomial tmp = b;
    int deg = a.size() + b.size() - 1;
    int temp = deg;

    // 项数较小直接硬算

    if ((LL) a.size() * (LL) b.size() <= (LL) deg * 50LL) {
        tmp.resize(0);
        tmp.resize(deg, 0);
        for (int i = 0; i < a.size(); i++) {
            for (int j = 0; j < b.size(); j++) {
                tmp[i + j] = add(tmp[i + j], mul(a[i], b[j]));
            }
        }
        a = tmp;
        return a;
    }

    // 项数较多跑 NTT

    NTT(a, deg, 1);
    NTT(tmp, deg, 1);
    for (int i = 0; i < deg; i++) {
        Mul(a[i], tmp[i]);
    }
    NTT(a, deg, -1);
    a.resize(temp);
    return a;
}

polynomial operator*(const polynomial& a, const polynomial& b) {
    polynomial ans = a;
    return ans *= b;
}
```

逆

```
polynomial inverse(const polynomial& a) {
    polynomial ans({pow(a[0], mod - 2)});
```

```
    polynomial temp;
    polynomial tempa;
    int deg = a.size();
    for (int i = 0; (1 << i) < deg; i++) {
        tempa.resize(0);
        tempa.resize(1 << i << 1, 0);
        for (int j = 0; j != tempa.size() and j != deg; j++) {
            tempa[j] = a[j];
        }
        temp = ans * (polynomial({2}) - tempa * ans);
        if (temp.size() > (1 << i << 1)) {
            temp.resize(1 << i << 1, 0);
        }
        temp.clearzero();
        std::swap(temp, ans);
    }
    ans.resize(deg);
    return ans;
}
```

对数

```
polynomial diffrential(const polynomial& a) {
    if (!a.size()) {
        return a;
    }
    polynomial ans(vi(a.size() - 1));
    for (int i = 1; i < a.size(); i++) {
        ans[i - 1] = mul(a[i], i);
    }
    return ans;
}

polynomial integral(const polynomial& a) {
    polynomial ans(vi(a.size() + 1));
    for (int i = 0; i < a.size(); i++) {
        ans[i + 1] = mul(a[i], pow(i + 1, mod - 2));
    }
    return ans;
}

polynomial ln(const polynomial& a) {
    int deg = a.size();
    polynomial da = diffrential(a);
    polynomial inva = inverse(a);
    polynomial ans = integral(da * inva);
    ans.resize(deg);
    return ans;
}
```

指数

```
polynomial exp(const polynomial& a) {
    polynomial ans({1});
    polynomial temp;
    polynomial tempa;
    polynomial tempaa;
    int deg = a.size();
    for (int i = 0; (1 << i) < deg; i++) {
        tempa.resize(0);
        tempa.resize(1 << i << 1, 0);
        for (int j = 0; j != tempa.size() and j != deg; j++) {
            tempa[j] = a[j];
        }
        tempaa = ans;
        tempaa.resize(1 << i << 1);
        temp = ans * (tempa + polynomial({1}) - ln(tempaa));
        if (temp.size() > (1 << i << 1)) {
            temp.resize(1 << i << 1, 0);
        }
        temp.clearzero();
        std::swap(temp, ans);
    }
    ans.resize(deg);
    return ans;
}
```

根号

```
polynomial sqrt(polynomial& a) {
    polynomial ans({cipolla(a[0])});
    if (ans[0] == -1) return ans;
    polynomial temp;
    polynomial tempa;
    polynomial tempaa;
    int deg = a.size();
    for (int i = 0; (1 << i) < deg; i++) {
        tempa.resize(0);
        tempa.resize(1 << i << 1, 0);
        for (int j = 0; j != tempa.size() and j != deg; j++) {
            tempa[j] = a[j];
        }
        tempaa = ans;
        tempaa.resize(1 << i << 1);
        temp = (tempa * inverse(tempaa) + ans) * inv2;
        if (temp.size() > (1 << i << 1)) {
            temp.resize(1 << i << 1, 0);
        }
        temp.clearzero();
        std::swap(temp, ans);
    }
    ans.resize(deg);
    return ans;
}

// 特判 //

int cnt = 0;
for (int i = 0; i < a.size(); i++) {
    if (a[i] == 0) {
        cnt++;
    } else {
        break;
    }
}
if (cnt) {
    if (cnt == n) {
        for (int i = 0; i < n; i++) {
            std::cout << "0 ";
        }
        std::cout << endl;
        return 0;
    }
    if (cnt & 1) {
        std::cout << "-1" << endl;
        return 0;
    }
    polynomial b(vi(a.size() - cnt));
    for (int i = cnt; i < a.size(); i++) {
        b[i - cnt] = a[i];
    }
    a = b;
}
a.resize(n - cnt / 2);
a = sqrt(a);
if (a[0] == -1) {
    std::cout << "-1" << endl;
    return 0;
}
reverse(all(a));
a.resize(n);
reverse(all(a));
```

6.4 wsy poly

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

using ul = std::uint32_t;
using li = std::int32_t;
using ll = std::int64_t;
using ull = std::uint64_t;
using llf = long double;
using lf = double;
using vul = std::vector<ul>;
using vvul = std::vector<vul>;
using pulb = std::pair<ul, bool>;
using vpulb = std::vector<pulb>;
using vvpulb = std::vector<vpulb>;
```

```
using vb = std::vector<bool>;

const ul base = 998244353;

std::mt19937 rnd;

ul plus(ul a, ul b) { return a + b < base ? a + b : a + b - base; }

ul minus(ul a, ul b) { return a < b ? a + base - b : a - b; }

ul mul(ul a, ul b) { return ull(a) * ull(b) % base; }

void exgcd(li a, li b, li& x, li& y) {
    if (b) {
        exgcd(b, a % b, y, x);
        y -= x * (a / b);
    } else {
        x = 1;
        y = 0;
    }
}

ul inverse(ul a) {
    li x, y;
    exgcd(a, base, x, y);
    return x < 0 ? x + li(base) : x;
}

ul pow(ul a, ul b) {
    ul ret = 1;
    ul temp = a;
    while (b) {
        if (b & 1) {
            ret = mul(ret, temp);
        }
        temp = mul(temp, temp);
        b >>= 1;
    }
    return ret;
}


ul sqrt(ul x) {
    ul a;
    ul w2;
    while (true) {
        a = rnd() % base;
        w2 = minus(mul(a, a), x);
        if (pow(w2, base - 1 >> 1) == base - 1) {
            break;
        }
    }
    ul b = base + 1 >> 1;
    ul rs = 1, rt = 0;
    ul as = a, at = 1;
    ul qs, qt;
    while (b) {
        if (b & 1) {
            qs = plus(mul(rs, as), mul(mul(rt, at), w2));
            qt = plus(mul(rs, at), mul(rt, as));
            rs = qs;
            rt = qt;
        }
        b >>= 1;
        qs = plus(mul(as, as), mul(mul(at, at), w2));
        qt = plus(mul(as, at), mul(as, at));
        as = qs;
        at = qt;
    }
    return rs + rs < base ? rs : base - rs;
}

ul log(ul x, ul y, bool inited = false) {
    static std::map<ul, ul> bs;
    const ul d = std::round(std::sqrt(lf(base - 1)));
    if (!inited) {
        bs.clear();
        for (ul i = 0, j = 1; i != d; ++i, j = mul(j, x)) {
            bs[j] = i;
        }
    }
    ul temp = inverse(pow(x, d));
    for (ul i = 0, j = 1;; i += d, j = mul(j, temp)) {
        auto it = bs.find(mul(y, j));
        if (it != bs.end()) {
            return it->second + i;
        }
```

```
    }
}

ul powroot(ul x, ul y, bool inited = false) {
    const ul g = 3;
    ul lgx = log(g, x, inited);
    li s, t;
    exgcd(y, base - 1, s, t);
    if (s < 0) {
        s += base - 1;
    }
    return pow(g, ull(s) * ull(lgx) % (base - 1));
}

class polynomial : public vul {
    public:
    void clearzero() {
        while (size() && !back()) {
            pop_back();
        }
    }
    polynomial() = default;
    polynomial(const vul& a) : vul(a) {}
    polynomial(vul&& a) : vul(std::move(a)) {}
    ul degree() const { return size() - 1; }
    ul operator()(ul x) const {
        ul ret = 0;
        for (ul i = size() - 1; ~i; --i) {
            ret = mul(ret, x);
            ret = plus(ret, vul::operator[](i));
        }
        return ret;
    }
};

polynomial& operator+=(polynomial& a, const polynomial& b) {
    a.resize(std::max(a.size(), b.size()), 0);
    for (ul i = 0; i != b.size(); ++i) {
        a[i] = plus(a[i], b[i]);
    }
    a.clearzero();
    return a;
}

polynomial operator+(const polynomial& a, const polynomial& b) {
    polynomial ret = a;
    return ret += b;
}

polynomial& operator-=(polynomial& a, const polynomial& b) {
    a.resize(std::max(a.size(), b.size()), 0);
    for (ul i = 0; i != b.size(); ++i) {
        a[i] = minus(a[i], b[i]);
    }
    a.clearzero();
    return a;
}

polynomial operator-(const polynomial& a, const polynomial& b) {
    polynomial ret = a;
    return ret -= b;
}

class ntt_t {
    public:
    static const ul lgsz = 20;
    static const ul sz = 1 << lgsz;
    static const ul g = 3;
    ul w[sz + 1];
    ul leninv[lgsz + 1];
    ntt_t() {
        ul wn = pow(g, (base - 1) >> lgsz);
        w[0] = 1;
        for (ul i = 1; i <= sz; ++i) {
            w[i] = mul(w[i - 1], wn);
        }
        leninv[lgsz] = inverse(sz);
        for (ul i = lgsz - 1; ~i; --i) {
            leninv[i] = plus(leninv[i + 1], leninv[i + 1]);
        }
    }
    void operator()(vul& v, ul& n, bool inv) {
        ul lgn = 0;
        while ((1 << lgn) < n) {
            ++lgn;
        }
        n = 1 << lgn;
```

```
        v.resize(n, 0);
        for (ul i = 0, j = 0; i != n; ++i) {
            if (i < j) {
                std::swap(v[i], v[j]);
            }
            ul k = n >> 1;
            while (k & j) {
                j &= ~k;
                k >>= 1;
            }
            j |= k;
        }
        for (ul lgmid = 0; (1 << lgmid) != n; ++lgmid) {
            ul mid = 1 << lgmid;
            ul len = mid << 1;
            for (ul i = 0; i != n; i += len) {
                for (ul j = 0; j != mid; ++j) {
                    ul t0 = v[i + j];
                    ul t1 =
                        mul(w[inv ? (len - j << lgsz - lgmid - 1) : (j << lgsz - lgmid - 1)],
                            v[i + j + mid]);
                    v[i + j] = plus(t0, t1);
                    v[i + j + mid] = minus(t0, t1);
                }
            }
        }
        if (inv) {
            for (ul i = 0; i != n; ++i) {
                v[i] = mul(v[i], leninv[lgn]);
            }
        }
    }
} ntt;

polynomial& operator*=(polynomial& a, const polynomial& b) {
    if (!b.size() || !a.size()) {
        a.resize(0);
        return a;
    }
    polynomial temp = b;
    ul npmp1 = a.size() + b.size() - 1;
    if (ull(a.size()) * ull(b.size()) <= ull(npmp1) * ull(50)) {
        temp.resize(0);
        temp.resize(npmp1, 0);
        for (ul i = 0; i != a.size(); ++i) {
            for (ul j = 0; j != b.size(); ++j) {
                temp[i + j] = plus(temp[i + j], mul(a[i], b[j]));
            }
        }
        a = temp;
        a.clearzero();
        return a;
    }
    ntt(a, npmp1, false);
    ntt(temp, npmp1, false);
    for (ul i = 0; i != npmp1; ++i) {
        a[i] = mul(a[i], temp[i]);
    }
    ntt(a, npmp1, true);
    a.clearzero();
    return a;
}

polynomial operator*(const polynomial& a, const polynomial& b) {
    polynomial ret = a;
    return ret *= b;
}

polynomial& operator*=(polynomial& a, ul b) {
    if (!b) {
        a.resize(0);
        return a;
    }
    for (ul i = 0; i != a.size(); ++i) {
        a[i] = mul(a[i], b);
    }
    return a;
}

polynomial operator*(const polynomial& a, ul b) {
    polynomial ret = a;
    return ret *= b;
}

polynomial inverse(const polynomial& a, ul lgdeg) {
    polynomial ret({inverse(a[0])});
    polynomial temp;
```

```
    polynomial tempa;
    for (ul i = 0; i != lgdeg; ++i) {
        tempa.resize(0);
        tempa.resize(1 << i << 1, 0);
        for (ul j = 0; j != tempa.size() && j != a.size(); ++j) {
            tempa[j] = a[j];
        }
        temp = ret * (polynomial({2}) - tempa * ret);
        if (temp.size() > (1 << i << 1)) {
            temp.resize(1 << i << 1, 0);
        }
        temp.clearzero();
        std::swap(temp, ret);
    }
    return ret;
}

void quotientremain(const polynomial& a, polynomial b, polynomial& q, polynomial& r) {
    if (a.size() < b.size()) {
        q = polynomial();
        r = std::move(a);
        return;
    }
    std::reverse(b.begin(), b.end());
    auto ta = a;
    std::reverse(ta.begin(), ta.end());
    ul n = a.size() - 1;
    ul m = b.size() - 1;
    ta.resize(n - m + 1);
    ul lgnmmp1 = 0;
    while ((1 << lgnmmp1) < n - m + 1) {
        ++lgnmmp1;
    }
    q = ta * inverse(b, lgnmmp1);
    q.resize(n - m + 1);
    std::reverse(b.begin(), b.end());
    std::reverse(q.begin(), q.end());
    r = a - b * q;
}

polynomial mod(const polynomial& a, const polynomial& b) {
    polynomial q, r;
    quotientremain(a, b, q, r);
    return r;
}

polynomial quotient(const polynomial& a, const polynomial& b) {
    polynomial q, r;
    quotientremain(a, b, q, r);
    return q;
}

polynomial sqrt(const polynomial& a, ul lgdeg) {
    polynomial ret({sqrt(a[0])});
    polynomial temp;
    polynomial tempa;
    for (ul i = 0; i != lgdeg; ++i) {
        tempa.resize(0);
        tempa.resize(1 << i << 1, 0);
        for (ul j = 0; j != tempa.size() && j != a.size(); ++j) {
            tempa[j] = a[j];
        }
        temp = (tempa * inverse(ret, i + 1) + ret) * (base + 1 >> 1);
        if (temp.size() > (1 << i << 1)) {
            temp.resize(1 << i << 1, 0);
        }
        temp.clearzero();
        std::swap(temp, ret);
    }
    return ret;
}

polynomial diffrential(const polynomial& a) {
    if (!a.size()) {
        return a;
    }
    polynomial ret(vul(a.size() - 1, 0));
    for (ul i = 1; i != a.size(); ++i) {
        ret[i - 1] = mul(a[i], i);
    }
    return ret;
}

polynomial integral(const polynomial& a) {
    polynomial ret(vul(a.size() + 1, 0));
    for (ul i = 0; i != a.size(); ++i) {
        ret[i + 1] = mul(a[i], inverse(i + 1));
```

```
    }
    return ret;
}

polynomial ln(const polynomial& a, ul lgdeg) {
    polynomial da = diffrential(a);
    polynomial inva = inverse(a, lgdeg);
    polynomial ret = integral(da * inva);
    if (ret.size() > (1 << lgdeg)) {
        ret.resize(1 << lgdeg);
        ret.clearzero();
    }
    return ret;
}

polynomial exp(const polynomial& a, ul lgdeg) {
    polynomial ret({1});
    polynomial temp;
    polynomial tempa;
    for (ul i = 0; i != lgdeg; ++i) {
        tempa.resize(0);
        tempa.resize(1 << i << 1, 0);
        for (ul j = 0; j != tempa.size() && j != a.size(); ++j) {
            tempa[j] = a[j];
        }
        temp = ret * (polynomial({1}) - ln(ret, i + 1) + tempa);
        if (temp.size() > (1 << i << 1)) {
            temp.resize(1 << i << 1, 0);
        }
        temp.clearzero();
        std::swap(temp, ret);
    }
    return ret;
}

polynomial pow(const polynomial& a, ul k, ul lgdeg) { return exp(ln(a, lgdeg) * k, lgdeg); }

polynomial alpi[1 << 16][17];

polynomial getalpi(const ul x[], ul l, ul lgrml) {
    if (lgrml == 0) {
        return alpi[l][lgrml] = vul({minus(0, x[l]), 1});
    }
    return alpi[l][lgrml] = getalpi(x, l, lgrml - 1) * getalpi(x, l + (1 << lgrml - 1), lgrml - 1);
}

void multians(const polynomial& f, const ul x[], ul y[], ul l, ul lgrml) {
    if (f.size() <= 700) {
        for (ul i = l; i != l + (1 << lgrml); ++i) {
            y[i] = f(x[i]);
        }
        return;
    }
    if (lgrml == 0) {
        y[l] = f(x[l]);
        return;
    }
    multians(mod(f, alpi[l][lgrml - 1]), x, y, l, lgrml - 1);
    multians(mod(f, alpi[l + (1 << lgrml - 1)][lgrml - 1]), x, y, l + (1 << lgrml - 1), lgrml - 1);
}

ul sqrt(ul x) {
    ul a;
    ul w2;
    while (true) {
        a = rnd() % base;
        w2 = minus(mul(a, a), x);
        if (pow(w2, base - 1 >> 1) == base - 1) {
            break;
        }
    }
    ul b = base + 1 >> 1;
    ul rs = 1, rt = 0;
    ul as = a, at = 1;
    ul qs, qt;
    while (b) {
        if (b & 1) {
            qs = plus(mul(rs, as), mul(mul(rt, at), w2));
            qt = plus(mul(rs, at), mul(rt, as));
            rs = qs;
            rt = qt;
        }
        b >>= 1;
        qs = plus(mul(as, as), mul(mul(at, at), w2));
        qt = plus(mul(as, at), mul(as, at));
        as = qs;
        at = qt;
```

```
    }
    return rs + rs < base ? rs : base - rs;
}

ul log(ul x, ul y, bool inited = false) {
    static std::map<ul, ul> bs;
    const ul d = std::round(std::sqrt(lf(base - 1)));
    if (!inited) {
        bs.clear();
        for (ul i = 0, j = 1; i != d; ++i, j = mul(j, x)) {
            bs[j] = i;
        }
    }
    ul temp = inverse(pow(x, d));
    for (ul i = 0, j = 1;; i += d, j = mul(j, temp)) {
        auto it = bs.find(mul(y, j));
        if (it != bs.end()) {
            return it->second + i;
        }
    }
}

ul powroot(ul x, ul y, bool inited = false) {
    const ul g = 3;
    ul lgx = log(g, x, inited);
    li s, t;
    exgcd(y, base - 1, s, t);
    if (s < 0) {
        s += base - 1;
    }
    return pow(g, ull(s) * ull(lgx) % (base - 1));
}

ul n;

int main() {
    std::scanf("%u", &n);
    polynomial f;
    for (ul i = 0; i <= n; ++i) {
        ul t;
        std::scanf("%u", &t);
        f.push_back(t % base);
    }
    polynomial g = exp(ln(f * inverse(f[0]), 17) * inverse(3), 17) * powroot(f[0], 3);
    while (g.size() <= n) {
        g.push_back(0);
    }
    for (ul i = 0; i <= n; ++i) {
        if (i) {
            std::putchar(' ');
        }
        std::printf("%u", g[i]);
    }
    std::putchar('\n');
    return 0;
}
```

Lagrange interpolation

一般的插值

给出一个多项式 $f(x)$ 上的 n 个点 (x_i, y_i), 求 $f(k)$.

插值的结果是

$$f(x) = \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j},$$

直接带入 k 并且取模即可, 时间复杂度 $O(n^2)$.

```
auto lagrange = (const vi& x, const vi& y, int n, int k) {
    for (int i = 1; i <= n; i++) {
        LL s1 = y[i] % mod, s2 = 1ll;
        for (int j = 1; j <= n; j++) {
            if (i != j) {
                s1 = s1 * (k - x[j]) % mod;
                s2 = s2 * (x[i] - x[j]) % mod;
            }
        }
```

```
        Add(ans, mul(s1, quick_power(s2, mod - 2, mod)));
    }
    return ans;
};
```

坐标连续的插值

给出的点是 (i, y_i).

$$\begin{aligned}
f(x) &= \sum_{i=1}^{n} y_i \prod_{j\neq i} \frac{x-x_j}{x_i-x_j} \\
&= \sum_{i=1}^{n} y_i \prod_{j\neq i} \frac{x-j}{i-j} \\
&= \sum_{i=1}^{n} y_i \cdot \frac{\prod\limits_{j=1}^{n}(x-j)}{(x-i)(-1)^{n+1-i}(i-1)!(n+1-i)!} \\
&= \left(\prod_{j=1}^{n}(x-j)\right)\left(\sum_{i=1}^{n} \frac{(-1)^{n+1-i}y_i}{(x-i)(i-1)!(n+1-i)!}\right),
\end{aligned}$$

时间复杂度为 $O(n)$.

7 math - game theory

7.1 nim game

若 nim 和为 0, 则先手必败.

暴力打表.

```
vi SG(21, -1); /* 记忆化 */
std::function<int(int, int)> sg = [&](int x) -> int {
    if (/* 为最终态 */) return SG[x] = 0;
    if (SG[x] != -1) return SG[x];
    vi st;
    for (/* 枚举所有可到达的状态 y */) {
        st.push_back(sg(y));
    }
    std::sort(all(st));
    for (int i = 0; i < st.size(); i++) {
        if (st[i] != i) return SG[x] = i;
    }
    return SG[x] = st.size();
};
```

7.2 anti - nim game

若

1. 所有堆的石子均为一个, 且 nim 和不为 0,
2. 至少有一堆石子超过一个, 且 nim 和为 0,

则先手必败.

8 math - linear algebra

8.1 matrix

determinant mod 998244353

```
/* determinant */
auto det = [&](int n, vvi e) -> int {
    int ans = 1;
    for (int i = 1; i <= n; i++) {
        if (a[i][i] == 0) {
            for (int j = i + 1; j <= n; j++) {
                if (a[j][i] != 0) {
                    for (int k = i; k <= n; k++) {
                        std::swap(a[i][k], a[j][k]);
                    }
                    ans = sub(mod, ans);
                    break;
                }
            }
        }
        if (a[i][i] == 0) return 0;
        Mul(ans, a[i][i]);
        int x = pow(a[i][i], mod - 2);
        for (int k = i; k <= n; k++) {
            Mul(a[i][k], x);
        }
        for (int j = i + 1; j <= n; j++) {
            int x = a[j][i];
            for (int k = i; k <= n; k++) {
                Sub(a[j][k], mul(a[i][k], x));
            }
        }
    }
    return ans;
};
```

matrix multiplication

$A_{n\times m}$ 与 $B_{m\times k}$ 相乘并模 998244353.

```
/* matrix multiplication */
auto matmul = [&](int n, int m, int k, const vvi& a, const vvi& b) -> vvi {
    vvi c(n + 1, vi(k + 1));
    for (int i = 1; i <= n; i++) {
        for (int l = 1; l <= m; l++) {
            int x = a[i][l];
            for (int j = 1; j <= k; j++) {
                Add(c[i][j], mul(x, b[l][j]));
            }
        }
    }
    return c;
};
```

8.2 linear basis

```
/* linear basis */
vl p(63), s(63);  /* basis and case */
auto insert = [&](LL x, int id) {
    LL ans = 0;
    for (int i = 62; i >= 0; i--) {
        if (~(x >> i) & 1) continue;
        if (!p[i]) {
            p[i] = x;
            s[i] = ans ^ (1ll << id);
            break;
        }
        x ^= p[i], ans ^= s[i];
    }
    return x;
};
```

```cpp
auto query = [&](LL x) {
    LL ans = 0;
    for (int i = 62; i >= 0; i--) {
        if (~(x >> i) & 1) continue;
        x ^= p[i], ans ^= s[i];
    }
    return (x ? -1 : ans);
};
auto queryMax = [&]() {
    LL ans = 0;
    for (int i = 62; i >= 0; i--)
        if ((ans ^ p[i]) > ans) ans ^= p[i];
    return ans;
};
```

8.3 linear programming

9 complex number

```
tandu struct Comp {
    T a, b;
    Comp(T _a = 0, T _b = 0) { a = _a, b = _b; }
    Comp operator+(const Comp& x) const { return Comp(a + x.a, b + x.b); }
    Comp operator-(const Comp& x) const { return Comp(a - x.a, b - x.b); }
    Comp operator*(const Comp& x) const { return Comp(a * x.a - b * x.b, a * x.b + b * x.a); }
    bool operator==(const Comp& x) const { return a == x.a and b == x.b; }
    T real() { return a; }
    T imag() { return b; }
    U norm() { return (U) a * a + (U) b * b; }
    Comp conj() { return Comp(a, -b); }
    Comp operator/(const Comp& x) const {
        Comp y = x;
        Comp c = Comp(a, b) * y.conj();
        T d = y.norm();
        return Comp(c.a / d, c.b / d);
    }
};
typedef Comp<LL, LL> complex;
complex gcd(complex a, complex b) {
    LL d = b.norm();
    if (d == 0) return a;
    std::vector<complex> v(4);
    complex c = a * b.conj();
    auto fdiv = [&](LL a, LL b) -> LL { return a / b - ((a ^ b) < 0 && (a % b)); };
    v[0] = complex(fdiv(c.real(), d), fdiv(c.imag(), d));
    v[1] = v[0] + complex(1, 0);
    v[2] = v[0] + complex(0, 1);
    v[3] = v[0] + complex(1, 1);
    for (auto& x : v) {
        x = a - x * b;
    }
    std::sort(all(v), [&](complex a, complex b) { return a.norm() < b.norm(); });
    return gcd(b, v[0]);
};
```

10 graph

10.1 topology sort

```
/* topology sort */
vi top;
auto topsort = [&]() -> bool {
    vi d(n + 1);
    std::queue<int> q;
    for (int i = 1; i <= n; i++) {
        d[i] = e[i].size();
        if (!d[i]) q.push(i);
    }
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        top.push_back(u);
        for (auto v : e[u]) {
            d[v]--;
            if (!d[v]) q.push(v);
        }
    }
    if (top.size() != n) return false;
    return true;
};
```

10.2 shortest path

Floyd

```
/* floyd */
auto floyd = [&]() -> vvi {
    vvi dist(n + 1, vi(n + 1, inf));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            Min(dist[i][j], e[i][j]);
        }
        dist[i][i] = 0;
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                Min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
    return dist;
};
```

Dijkstra

```
/* dijkstra */
auto dijkstra = [&](int s) -> vl {
    vl dist(n + 1, INF);
    vi vis(n + 1, 0);
    dist[s] = 0;
    std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> q;
    q.emplace(0LL, s);
    while (!q.empty()) {
        auto [dis, u] = q.top();
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (const auto& [v, w] : e[u]) {
            if (dist[v] > dis + w) {
                dist[v] = dis + w;
                q.emplace(dist[v], v);
            }
        }
    }
    return dist;
};
```

SPFA

```
/* SPFA */
int n, m, s;
vl dist(n + 1, INF);
std::vector<bool> vis(n + 1);
std::vector<PLI > e(n + 1);
void spfa(int s){
    for (int i = 1; i <= n; i++) dist[i] = INF;
    dist[s] = 0;
    std::queue<int> q;
    q.push(s);
    vis[s] = true;
    while(q.size()){
        auto u = q.front();
        q.pop();
        vis[u] = false;
        for(const auto& [v, w] : e[u]){
            if(dist[v] > dist[u] + w){
                dist[v] = dist[u] + w;
                if(!vis[v]){
                    q.push(v);
                    vis[v] = true;
                }
            }
        }
    }
}
```

Johnson

```
/* johnson */
auto johnson = [&]() -> vvl {
    /* 负环 */
    vl dist1(n + 1);
    vi vis(n + 1), cnt(n + 1);
    auto spfa = [&]() -> bool {
        std::queue<int> q;
        for (int u = 1; u <= n; u++) {
            q.push(u);
            vis[u] = false;
        }
        while (!q.empty()) {
            auto u = q.front();
            q.pop();
            vis[u] = false;
            for (auto [v, w] : e[u]) {
                if (dist1[v] > dist1[u] + w) {
                    dist1[v] = dist1[u] + w;
                    Max(cnt[v], cnt[u] + 1);
                    if (cnt[v] >= n) return true;
                    if (!vis[v]) {
                        q.push(v);
                        vis[v] = true;
                    }
                }
            }
        }
        return false;
    };
    /* dijkstra */
    vl dist2(n + 1);
    auto dijkstra = [&](int s) {
        for (int u = 1; u <= n; u++) {
            dist2[u] = 1e9;
            vis[u] = false;
        }
        dist2[s] = 0;
        std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> q;
        q.emplace(0, s);
        while (!q.empty()) {
            auto [d, u] = q.top();
            q.pop();
            if (vis[u]) continue;
            vis[u] = true;
            for (const auto& [v, w] : e[u]) {
                if (dist2[v] > d + w) {
                    dist2[v] = d + w;
                    q.emplace(dist2[v], v);
                }
            }
```

```cpp
        }
    };
    if (spfa()) return vvl{};
    for (int u = 1; u <= n; u++) {
        for (auto& [v, w] : e[u]) {
            w += dist1[u] - dist1[v];
        }
    }
    vvl dist(n + 1, vl(n + 1));
    for (int u; u <= n; u++) {
        dijkstra(u);
        for (int v = 1; v <= n; v++) {
            if (dist2[v] == 1e9) {
                dist[u][v] = INF;
            } else {
                dist[u][v] = dist2[v] + dist1[v] - dist1[u];
            }
        }
    }
    return dist;
};
```

最短路计数 - Dijkstra

```cpp
/* dijkstra */
auto dijkstra = [&](int s) -> std::pair<vl, vi> {
    vl dist(n + 1, INF);
    vi cnt(n + 1), vis(n + 1);
    dist[s] = 0;
    cnt[s] = 1;
    std::priority_queue<PLI, std::vector<PLI>, std::greater<PLI>> q;
    q.emplace(0LL, s);
    while (!q.empty()) {
        auto [dis, u] = q.top();
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (const auto& [v, w] : e[u]) {
            if (dist[v] > dis + w) {
                dist[v] = dis + w;
                cnt[v] = cnt[u];
                q.push({dist[v], v});
            } else if (dist[v] == dis + w) {
                // cnt[v] += cnt[u];
                cnt[v] += cnt[u];
                cnt[v] %= mod;
            }
        }
    }
    return {dist, cnt};
};
```

最短路计数 - Floyd

```cpp
/* floyd */
auto floyd() = [&] -> std::pair<vvi, vvi> {
    vvi dist(n + 1, vi(n + 1, inf));
    vvi cnt(n + 1, vi(n + 1));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            Min(dist[i][j], e[i][j]);
        }
        dist[i][i] = 0;
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (dist[i][j] == dist[i][k] + dist[k][j]) {
                    cnt[i][j] += cnt[i][k] * cnt[k][j];
                } else if (dist[i][j] > dist[i][k] + dist[k][j]) {
                    cnt[i][j] = cnt[i][k] * cnt[k][j];
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    return {dist, cnt};
};
```

负环

判断的是最短路长度.

```
/* SPFA */
auto spfa = [&]() -> bool {
    std::queue<int> q;
    vi vis(n + 1), cnt(n + 1);
    for (int i = 1; i <= n; i++) {
        q.push(i);
        vis[i] = true;
    }
    while (!q.empty()) {
        auto u = q.front();
        q.pop();
        vis[u] = false;
        for (const auto& [v, w] : e[u]) {
            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                cnt[v] = cnt[u] + 1;
                if (cnt[v] >= n) return true;
                if (!vis[v]) {
                    q.push(v);
                    vis[v] = true;
                }
            }
        }
    }
    return false;
}
```

10.3 minimum spanning tree

Kruskal

```
/* kruskal */
std::vector<std::tuple<int, int, int>> edge;
auto kruskal = [&]() -> int {
    std::sort(all(edge), [&](std::tuple<int, int, int> a, std::tuple<int, int, int> b) {
        auto [x1, y1, w1] = a;
        auto [x2, y2, w2] = b;
        return w1 < w2;
    });
    int res = 0, cnt = 0;
    for (int i = 0; i < m; i++) {
        auto [a, b, w] = edge[i];
        a = find(a), b = find(b);
        if (a != b) {
            fa[a] = b;
            res += w;
            /* res = std::max(res, w); */
            cnt++;
        }
    }
    if (cnt < n - 1) return -1;
    return res;
}
```

10.4 SCC

Tarjan

```
/* tarjan */
vi dfn(n + 1), low(n + 1), stk(n + 1), belong(n + 1);
int timestamp = 0, top = 0, scc_cnt = 0;
std::vector<bool> in_stk(n + 1);
auto tarjan = [&](auto&& self, int u) -> void {
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u;
    in_stk[u] = true;
    for (const auto& v : e[u]) {
        if (!dfn[v]) {
            self(self, v);
```

```
            Min(low[u], low[v]);
        } else if (in_stk[v]) {
            Min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        scc_cnt++;
        int v;
        do {
            v = stk[top--];
            in_stk[v] = false;
            belong[v] = scc_cnt;
        } while (v != u);
    }
};
```

10.5 DCC

点双连通分量

求点双连通分量.

```
vi dfn(n + 1), low(n + 1), is_bcc(n + 1), stk;
int timestamp = 0, bcc_cnt = 0, root = 0;
vvi bcc(2 * n + 1);
std::function<void(int, int)> tarjan = [&](int u, int fa) {
    dfn[u] = low[u] = ++timestamp;
    int child = 0;
    stk.push_back(u);
    if (u == root and e[u].empty()) {
        bcc_cnt++;
        bcc[bcc_cnt].push_back(u);
        return;
    }
    for (auto v : e[u]) {
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = std::min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                child++;
                if (u != root or child > 1) {
                    is_bcc[u] = 1;
                }
                bcc_cnt++;
                int z;
                do {
                    z = stk.back();
                    stk.pop_back();
                    bcc[bcc_cnt].push_back(z);
                } while (z != v);
                bcc[bcc_cnt].push_back(u);
            }
        } else if (v != fa) {
            low[u] = std::min(low[u], dfn[v]);
        }
    }
};
for (int i = 1; i <= n; i++) {
    if (!dfn[i]) {
        root = i;
        tarjan(i, i);
    }
}
```

求割点.

```
vi dfn(n + 1), low(n + 1), is_bcc(n + 1);
int timestamp = 0, bcc = 0, root = 0;
std::function<void(int, int)> tarjan = [&](int u, int fa) {
    dfn[u] = low[u] = ++timestamp;
    int child = 0;
    for (auto v : e[u]) {
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = std::min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                child++;
                if ((u != root or child > 1) and !is_bcc[u]) {
                    bcc++;
```

```
                    is_bcc[u] = 1;
                }
            }
        } else if (v != fa) {
            low[u] = std::min(low[u], dfn[v]);
        }
    }
};
for (int i = 1; i <= n; i++) {
    if (!dfn[i]) {
        root = i;
        tarjan(i, i);
    }
}
```

边双连通分量

求边双连通分量.

```
std::vector<vpi> e(n + 1);
for (int i = 1; i <= m; i++) {
    int u, v;
    std::cin >> u >> v;
    e[u].emplace_back(v, i);
    e[v].emplace_back(u, i);
}
vi dfn(n + 1), low(n + 1), is_ecc(n + 1), fa(n + 1), stk;
int timestamp = 0, ecc_cnt = 0;
vvi ecc(2 * n + 1);
std::function<void(int, int)> tarjan = [&](int u, int id) {
    low[u] = dfn[u] = ++timestamp;
    stk.push_back(u);
    for (auto [v, idx] : e[u]) {
        if (!dfn[v]) {
            tarjan(v, idx);
            low[u] = std::min(low[u], low[v]);
        } else if (idx != id) {
            low[u] = std::min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        ecc_cnt++;
        int v;
        do {
            v = stk.back();
            stk.pop_back();
            ecc[ecc_cnt].push_back(v);
        } while (v != u);
    }
};
for (int i = 1; i <= n; i++) {
    if (!dfn[i]) {
        tarjan(i, 0);
    }
}
```

10.6 2-sat

给出 n 个集合, 每个集合有 2 个元素, 已知若干个数对 (a, b), 表示 a 与 b 矛盾. 要从每个集合各选择一个元素, 判断能否一共选 n 个两两不矛盾的元素.

```
/* two sat */
auto 2sat = [&](int n, const vpi& v) -> vi {
    /* tarjan */
    vvi e(2 * n);
    vi dfn(2 * n), low(2 * n), stk(2 * n), belong(2 * n);
    int timestamp = 0, top = 0, scc_cnt = 0;
    std::vector<bool> in_stk(2 * n);
    auto tarjan = [&](auto&& self, int u) -> void {
        dfn[u] = low[u] = ++timestamp;
        stk[++top] = u;
        in_stk[u] = true;
        for (const auto& v : e[u]) {
            if (!dfn[v]) {
                self(self, v);
                Min(low[u], low[v]);
```

```cpp
            } else if (in_stk[v]) {
                Min(low[u], dfn[v]);
            }
        }
        if (dfn[u] == low[u]) {
            scc_cnt++;
            int v;
            do {
                v = stk[top--];
                in_stk[v] = false;
                belong[v] = scc_cnt;
            } while (v != u);
        }
    };
    for (const auto& [a, b] : v) {
        e[a].push_back(b ^ 1);
        e[b].push_back(a ^ 1);
    }
    for (int i = 0; i < 2 * n; i++) {
        if (!dfn[i]) tarjan(tarjan, i);
    }
    vi ans;
    for (int i = 0; i < 2 * n; i += 2) {
        if (belong[i] == belong[i + 1]) return vi{};
        ans.push_back(belong[i] > belong[i + 1] ? i + 1 : i);
    }
    return ans;
};
```

上述将 i 与 $i+1$ 作为一个集合里的元素, 编号为 0 至 $2n-1$.

10.7 minimum ring

Floyd

```cpp
/* minimum ring */
auto min_circle = [&]() -> int {
    vvi dist(n + 1, vi(n + 1, inf));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            Min(dist[i][j], g[i][j]);
        }
        dist[i][i] = 0;
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i < k; i++) {
            for (int j = 1; j < i; j++) {
                Min(ans, dist[i][j] + g[i][k] + g[k][j]);
            }
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                Min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
    return ans;
};
```

tree - diameter

10.8 tree - center of gravity

```cpp
/* center of gravity */
int sum;    /* 点权和 */
vi size(n + 1), weight(n + 1), w(n + 1), depth(n + 1);
std::array<int, 2> centroid = {0, 0};
auto get_centroid = [&](auto&& self, int u, int fa) -> void {
    size[u] = w[u];
    weight[u] = 0;
    for (auto v : e[u]) {
        if (v == fa) continue;
        self(self, v, u);
        size[u] += size[v];
```

```
        Max(weight[u], size[v]);
    }
    Max(weight[u], sum - size[u]);
    if (weight[u] <= sum / 2) {
        centroid[centroid[0] != 0] = u;
    }
};
```

10.9 tree - DSU on tree

给出一课 n 个节点以 1 为根的树, 每个节点染上一种颜色, 询问以 u 为节点的子树中有多少种颜色.

```
// Problem: U41492 树上数颜色

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

    int n, m, dfn = 0, cnttot = 0;
    std::cin >> n;
    vvi e(n + 1);
    vi siz(n + 1), col(n + 1), son(n + 1), dfnl(n + 1), dfnr(n + 1), rank(n + 1);
    vi ans(n + 1), cnt(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);
    }
    for (int i = 1; i <= n; i++) {
        std::cin >> col[i];
    }
    auto add = [&](int u) -> void {
        if (cnt[col[u]] == 0) cnttot++;
        cnt[col[u]]++;
    };
    auto del = [&](int u) -> void {
        cnt[col[u]]--;
        if (cnt[col[u]] == 0) cnttot--;
    };
    auto dfs1 = [&](auto&& self, int u, int fa) -> void {
        dfnl[u] = ++dfn;
        rank[dfn] = u;
        siz[u] = 1;
        for (auto v : e[u]) {
            if (v == fa) continue;
            self(self, v, u);
            siz[u] += siz[v];
            if (!son[u] or siz[son[u]] < siz[v]) son[u] = v;
        }
        dfnr[u] = dfn;
    };
    auto dfs2 = [&](auto&& self, int u, int fa, bool op) -> void {
        for (auto v : e[u]) {
            if (v == fa or v == son[u]) continue;
            self(self, v, u, false);
        }
        if (son[u]) self(self, son[u], u, true);
        for (auto v : e[u]) {
            if (v == fa or v == son[u]) continue;
            rep(i, dfnl[v], dfnr[v]) { add(rank[i]); }
        }
        add(u);
        ans[u] = cnttot;
        if (op == false) {
            rep(i, dfnl[u], dfnr[u]) { del(rank[i]); }
        }
    };
    dfs1(dfs1, 1, 0);
    dfs2(dfs2, 1, 0, false);
    std::cin >> m;
    for (int i = 1; i <= m; i++) {
        int u;
        std::cin >> u;
        std::cout << ans[u] << endl;
    }
    return 0;
}
```

10.10 tree - AHU

```
/* AHU */
std::map<vi, int> mapple;
std::function<int(vvi&, int, int)> tree_hash = [&](vvi& e, int u, int fa) -> int {
    vi code;
    if (u == 0) code.push_back(-1);
    for (auto v : e[u]) {
        if (v == fa) continue;
        code.push_back(tree_hash(e, v, u));
    }
    std::sort(all(code));
    int id = mapple.size();
    auto it = mapple.find(code);
    if (it == mapple.end()) {
        mapple[code] = id;
    } else {
        id = it->ss;
    }
    return id;
};
```

10.11 tree - LCA

```
/* LCA */
int B = 30;
vvi e(n + 1), fa(n + 1, vi(B));
vi dep(n + 1);
auto dfs = [&](auto&& self, int u) -> void {
    for (auto v : e[u]) {
        if (v == fa[u][0]) continue;
        dep[v] = dep[u] + 1;
        fa[v][0] = u;
        self(self, v);
    }
};
auto init = [&]() -> void {
    dep[root] = 1;
    dfs(dfs, root);
    for (int j = 1; j < B; j++) {
        for (int i = 1; i <= n; i++) {
            fa[i][j] = fa[fa[i][j - 1]][j - 1];
        }
    }
};
init();
auto LCA = [&](int a, int b) -> int {
    if (dep[a] > dep[b]) std::swap(a, b);
    int d = dep[b] - dep[a];
    for (int i = 0; (1 << i) <= d; i++) {
        if (d & (1 << i)) b = fa[b][i];
    }
    if (a == b) return a;
    for (int i = B - 1; i >= 0 and a != b; i--) {
        if (fa[a][i] == fa[b][i]) continue;
        a = fa[a][i];
        b = fa[b][i];
    }
    return fa[a][0];
};
auto dist = [&](int a, int b) -> int { return dep[a] + dep[b] - dep[LCA(a, b)] * 2; };
```

10.12 tree - heavy light decomposion

对一棵有根树进行如下 4 种操作:

1. 1 x y z: 将节点 x 到节点 y 的最短路径上所有节点的值加上 z.
2. 2 x y: 查询节点 x 到节点 y 的最短路径上所有节点的值的和.
3. 3 x z: 将以节点 x 为根的子树上所有节点的值加上 z.
4. 4 x: 查询以节点 x 为根的子树上所有节点的值的和.

```
/* heavy light decomposion */
int cnt = 0;
vi son(n + 1), fa(n + 1), siz(n + 1), depth(n + 1);
vi dfn(n + 1), rank(n + 1), top(n + 1), botton(n + 1);
auto dfs1 = [&](auto&& self, int u) -> void {
    son[u] = -1, siz[u] = 1;
    for (auto v : e[u]) {
        if (depth[v] != 0) continue;
        depth[v] = depth[u] + 1;
        fa[v] = u;
        self(self, v);
        siz[u] += siz[v];
        if (son[u] == -1 or siz[v] > siz[son[u]]) son[u] = v;
    }
};
auto dfs2 = [&](auto&& self, int u, int t) -> void {
    top[u] = t;
    dfn[u] = ++cnt;
    rank[cnt] = u;
    botton[u] = dfn[u];
    if (son[u] == -1) return;
    self(self, son[u], t);
    Max(botton[u], botton[son[u]]);
    for (auto v : e[u]) {
        if (v != son[u] and v != fa[u]) {
            self(self, v, v);
            Max(botton[u], botton[v]);
        }
    }
};
depth[root] = 1;
dfs1(dfs1, root);
dfs2(dfs2, root, root);

/* 求 LCA */
auto LCA = [&](int a, int b) -> int {
    while (top[a] != top[b]) {
        if (depth[top[a]] < depth[top[b]]) std::swap(a, b);
        a = fa[top[a]];
    }
    return (depth[a] > depth[b] ? b : a);
};

/* 维护 u 到 v 的路径 */
while (top[u] != top[v]) {
    if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
    opt(dfn[top[u]], dfn[u]);
    u = fa[top[u]];
}
if (dfn[u] > dfn[v]) std::swap(u, v);
opt(dfn[u], dfn[v]);

/* 维护 u 为根的子树 */
opt(dfn[u], botton[u]);

/*
线段树的 build() 函数中
if(l == r) tree[u] = {l, l, w[rank[l]], 0};
*/

build(1, 1, n);
for (int i = 1; i <= m; i++) {
    int op, u, v;
    LL k;
    std::cin >> op;
    if (op == 1) {
        std::cin >> u >> v >> k;
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
            modify(1, dfn[top[u]], dfn[u], k);
            u = fa[top[u]];
        }
        if (dfn[u] > dfn[v]) std::swap(u, v);
        modify(1, dfn[u], dfn[v], k);
    } else if (op == 2) {
        std::cin >> u >> v;
        LL ans = 0;
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
            ans = (ans + query(1, dfn[top[u]], dfn[u])) % p;
            u = fa[top[u]];
        }
        if (dfn[u] > dfn[v]) std::swap(u, v);
        ans = (ans + query(1, dfn[u], dfn[v])) % p;
        std::cout << ans << endl;
    } else if (op == 3) {
```

```
        std::cin >> u >> k;
        modify(1, dfn[u], botton[u], k);
    } else {
        std::cin >> u;
        std::cout << query(1, dfn[u], botton[u]) % p << endl;
    }
}
```

10.13 tree - virtual tree

```
/* virtual tree */
auto build_vtree = [&](vi ver) -> void {
    std::sort(all(ver), [&](int x, int y) { return dfn[x] < dfn[y]; });
    vi stk = {1};
    for (auto v : ver) {
        int u = stk.back();
        int lca = LCA(v, u);
        if (lca != u) {
            while (dfn[lca] < dfn[stk.end()[-2]]) {
                g[stk.end()[-2]].push_back(stk.back());
                stk.pop_back();
            }
            u = stk.back();
            if (dfn[lca] != dfn[stk.end()[-2]]) {
                g[lca].push_back(u);
                stk.pop_back();
                stk.push_back(lca);
            } else {
                g[lca].push_back(u);
                stk.pop_back();
            }
        }
        stk.push_back(v);
    }
    while (stk.size() > 1) {
        int u = stk.end()[-2];
        int v = stk.back();
        g[u].push_back(v);
        stk.pop_back();
    }
};
```

10.14 tree - pseudo tree

```
/* ring detection (directed) */
vi vis(n + 1), fa(n + 1), ring;
auto dfs = [&](auto&& self, int u) -> bool {
    vis[u] = 1;
    for (const auto& v : e[u]) {
        if (!vis[v]) {
            fa[v] = u;
            if (self(self, v)) {
                return true;
            }
        } else if (vis[v] == 1) {
            ring.push_back(v);
            for (auto x = u; x != v; x = fa[x]) {
                ring.push_back(x);
            }
            reverse(all(ring));
            return true;
        }
    }
    vis[u] = 2;
    return false;
};
for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
        if (dfs(dfs, i)) {
            // operations //
        }
    }
}

/* cycle detection (undirected) */
vi vis(n + 1), ring;
vpi fa(n + 1);
```

```cpp
auto dfs = [&](auto&& self, int u, int from) -> bool {
    vis[u] = 1;
    for (const auto& [v, id] : e[u]) {
        if (id == from) continue;
        if (!vis[v]) {
            fa[v] = {u, id};
            if (self(self, v, id)) {
                return true;
            }
        } else if (vis[v] == 1) {
            ring.push_back(v);
            for (auto x = u; x != v; x = fa[x].ff) {
                ring.push_back(x);
            }
            return true;
        }
    }
    vis[u] = 2;
    return false;
};
for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
        if (dfs(dfs, i, 0)) {
            // operations //
        }
    }
}
```

10.15 tree - divide and conquer on tree

点分治

第一个题

一棵 $n \leqslant 10^4$ 个点的树，边权 $w \leqslant 10^4$. $m \leqslant 100$ 次询问树上是否存在长度为 $k \leqslant 10^7$ 的路径.

```cpp
// 洛谷 P3806 【模板】点分治1

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

    int n, m, k;
    std::cin >> n >> m;

    std::vector<vpi> e(n + 1);
    std::map<int, PII> mp;

    for (int i = 1; i < n; i++) {
        int u, v, w;
        std::cin >> u >> v >> w;
        e[u].emplace_back(v, w);
        e[v].emplace_back(u, w);
    }
    for (int i = 1; i <= m; i++) {
        std::cin >> k;
        mp[i] = {k, 0};
    }

    /* centroid decomposition */
    int top1 = 0, top2 = 0, root;
    vi len1(n + 1), len2(n + 1), vis(n + 1);
    static std::array<int, 20000010> cnt;

    std::function<int(int, int)> get_size = [&](int u, int fa) -> int {
        if (vis[u]) return 0;
        int ans = 1;
        for (auto [v, w] : e[u]) {
            if (v == fa) continue;
            ans += get_size(v, u);
        }
        return ans;
    };

    std::function<int(int, int, int, int&)> get_root = [&](int u, int fa, int tot,
                                                            int& root) -> int {
        if (vis[u]) return 0;
        int sum = 1, maxx = 0;
        for (auto [v, w] : e[u]) {
```

```
            if (v == fa) continue;
            int tmp = get_root(v, u, tot, root);
            Max(maxx, tmp);
            sum += tmp;
        }
        Max(maxx, tot - sum);
        if (2 * maxx <= tot) root = u;
        return sum;
    };

    std::function<void(int, int, int)> get_dist = [&](int u, int fa, int dist) -> void {
        if (dist <= 10000000) len1[++top1] = dist;
        for (auto [v, w] : e[u]) {
            if (v == fa or vis[v]) continue;
            get_dist(v, u, dist + w);
        }
    };

    auto solve = [&](int u, int dist) -> void {
        top2 = 0;
        for (auto [v, w] : e[u]) {
            if (vis[v]) continue;
            top1 = 0;
            get_dist(v, u, w);
            for (int i = 1; i <= top1; i++) {
                for (int tt = 1; tt <= m; tt++) {
                    int k = mp[tt].ff;
                    if (k >= len1[i]) mp[tt].ss |= cnt[k - len1[i]];
                }
            }
            for (int i = 1; i <= top1; i++) {
                len2[++top2] = len1[i];
                cnt[len1[i]] = 1;
            }
        }
        for (int i = 1; i <= top2; i++) cnt[len2[i]] = 0;
    };

    std::function<void(int)> divide = [&](int u) -> void {
        vis[u] = cnt[0] = 1;
        solve(u, 0);
        for (auto [v, w] : e[u]) {
            if (vis[v]) continue;
            get_root(v, u, get_size(v, u), root);
            divide(root);
        }
    };

    get_root(1, 0, get_size(1, 0), root);
    divide(root);

    for (int i = 1; i <= m; i++) {
        if (mp[i].ss == 0) {
            std::cout << "NAY" << endl;
        } else {
            std::cout << "AYE" << endl;
        }
    }

    return 0;
}
```

第二个题

一棵 $n \leqslant 4 \times 10^4$ 个点的树, 边权 $w \leqslant 10^3$. 询问树上长度不超过 $k \leqslant 2 \times 10^4$ 的路径的数量.

```
// 洛谷 P4178 Tree

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

    int n, k;
    std::cin >> n;
    std::vector<vpi> e(n + 1);
    for (int i = 1; i < n; i++) {
        int u, v, w;
        std::cin >> u >> v >> w;
        e[u].emplace_back(v, w);
        e[v].emplace_back(u, w);
    }
    std::cin >> k;

    /* centroid decomposition */
```

```
    int root;
    vi len, vis(n + 1);

    std::function<int(int, int)> get_size = [&](int u, int fa) -> int {
        if (vis[u]) return 0;
        int ans = 1;
        for (auto [v, w] : e[u]) {
            if (v == fa) continue;
            ans += get_size(v, u);
        }
        return ans;
    };

    std::function<int(int, int, int, int&)> get_root = [&](int u, int fa, int tot,
                                                            int& root) -> int {
        if (vis[u]) return 0;
        int sum = 1, maxx = 0;
        for (auto [v, w] : e[u]) {
            if (v == fa) continue;
            int tmp = get_root(v, u, tot, root);
            maxx = std::max(maxx, tmp);
            sum += tmp;
        }
        maxx = std::max(maxx, tot - sum);
        if (2 * maxx <= tot) root = u;
        return sum;
    };

    std::function<void(int, int, int)> get_dist = [&](int u, int fa, int dist) -> void {
        len.push_back(dist);
        for (auto [v, w] : e[u]) {
            if (v == fa || vis[v]) continue;
            get_dist(v, u, dist + w);
        }
    };

    auto solve = [&](int u, int dist) -> int {
        len.clear();
        get_dist(u, 0, dist);
        std::sort(all(len));
        int ans = 0;
        for (int l = 0, r = len.size() - 1; l < r;) {
            if (len[l] + len[r] <= k) {
                ans += r - l++;
            } else {
                r--;
            }
        }
        return ans;
    };

    std::function<int(int)> divide = [&](int u) -> int {
        vis[u] = true;
        int ans = solve(u, 0);
        for (auto [v, w] : e[u]) {
            if (vis[v]) continue;
            ans -= solve(v, w);
            get_root(v, u, get_size(v, u), root);
            ans += divide(root);
        }
        return ans;
    };

    get_root(1, 0, get_size(1, 0), root);
    std::cout << divide(root) << endl;

    return 0;
}
```

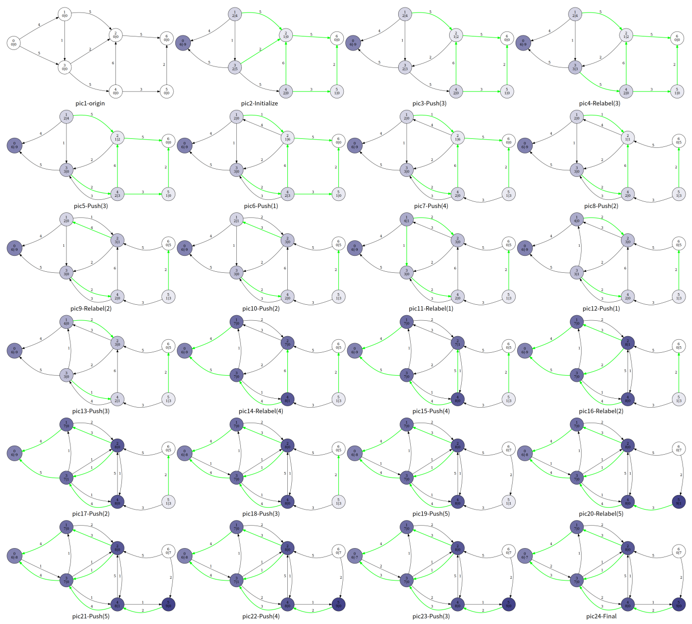
10.16 network flow - maximal flow

Dinic

```
/* dinic */
struct edge {
    int from, to;
    LL cap, flow;

    edge(int u, int v, LL c, LL f) : from(u), to(v), cap(c), flow(f) {}
};

```

```
struct Dinic {
    int n, m = 0, s, t;
    std::vector<edge> e;
    vi g[N];
    int d[N], cur[N], vis[N];

    void init(int n) {
        for (int i = 0; i < n; i++) g[i].clear();
        e.clear();
        m = 0;
    }

    void add(int from, int to, LL cap) {
        e.push_back(edge(from, to, cap, 0));
        e.push_back(edge(to, from, 0, 0));
        g[from].push_back(m++);
        g[to].push_back(m++);
    }

    bool bfs() {
        for (int i = 1; i <= n; i++) {
            vis[i] = 0;
        }
        std::queue<int> q;
        q.push(s), d[s] = 0, vis[s] = 1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = 0; i < g[u].size(); i++) {
                edge& ee = e[g[u][i]];
                if (!vis[ee.to] and ee.cap > ee.flow) {
                    vis[ee.to] = 1;
                    d[ee.to] = d[u] + 1;
                    q.push(ee.to);
                }
            }
        }
        return vis[t];
    }

    LL dfs(int u, LL now) {
        if (u == t || now == 0) return now;
        LL flow = 0, f;
        for (int& i = cur[u]; i < g[u].size(); i++) {
            edge& ee = e[g[u][i]];
            edge& er = e[g[u][i] ^ 1];
            if (d[u] + 1 == d[ee.to] and (f = dfs(ee.to, std::min(now, ee.cap - ee.flow))) > 0) {
                ee.flow += f, er.flow -= f;
                flow += f, now -= f;
                if (now == 0) break;
            }
        }
        return flow;
    }

    LL dinic() {
        LL ans = 0;
        while (bfs()) {
            for (int i = 1; i <= n; i++) cur[i] = 0;
            ans += dfs(s, INF);
        }
        return ans;
    }
} maxf;
```

HLPP




```cpp
/* hlpp */
struct HLPP {
    int n, m = 0, s, t;
    std::vector<edge> e;      /* 边 */
    std::vector<node> nd;     /* 点 */
    std::vector<int> g[N];    /* 点的连边编号 */
    std::priority_queue<node> q;
    std::queue<int> qq;
    bool vis[N];
    int cnt[N];

    void init() {
        e.clear();
        nd.clear();
        for (int i = 0; i <= n + 1; i++) {
            nd.pushback(node(inf, i, 0));
            g[i].clear();
            vis[i] = false;
        }
    }

    void add(int u, int v, LL w) {
        e.pushback(edge(u, v, w));
        e.pushback(edge(v, u, 0));
        g[u].pushback(m++);
        g[v].pushback(m++);
    }

    void bfs() {
        nd[t].hight = 0;
        qq.push(t);
        while (!qq.empty()) {
            int u = qq.front();
            qq.pop();
            vis[u] = false;
            for (auto j : g[u]) {
                int v = e[j].to;
                if (e[j].cap == 0 && nd[v].hight > nd[u].hight + 1) {
                    nd[v].hight = nd[u].hight + 1;
                    if (vis[v] == false) {
                        qq.push(v);
                        vis[v] = true;
                    }
                }
            }
        }
        return;
    }

    void _push(int u) {
        for (auto j : g[u]) {
            edge &ee = e[j], &er = e[j ^ 1];
            int v = ee.to;
            node &nu = nd[u], &nv = nd[v];
            if (ee.cap && nv.hight + 1 == nu.hight) {
```

```
                LL flow = std::min(ee.cap, nu.flow);
                ee.cap -= flow, er.cap += flow;
                nu.flow -= flow, nv.flow += flow;
                if (vis[v] == false && v != t && v != s) {
                    q.push(nv);
                    vis[v] = true;
                }
                if (nu.flow == 0) break;
            }
        }
    }

    void relabel(int u) {
        nd[u].hight = inf;
        for (auto j : g[u]) {
            int v = e[j].to;
            if (e[j].cap && nd[v].hight + 1 < nd[u].hight) {
                nd[u].hight = nd[v].hight + 1;
            }
        }
    }

    LL hlpp() {
        bfs();
        if (nd[s].hight == inf) return 0;
        nd[s].hight = n;
        for (int i = 1; i <= n; i++) {
            if (nd[i].hight < inf) cnt[nd[i].hight]++;
        }
        for (auto j : g[s]) {
            int v = e[j].to;
            int flow = e[j].cap;
            if (flow) {
                e[j].cap -= flow, e[j ^ 1].cap += flow;
                nd[s].flow -= flow, nd[v].flow += flow;
                if (vis[v] == false && v != s && v != t) {
                    q.push(nd[v]);
                    vis[v] = true;
                }
            }
        }
        while (!q.empty()) {
            int u = q.top().id;
            q.pop();
            vis[u] = false;
            _push(u);
            if (nd[u].flow) {
                cnt[nd[u].hight]--;
                if (cnt[nd[u].hight] == 0) {
                    for (int i = 1; i <= n; i++) {
                        if (i != s && i != t && nd[i].hight > nd[u].hight && nd[i].hight < n + 1) {
                            nd[i].hight = n + 1;
                        }
                    }
                }
                relabel(u);
                cnt[nd[u].hight]++;
                q.push(nd[u]);
                vis[u] = true;
            }
        }
        return nd[t].flow;
    }
} maxf;
```

10.17 network flow - minimum cost flow

Dinic + SPFA

```
struct edge {
    int from, to;
    LL cap, cost;

    edge(int u, int v, LL c, LL w) : from(u), to(v), cap(c), cost(w) {}
};

struct MCMF {
    int n, m = 0, s, t;
    std::vector<edge> e;
    vi g[N];
    int cur[N], vis[N];
```

```
    LL dist[N], minc;

    void init(int n) {
        for (int i = 0; i < n; i++) g[i].clear();
        e.clear();
        minc = m = 0;
    }

    void add(int from, int to, LL cap, LL cost) {
        e.push_back(edge(from, to, cap, cost));
        e.push_back(edge(to, from, 0, -cost));
        g[from].push_back(m++);
        g[to].push_back(m++);
    }

    bool spfa() {
        rep(i, 1, n) { dist[i] = INF, cur[i] = 0; }
        std::queue<int> q;
        q.push(s), dist[s] = 0, vis[s] = 1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            vis[u] = 0;
            for (int j = cur[u]; j < g[u].size(); j++) {
                edge& ee = e[g[u][j]];
                int v = ee.to;
                if (ee.cap && dist[v] > dist[u] + ee.cost) {
                    dist[v] = dist[u] + ee.cost;
                    if (!vis[v]) {
                        q.push(v);
                        vis[v] = 1;
                    }
                }
            }
        }
        return dist[t] != INF;
    }

    LL dfs(int u, LL now) {
        if (u == t) return now;
        vis[u] = 1;
        LL ans = 0;
        for (int& i = cur[u]; i < g[u].size() && ans < now; i++) {
            edge &ee = e[g[u][i]], &er = e[g[u][i] ^ 1];
            int v = ee.to;
            if (!vis[v] && ee.cap && dist[v] == dist[u] + ee.cost) {
                LL f = dfs(v, std::min(ee.cap, now - ans));
                if (f) {
                    minc += f * ee.cost, ans += f;
                    ee.cap -= f;
                    er.cap += f;
                }
            }
        }
        vis[u] = 0;
        return ans;
    }

    PLL mcmf() {
        LL maxf = 0;
        while (spfa()) {
            LL tmp;
            while ((tmp = dfs(s, INF))) maxf += tmp;
        }
        return std::makepair(maxf, minc);
    }
} minc_maxf;
```

Primal-Dual 原始对偶算法

```
/* primal dual */
struct edge {
    int from, to;
    LL cap, cost;

    edge(int u, int v, LL c, LL w) : from(u), to(v), cap(c), cost(w) {}
};

struct node {
    int v, e;

    node(int _v = 0, int _e = 0) : v(_v), e(_e) {}
};
```

```

const int maxn = 5000 + 10;

struct MCMF {
    int n, m = 0, s, t;
    std::vector<edge> e;
    vi g[maxn];
    int dis[maxn], vis[maxn], h[maxn];
    node p[maxn * 2];

    void add(int from, int to, LL cap, LL cost) {
        e.push_back(edge(from, to, cap, cost));
        e.push_back(edge(to, from, 0, -cost));
        g[from].push_back(m++);
        g[to].push_back(m++);
    }

    bool dijkstra() {
        std::priority_queue<PII, std::vector<PII>, std::greater<PII>> q;
        for (int i = 1; i <= n; i++) {
            dis[i] = inf;
            vis[i] = 0;
        }
        dis[s] = 0;
        q.push({0, s});
        while (!q.empty()) {
            int u = q.top().ss;
            q.pop();
            if (vis[u]) continue;
            vis[u] = 1;
            for (auto i : g[u]) {
                edge ee = e[i];
                int v = ee.to, nc = ee.cost + h[u] - h[v];
                if (ee.cap and dis[v] > dis[u] + nc) {
                    dis[v] = dis[u] + nc;
                    p[v] = node(u, i);
                    if (!vis[v]) q.push({dis[v], v});
                }
            }
        }
        return dis[t] != inf;
    }

    void spfa() {
        std::queue<int> q;
        for (int i = 1; i <= n; i++) h[i] = inf;
        h[s] = 0, vis[s] = 1;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            vis[u] = 0;
            for (auto i : g[u]) {
                edge ee = e[i];
                int v = ee.to;
                if (ee.cap and h[v] > h[u] + ee.cost) {
                    h[v] = h[u] + ee.cost;
                    if (!vis[v]) {
                        vis[v] = 1;
                        q.push(v);
                    }
                }
            }
        }
    }

    PLL mcmf() {
        LL maxf = 0, minc = 0;
        spfa();
        while (dijkstra()) {
            LL minf = INF;
            for (int i = 1; i <= n; i++) h[i] += dis[i];
            for (int i = t; i != s; i = p[i].v) minf = std::min(minf, e[p[i].e].cap);
            for (int i = t; i != s; i = p[i].v) {
                e[p[i].e].cap -= minf;
                e[p[i].e ^ 1].cap += minf;
            }
            maxf += minf;
            minc += minf * h[t];
        }
        return std::make_pair(maxf, minc);
    }
} minc_maxf;
```

存在负环的网络

流满后推流, 转化为上下界网络流.

10.18 network flow - minimal cut

最小割解决的问题是将图中的点集 V 划分成 S 与 T, 使得 S 与 T 之间的连边的容量总和最小.

最大流最小割定理

网络中 s 到 t 的最大流流量的值等于所要求的最小割的值, 所以求最小割只需要跑 Dinic 即可.

获得 S 中的所有点

在 Dinic 的 bfs 函数中, 每次将所有点的 d 数组值改为无穷大, 最后跑完最大流之后 d 数组不为无穷大的就是和源点一起在 S 集合中的点.

例子

最小割的本质是对图中点集进行 2-划分, 网络流只是求解答案的手段.

1. 在图中花费最小的代价断开一些边使得源点 s 无法流到汇点 t.

   直接跑最大流就得到了答案.

2. 在图中删除最少的点使得源点 s 无法流到汇点 t.

   对每个点进行拆点, 在 i 与 i' 之间建立容量为 1 的有向边.

10.19 network flow with upper / lower bound

10.19.1 无源汇上下界可行流

每条有向边有流量的上下界限制, 但整张图并未确定源点与汇点. 如果存在满足每个点的流入量等于流出量, 且每条边的流量满足其上下界限制的流, 称之为可行流.

1. 将每条边先给予大小为下界的流量,

2. 对每个点计算总流入量 in_u 与总流出量 out_u 的值,

3. 建立超级源点到每个点, 容量大小为 $\max\{0, \mathrm{in}_u - \mathrm{out}_u\}$ 的边; 建立每个点到超级汇点, 容量大小为 $\max\{0, \mathrm{out}_u - \mathrm{in}_u\}$,

4. 跑从超级源点到超级汇点的最大流, 如果超级源点每条边都流满意味着存在可行流. 将每条边的流量加上预先给每条边设置的下界流量即为可行流方案.

10.19.2 有源汇上下界可行流

1. 建立汇点 t 到源点 s 的, 容量为 ∞ 的有向边, 将其转化为无源汇的问题.

10.19.3 有源汇上下界最大流

1. 建立汇点 t 到源点 s 的, 容量为 ∞ 的有向边, 将其转化为无源汇的问题,
2. 跑上下界可行流, 可行流流量为边 $t \xrightarrow{\infty} s$ 的流量.
3. 删除 $t \xrightarrow{\infty} s$ 的边, 再残量网络上跑 s 到 t 的最大流,
4. 答案等于可行流流量 + 最大流流量.

10.19.4 有源汇上下界最小流

1. 建立汇点 t 到源点 s 的, 容量为 ∞ 的有向边, 将其转化为无源汇的问题,
2. 跑上下界可行流, 可行流流量为边 $t \xrightarrow{\infty} s$ 的流量.
3. 删除 $t \xrightarrow{\infty} s$ 的边, 再残量网络上跑 t 到 s 的最大流,
4. 答案等于可行流流量 - 最大流流量.

10.19.5 有源汇上下界最小费用可行流

1. 按下界流满并计算费用,
2. 类似有源汇上下界最大流建图, 跑超级源点到超级汇点的费用流,
3. 答案等于按下界的费用加上后续残量网络.

10.20 matching - matching on bipartite graph

二分图最大匹配

Kuhn-Munkres

时间复杂度: $O(n^3)$.

```
/* Kuhn-Munkres */
auto KM = [&](int n1, int n2, vvi e) -> std::pair<vi, vi> {
    vi vis(n2 + 1);
    vi l(n1 + 1, -1), r(n2 + 1, -1);
    std::function<bool(int)> dfs = [&](int u) -> bool {
        for (auto v : e[u]) {
            if (!vis[v]) {
                vis[v] = 1;
                if (r[v] == -1 or dfs(r[v])) {
                    r[v] = u;
                    return true;
                }
            }
        }
        return false;
    };
    for (int i = 1; i <= n1; i++) {
        std::fill(all(vis), 0);
        dfs(i);
    }
    for (int i = 1; i <= n2; i++) {
        if (r[i] == -1) continue;
        l[r[i]] = i;
    }
    return {l, r};
};
auto [mchl, mchr] = KM(n1, n2, e);
std::cout << mchl.size() - std::count(all(mchl), -1) << endl;
```

Hopcroft-Karp

据说时间复杂度是 $O(m\sqrt{n})$ 的, 但是快的飞起.

```
/* Hopcroft-Karp */
vpi e(m);
auto hopcroft_karp = [&](int n, int m, vpi& e) -> std::pair<vi, vi> {
    vi g(e.size()), l(n + 1, -1), r(m + 1, -1), d(n + 2);
    for (auto [u, v] : e) d[u]++;
    std::partial_sum(all(d), d.begin());
    for (auto [u, v] : e) g[--d[u]] = v;
    for (vi a, p, q(n + 1);;) {
        a.assign(n + 1, -1);
        p.assign(n + 1, -1);
        int t = 1;
        for (int i = 1; i <= n; i++) {
            if (l[i] == -1) {
                q[t++] = a[i] = p[i] = i;
            }
        }
        bool match = false;
        for (int i = 1; i < t; i++) {
            int u = q[i];
            if (l[a[u]] != -1) continue;
            for (int j = d[u]; j < d[u + 1]; j++) {
                int v = g[j];
                if (r[v] == -1) {
                    while (v != -1) {
                        r[v] = u;
                        std::swap(l[u], v);
                        u = p[u];
                    }
                    match = true;
                    break;
                }
                if (p[r[v]] == -1) {
                    q[t++] = v = r[v];
                    p[v] = u;
                    a[v] = a[u];
                }
            }
        }
        if (!match) break;
    }
    return {l, r};
};
```

二分图最大权匹配

Kuhn-Munkres

注意是否为完美匹配, 非完美选 0, 完美选 $-INF$. (存疑)

```
/* Kuhn-Munkres */
auto KM = [&](int n, vvl e) -> std::tuple<LL, vi, vi> {
    vl la(n + 1), lb(n + 1), pp(n + 1), vx(n + 1);
    vi l(n + 1, -1), r(n + 1, -1);
    vi va(n + 1), vb(n + 1);
    LL delta;
    auto bfs = [&](int x) -> void {
        int a, y = 0, y1 = 0;
        std::fill(all(pp), 0);
        std::fill(all(vx), INF);
        r[y] = x;
        do {
            a = r[y], delta = INF, vb[y] = 1;
            for (int b = 1; b <= n; b++) {
                if (!vb[b]) {
                    if (vx[b] > la[a] + lb[b] - e[a][b]) {
                        vx[b] = la[a] + lb[b] - e[a][b];
                        pp[b] = y;
                    }
                    if (vx[b] < delta) {
                        delta = vx[b];
                        y1 = b;
                    }
                }
            }
            for (int b = 0; b <= n; b++) {
```

```
                if (vb[b]) {
                    la[r[b]] -= delta;
                    lb[b] += delta;
                } else
                    vx[b] -= delta;
            }
            y = y1;
        } while (r[y] != -1);
        while (y) {
            r[y] = r[pp[y]];
            y = pp[y];
        }
    };
    for (int i = 1; i <= n; i++) {
        std::fill(all(vb), 0);
        bfs(i);
    }
    LL ans = 0;
    for (int i = 1; i <= n; i++) {
        if (r[i] == -1) continue;
        l[r[i]] = i;
        ans += e[r[i]][i];
    }
    return {ans, l, r};
};

auto [ans, mchl, mchr] = KM(n, e);
```

10.21 matching - matching on general graph

11 geometry

11.1 two demention

点与向量

```
struct Point {
    LL x = 0, y = 0;
    Point() = default;
    Point(long long x, long long y) : x(x), y(y) {}
    operator bool() { return *this != Point{}; }
    friend bool operator==(Point p, Point q) { return p.x == q.x and p.y == q.y; }
    friend bool operator!=(Point p, Point q) { return !(p == q); }
    friend Point operator+(Point p, Point q) { return {p.x + q.x, p.y + q.y}; }
    friend Point operator-(Point p, Point q) { return {p.x - q.x, p.y - q.y}; }
    friend LL dot(Point p, Point q) { return p.x * q.x + p.y * q.y; }
    friend LL det(Point p, Point q) { return p.x * q.y - q.x * p.y; }
    friend bool operator<(Point p, Point q) {
        return std::pair{p.quad(), det(q, p)} < std::pair{q.quad(), 0ll};
        return (p.x == q.x ? p.y < q.y : p.x < q.x);
    }
    int quad() const {
        if (x > 0 && y >= 0) return 1;
        if (x <= 0 and y > 0) return 2;
        if (x < 0 and y <= 0) return 3;
        if (x >= 0 and y < 0) return 4;
        return 0;
    }
    friend LL dist(Point p, Point q) {
        return (p.x - q.x) * (p.x - q.x) + (p.y - q.y) * (p.y - q.y);
    }
};
std::istream& operator>>(std::istream& is, Point& p) { return is >> p.x >> p.y; }
std::ostream& operator<<(std::ostream& os, Point p) {
    return os << '(' << p.x << ',' << p.y << ')';
}
```

线段

```
struct line {
    point a, b;

    line(point _a = {}, point _b = {}) { a = _a, b = _b; }

    /* 交点类型为 double */
    friend point iPoint(line p, line q) {
        point v1 = p.b - p.a;
        point v2 = q.b - q.a;
        point u = q.a - p.a;
        return q.a + (q.b - q.a) * ((u ^ v1) * 1. / (v1 ^ v2));
    }

    /* 极角排序 */
    bool operator<(const line& p) const {
        double t1 = std::atan2((b - a).y, (b - a).x);
        double t2 = std::atan2((p.b - p.a).y, (p.b - p.a).x);
        if (fabs(t1 - t2) > eps) {
            return t1 < t2;
        }
        return ((p.a - a) ^ (p.b - a)) > eps;
    }
};
```

11.2 convex

2D

```
/* andrew */
auto andrew = [&](std::vector<point>& v) -> std::vector<point> {
    std::sort(all(v));
```

```cpp
    std::vector<point> stk;
    for (int i = 0; i < n; i++) {
        point x = v[i];
        while (stk.size() > 1 and ((stk.end()[-1] - stk.end()[-2]) ^ (x - stk.end()[-2])) <= 0) {
            stk.pop_back();
        }
        stk.push_back(x);
    }
    int tmp = stk.size();
    for (int i = n - 2; i >= 0; i--) {
        point x = v[i];
        while (stk.size() > tmp and ((stk.end()[-1] - stk.end()[-2]) ^ (x - stk.end()[-2])) <= 0) {
            stk.pop_back();
        }
        stk.push_back(x);
    }
    return stk;
};
```

half plane

```cpp
/* half plane */
auto half_plane = [&](std::vector<line>& ln) -> std::vector<point> {
    std::sort(all(ln));
    ln.erase(
        unique(
            all(ln),
            [](line& p, line& q) {
                double t1 = std::atan2((p.b - p.a).y, (p.b - p.a).x);
                double t2 = std::atan2((q.b - q.a).y, (q.b - q.a).x);
                return fabs((t1 - t2)) < eps;
            }),
        ln.end());
    auto check = [&](line p, line q, line r) -> bool {
        point a = iPoint(p, q);
        return ((r.b - r.a) ^ (a - r.a)) < -eps;
    };
    line q[ln.size() + 2];
    int hh = 1, tt = 0;
    q[++tt] = ln[0];
    q[++tt] = ln[1];
    for (int i = 2; i < (int) ln.size(); i++) {
        while (hh < tt and check(q[tt - 1], q[tt], ln[i])) tt--;
        while (hh < tt and check(q[hh + 1], q[hh], ln[i])) hh++;
        q[++tt] = ln[i];
    }
    while (hh < tt and check(q[tt - 1], q[tt], q[hh])) tt--;
    while (hh < tt and check(q[hh + 1], q[hh], q[tt])) hh++;
    q[tt + 1] = q[hh];
    std::vector<point> ans;
    for (int i = hh; i <= tt; i++) {
        ans.push_back(iPoint(q[i], q[i + 1]));
    }
    return ans;
};
```

12 offline algorithm

12.1 discretization

```
std::sort(all(a));
a.erase(unique(all(a)), a.end());
auto get_id = [&](const int& x) -> int { return lower_bound(all(a), x) - a.begin() + 1; };
```

12.2 Mo algorithm

普通莫队

```
int block = n / sqrt(2 * m / 3);
std::sort(all(q), [&](node a, node b) {
    return a.l / block == b.l / block ? (a.r == b.r ? 0 : ((a.l / block) & 1) ^ (a.r < b.r))
                                        : a.l < b.l;
});
auto move = [&](int x, int op) -> void {
    if (op == 1) {
        /* operations */
    } else {
        /* operations */
    }
};
for (int k = 1, l = 1, r = 0; k <= m; k++) {
    node Q = q[k];
    while (l > Q.l) {
        move(--l, 1);
    }
    while (r < Q.r) {
        move(++r, 1);
    }
    while (l < Q.l) {
        move(l++, -1);
    }
    while (r > Q.r) {
        move(r--, -1);
    }
}
```

12.3 CDQ

n 个三维数对 (a_i, b_i, c_i), 设 $f(i)$ 表示 $a_j \leqslant a_i, b_j \leqslant b_i, c_j \leqslant c_i (i \neq j)$ 的个数. 输出 $f(i)$ $(0 \leqslant i \leqslant n-1)$ 的值.

```
// 洛谷 P3810 【模板】三维偏序（陌上花开）

struct data {
    int a, b, c, cnt, ans;

    data(int _a = 0, int _b = 0, int _c = 0, int _cnt = 0, int _ans = 0) {
        a = _a, b = _b, c = _c, cnt = _cnt, ans = _ans;
    }

    bool operator!=(data x) {
        if (a != x.a) return true;
        if (b != x.b) return true;
        if (c != x.c) return true;
        return false;
    }
};

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

    int n, k;
    std::cin >> n >> k;
    static data v1[N], v2[N];
    for (int i = 1; i <= n; i++) {
```

```
        std::cin >> v1[i].a >> v1[i].b >> v1[i].c;
    }
    std::sort(v1 + 1, v1 + n + 1, [&](data x, data y) {
        if (x.a != y.a) return x.a < y.a;
        if (x.b != y.b) return x.b < y.b;
        return x.c < y.c;
    });
    int t = 0, top = 0;
    for (int i = 1; i <= n; i++) {
        t++;
        if (v1[i] != v1[i + 1]) {
            v2[++top] = v1[i];
            v2[top].cnt = t;
            t = 0;
        }
    }
    vi tr(N);
    auto add = [&](int pos, int val) -> void {
        while (pos <= k) {
            tr[pos] += val;
            pos += lowbit(pos);
        }
    };
    auto query = [&](int pos) -> int {
        int ans = 0;
        while (pos > 0) {
            ans += tr[pos];
            pos -= lowbit(pos);
        }
        return ans;
    };
    std::function<void(int, int)> CDQ = [&](int l, int r) -> void {
        if (l == r) return;
        int mid = (l + r) >> 1;
        CDQ(l, mid), CDQ(mid + 1, r);
        std::sort(v2 + l, v2 + mid + 1, [&](data x, data y) {
            if (x.b != y.b) return x.b < y.b;
            return x.c < y.c;
        });
        std::sort(v2 + mid + 1, v2 + r + 1, [&](data x, data y) {
            if (x.b != y.b) return x.b < y.b;
            return x.c < y.c;
        });
        int i = l, j = mid + 1;
        while (j <= r) {
            while (i <= mid && v2[i].b <= v2[j].b) {
                add(v2[i].c, v2[i].cnt);
                i++;
            }
            v2[j].ans += query(v2[j].c);
            j++;
        }
        for (int ii = l; ii < i; ii++) {
            add(v2[ii].c, -v2[ii].cnt);
        }
        return;
    };
    CDQ(1, top);
    vi ans(n + 1);
    for (int i = 1; i <= top; i++) {
        ans[v2[i].ans + v2[i].cnt] += v2[i].cnt;
    }
    for (int i = 1; i <= n; i++) {
        std::cout << ans[i] << endl;
    }
    return 0;
}
```

13 Print All Cases

13.1 print all trees with n nodes

构造所有 n 个节点的树.

13.1.1 有根树

表示其数量的数列在 oeis 上编号为 A000081. $n = 1, 2, 3 \cdots, 20$ 的项分别为:

$$1, 1, 2, 4, 9,$$
$$20, 48, 115, 286, 719,$$
$$1842, 4766, 12486, 32973, 87811,$$
$$235381, 634847, 1721159, 4688676, 12826228.$$

构造所有 $n \leq 20$ 的有根树的 (平均) 运行时间为 15.7054s.

```cpp
/* integer partition */
int n = 5;
std::vector<vvi> part(n + 1);
auto integerPartition = [&](int n) {
    // part[1] = {{1}};
    for (int i = 1; i <= n; i++) {
        part[i].push_back({i});
        for (int j = 1; j < i; j++) {
            for (const auto& v : part[i - j]) {
                vi tmp = v;
                tmp.push_back(j);
                std::sort(all(tmp));
                part[i].push_back(tmp);
            }
        }
        std::sort(all(part[i]));
        part[i].erase(unique(all(part[i])), part[i].end());
    }
};
integerPartition(n);
/* find all trees */
std::vector<std::vector<std::string>> trees(n + 1);
auto allTrees = [&](int n) {
    std::string s;
    for (int i = 1; i < n; i++) s += '(';
    for (int i = 1; i < n; i++) s += ')';
    trees[n].push_back(s);
    for (const auto& v : part[n - 1]) {
        std::vector<std::string> now;
        auto dfs = [&](auto&& self, int i) {
            if (i == v.size()) {
                std::string s = "";
                auto tmp = now;
                std::sort(all(tmp));
                for (const auto& ss : tmp) s += '(' + ss + ')';
                trees[n].push_back(s);
                return;
            }
            for (const auto& s : trees[v[i]]) {
                now.push_back(s);
                self(self, i + 1);
                now.pop_back();
            }
        };
        dfs(dfs, 0);
    }
    std::sort(all(trees[n]));
    trees[n].erase(unique(all(trees[n])), trees[n].end());
};
for (int i = 1; i <= n; i++) {
    allTrees(i);
    debug(i, trees[i].size());
    std::cout << '\n';
}
for (const auto& s : trees[n]) {
    vvi e(n + 1);
    vi fa(n + 1);
    int cnt = 1, now = 1;
```

```
    for (const auto& c : s) {
        if (c == '(') {
            cnt += 1;
            e[now].push_back(cnt);
            e[cnt].push_back(now);
            fa[cnt] = now;
            now = cnt;
        } else {
            now = fa[now];
        }
    }
    debug(e);
    /* do the things you need */
}
```

14 Magic

14.1 magic heap

对顶堆维护中位数.

```
/* magic heap */
struct MagicHeap {
    LL suml = 0, sumr = 0;
    std::priority_queue<int> ql;
    std::priority_queue<int, std::vector<int>, std::greater<int>> qr;
    void le2ri() {
        auto x = ql.top();
        suml -= x, ql.pop();
        sumr += x, qr.push(x);
    };
    void ri2le() {
        auto x = qr.top();
        sumr -= x, qr.pop();
        suml += x, ql.push(x);
    };
    void pushL(int x) { suml += x, ql.push(x); }
    void pushR(int x) { sumr += x, qr.push(x); }
    void push(int x) {
        if (ql.empty()) {
            pushL(x);
        } else if (qr.empty()) {
            (x <= ql.top() ? le2ri(), pushL(x) : pushR(x));
        } else {
            int le = ql.top(), ri = qr.top();
            if (le <= x and x <= ri) {
                (ql.size() == qr.size() ? pushL(x) : pushR(x));
            } else if (x < le) {
                if (ql.size() != qr.size()) le2ri();
                pushL(x);
            } else {
                if (ql.size() <= qr.size()) ri2le();
                pushR(x);
            }
        }
    }
    int size() { return ql.size() + qr.size(); }
    bool empty() { return ql.empty() and qr.empty(); }
    LL val() { return suml + sumr; }
    LL mid() { return ql.top(); }
    LL dist() { return sumr - suml + ql.top() * (ql.size() - qr.size()); }
};
```

14.2 operator queue

双栈维护队列半群.

```
template <typename T, typename Op>
struct OpQueue {
    static_assert(std::is_convertible_v<std::invoke_result_t<Op, T, T>, T>);
    const T e;
    const Op op;
    std::vector<T> l, r, a;
    OpQueue(T e, Op op) : e(e), op(op), l{e}, r{e} {}
    T val() const { return op(l.back(), r.back()); }
    void push(T x) {
        r.push_back(op(r.back(), x));
        a.push_back(x);
    }
    void pop() {
        if (l.size() == 1) {
            for (; !a.empty(); a.pop_back()) {
                l.push_back(op(a.back(), l.back()));
            }
            r.resize(1);
        }
        assert(l.size() > 1);
        l.pop_back();
    }
    int size() const { return l.size() + r.size() - 2; }
    bool empty() const { return l.size() + r.size() == 2; }
};
```

```
/* When using this, remember to replace "T" with correct type and "e" with identity in the half group. */
auto op = [](T a, T b) -> T {
    /* You operations */
};
OpQueue<T, decltype(op)> a(e, op);
```