ACM/ICPC Template The Last Dance

浙江工商大学

最后一舞

May 18, 2023

Contents

0 头文件

0.1 header

```cpp
#include "bits/stdc++.h"
using namespace std;

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef vector<ll> VI;
typedef pair<int, int> pii;
typedef pair<double, double> pdd;
typedef pair<ll, ll> pll;

#define endl "\n"
#define fi first
#define se second
#define eb emplace_back
#define mem(a, b) memset(a , b , sizeof(a))

const ll INF = 0x3f3f3f3f;
const ll mod = 998244353;
const double eps = 1e-6;
const double PI = acos(-1);
const double R = 0.57721566490153286060651209;

void solve() {

}

signed main() {
    ios_base::sync_with_stdio(false);
    // cin.tie(nullptr);
    // cout.tie(nullptr);
#ifdef FZT_ACM_LOCAL
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    signed test_index_for_debug = 1;
    char acm_local_for_debug = 0;
    do {
        if (acm_local_for_debug == '$') exit(0);
        if (test_index_for_debug > 20)
            throw runtime_error("Check the stdin!!!");
        auto start_clock_for_debug = clock();
        solve();
        auto end_clock_for_debug = clock();
        cout << "Test " << test_index_for_debug << " successful" << endl;
        cerr << "Test " << test_index_for_debug++ << " Run Time: "
             << double(end_clock_for_debug - start_clock_for_debug) / CLOCKS_PER_SEC <<
     "s" << endl;
        cout << "--------------------------------------------------" << endl;
    } while (cin >> acm_local_for_debug && cin.putback(acm_local_for_debug));
#else
    solve();
#endif
    return 0;
}
```

1 字符串

1.1 hash

```
struct Hash {
using ui64 = unsigned long long;
    static constexpr int P = 1331;
    std::vector<ui64> h, p;

    Hash(std::string s) : h(s.size() + 1), p(s.size() + 1) {
        p[0] = 1;
        for (int i = 1;i <= (int) s.size(); i++) {
            p[i] = p[i - 1] * P;
            h[i] = h[i - 1] * P + s[i - 1] - '0';
        }
    }

    ui64 rangeSum(int l, int r) {
        return h[r] - h[l - 1] * p[r - l + 1];
    }
};
```

1.2 KMP

```
template <typename T>
std::vector<int> kmp_table(int n, const T &s) {
    std::vector<int> p(n, 0);
    int k = 0;
    for (int i = 1; i < n; i++) {
        while (k > 0 && !(s[i] == s[k])) {
            k = p[k - 1];
        }
        if (s[i] == s[k]) {
            k++;
        }
        p[i] = k;
    }
    return p;
}

template <typename T>
std::vector<int> kmp_table(const T &s) {
    return kmp_table((int) s.size(), s);
}

template <typename T>
std::vector<int> kmp_search(int n, const T &s, int m, const T &w, const std::vector<int
    > &p) {
    assert(n >= 1 && (int) p.size() == n);
    std::vector<int> res;
    int k = 0;
    for (int i = 0; i < m; i++) {
        while (k > 0 && (k == n || !(w[i] == s[k]))) {
            k = p[k - 1];
        }
        if (w[i] == s[k]) {
            k++;
        }
        if (k == n) {
            res.push_back(i - n + 1);
        }
    }
    return res;
    // returns 0-indexed positions of occurrences of s in w
}

template <typename T>
std::vector<int> kmp_search(const T &s, const T &w, const std::vector<int> &p) {
    return kmp_search((int) s.size(), s, (int) w.size(), w, p);
}
```

1.3 automaton

```
struct Automaton {
    static constexpr int ALPHABET_SIZE = 26;
    std::vector<std::vector<int>> tr;
    std::vector<int> e;
    std::vector<int> fail;
    int tot;

    Automaton(int n) : tr(n, std::vector<int>(ALPHABET_SIZE)), e(n), fail(n), tot(0) {}
    Automaton(int m, std::vector<std::string> s) : Automaton(m) {
        for(int i = 0;i < (int) s.size(); i++) {
            insert(s[i]);
        }
        build();
    }

    void insert(std::string s) {
        int u = 0;
        for(int i = 0;i < (int) s.size(); i++) {
            if(!tr[u][s[i] - 'a']) tr[u][s[i] - 'a'] = ++tot;
            u = tr[u][s[i] - 'a'];
        }
        e[u]++;
    }

    void build() {
        std::queue<int> q;
        for(int i = 0;i < 26; i++) {
            if(tr[0][i]) q.push(tr[0][i]);
        }
        while(q.size()) {
            int u = q.front();
            q.pop();
            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];
                }
            }
        }
    }

    int query(std::string t) {
        int u = 0, res = 0;
        for(int i = 0;i < (int) t.size(); i++) {
            u = tr[u][t[i] - 'a'];
            for(int j = u;j && e[j] != -1; j = fail[j]) {
                res += e[j], e[j] = -1;
            }
        }
        return res;
    }
};
```

1.4 manacher

```
template <typename T>
std::vector<int> manacher(int n, const T &s) {
    if (n == 0) {
        return std::vector<int>();
    }
    std::vector<int> res(2 * n - 1, 0);
    int l = -1, r = -1;
    for (int z = 0; z < 2 * n - 1; z++) {
        int i = (z + 1) >> 1;
        int j = z >> 1;
        int p = (i >= r ? 0 : std::min(r - i, res[2 * (l + r) - z]));
        while (j + p + 1 < n && i - p - 1 >= 0) {
            if (!(s[j + p + 1] == s[i - p - 1])) {
                break;
            }
            p++;
        }
        if (j + p > r) {
            l = i - p;
            r = j + p;
        }
        res[z] = p;
    }
    return res;
}

template <typename T>
std::vector<int> manacher(const T &s) {
    return manacher((int) s.size(), s);
}
```

1.5 Z-function

```
template <typename T>
std::vector<int> z_function(int n, const T &s) {
    std::vector<int> z(n, n);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        z[i] = (i > r ? 0 : std::min(r - i + 1, z[i - l]));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

template <typename T>
std::vector<int> z_function(const T &s) {
    return z_function((int) s.size(), s);
}

template <typename T>
std::vector<int> z_search(int n, const T &s, int m, const T &w, std::vector<int> &z) {
    assert(n >= 1 && (int) z.size() == n);
    std::vector<int> p(m);
    int l = 0, r = -1;
    for (int i = 0; i < m; i++) {
        p[i] = (i > r ? 0 : std::min(r - i + 1, z[i - l]));
        while (i + p[i] < m && p[i] < n && s[p[i]] == w[i + p[i]]) {
            p[i]++;
        }
        if (i + p[i] - 1 > r) {
            l = i;
            r = i + p[i] - 1;
        }
    }
    return p;
};

template <typename T>
std::vector<int> z_search(const T &s, const T &w, std::vector<int> &z) {
    return z_search((int) s.size(), s, (int) w.size(), w, z);
};
```

1.6 suffix-array

```
template <typename T>
std::vector<int> suffix_array(int n, const T &s, int char_bound) {
    std::vector<int> a(n);
    if (n == 0) {
        return a;
    }
    if (char_bound != -1) {
        std::vector<int> aux(char_bound, 0);
        for (int i = 0; i < n; i++) {
            aux[s[i]]++;
        }
        int sum = 0;
        for (int i = 0; i < char_bound; i++) {
            int add = aux[i];
            aux[i] = sum;
            sum += add;
        }
        for (int i = 0; i < n; i++) {
            a[aux[s[i]]++] = i;
        }
    } else {
        iota(a.begin(), a.end(), 0);
        sort(a.begin(), a.end(), [&s](int i, int j) { return s[i] < s[j]; });
    }
    std::vector<int> sorted_by_second(n);
    std::vector<int> ptr_group(n);
    std::vector<int> new_group(n);
    std::vector<int> group(n);
    group[a[0]] = 0;
    for (int i = 1; i < n; i++) {
        group[a[i]] = group[a[i - 1]] + (!(s[a[i]] == s[a[i - 1]]));
    }
    int cnt = group[a[n - 1]] + 1;
    int step = 1;
    while (cnt < n) {
        int at = 0;
        for (int i = n - step; i < n; i++) {
            sorted_by_second[at++] = i;
        }
        for (int i = 0; i < n; i++) {
            if (a[i] - step >= 0) {
                sorted_by_second[at++] = a[i] - step;
            }
        }
        for (int i = n - 1; i >= 0; i--) {
            ptr_group[group[a[i]]] = i;
        }
        for (int i = 0; i < n; i++) {
            int x = sorted_by_second[i];
            a[ptr_group[group[x]]++] = x;
        }
        new_group[a[0]] = 0;
        for (int i = 1; i < n; i++) {
            if (group[a[i]] != group[a[i - 1]]) {
                new_group[a[i]] = new_group[a[i - 1]] + 1;
            } else {
                int pre = (a[i - 1] + step >= n ? -1 : group[a[i - 1] + step]);
                int cur = (a[i] + step >= n ? -1 : group[a[i] + step]);
                new_group[a[i]] = new_group[a[i - 1]] + (pre != cur);
            }
        }
        swap(group, new_group);
        cnt = group[a[n - 1]] + 1;
        step <<= 1;
    }
    return a;
}

template <typename T>
std::vector<int> suffix_array(const T &s, int char_bound) {
    return suffix_array((int) s.size(), s, char_bound);
}

template <typename T>
std::vector<int> build_lcp(int n, const T &s, const std::vector<int> &sa) {
    assert((int) sa.size() == n);
    std::vector<int> pos(n);
    for (int i = 0; i < n; i++) {
        pos[sa[i]] = i;
    }
    std::vector<int> lcp(std::max(n - 1, 0));
    int k = 0;
    for (int i = 0; i < n; i++) {
        k = std::max(k - 1, 0);
        if (pos[i] == n - 1) {
            k = 0;
        } else {
            int j = sa[pos[i] + 1];
            while (i + k < n && j + k < n && s[i + k] == s[j + k]) {
                k++;
            }
            lcp[pos[i]] = k;
        }
    }
    return lcp;
}

template <typename T>
std::vector<int> build_lcp(const T &s, const std::vector<int> &sa) {
    return build_lcp((int) s.size(), s, sa);
}
```

1.7 suffixAutomaton

```
struct SuffixAutomaton {
    static constexpr int ALPHABET_SIZE = 26, N = 1e4;
    struct Node {
        int len;
        int link;
        int next[ALPHABET_SIZE];
        int siz;
        Node() : len(0), link(0), next{} {}
    } t[2 * N];
    int cntNodes;
    SuffixAutomaton() {
        cntNodes = 1;
        std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
        t[0].len = -1;
    }
    int extend(int p, int c) {
        if (t[p].next[c]) {
            int q = t[p].next[c];
            if (t[q].len == t[p].len + 1)
                return q;
            int r = ++cntNodes;
            t[r].len = t[p].len + 1;
            t[r].link = t[q].link;
            std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = ++cntNodes;
        t[cur].len = t[p].len + 1; t[cur].siz++;
        while (!t[p].next[c]) {
            t[p].next[c] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, c);
        return cur;
    }
};
```

2 动态规划

2.1 01knapSack

```
template <typename T>
T knapSack(const std::vector<int>& v, const std::vector<T>& w, int V) {
    int n = (int) v.size();
    std::vector<T> dp(V + 1);
    for (int i = 0; i < n; i++) {
        for (int j = V; j >= v[i]; j--) {
            dp[j] = std::max(dp[j], dp[j - v[i]] + w[i]);
        }
    }
    return dp.back();
}
```

2.2 simpleBoundedKnapSack

```
template <typename T>
T simpleMultipleKnapSack(const std::vector<int>& v, const std::vector<T>& w, const std
    ::vector<int>& cnt, int V) {
    int n = (int) v.size();
    std::vector<T> dp(V + 1);
    for (int i = 0; i < n; i++) {
        for (int j = V; j >= 0; j--) {
            for (int k = 1; k <= cnt[i]; k++) {
                if (j - k * v[i] < 0) break ;
                dp[j] = std::max(dp[j], dp[j - k * v[i]] + k * w[i]);
            }
        }
    }
    return dp.back();
}
```

2.3 binaryBoundedKnapSack

```
template <typename T>
T binaryMultipleKnapSack(const std::vector<int>& v, const std::vector<T>& w, const std
    ::vector<int>& cnt, int V) {
    int n = (int) v.size();
    std::vector<T> dp(V + 1);
    std::vector<std::array<T, 2>> bags;
    for (int i = 0; i < n; i++) {
        for (int k = 1; k <= cnt[i]; k <<= 1) {
            cnt[i] -= k;
            bags.push_back({v[i] * k, w[i] * k});
        }
        if (cnt[i] > 0) {
            bags.push_back({v[i] * cnt[i], w[i] * cnt[i]});
        }
    }
    for (auto& [nv, nw] : bags) {
        for (int j = V; j >= nv; j--) {
            dp[j] = std::max(dp[j], dp[j - nv] + nw);
        }
    }
    return dp.back();
}
```

2.4 monotonousQueueMultipleKnapSack

```
// no correct
template <typename T>
T monotonousQueueMultipleKnapSack(const std::vector<int>& v, const std::vector<T>& w, const std::vector<int>& cnt, int V) {
    int n = (int) v.size();
    std::vector<T> dp(V + 1);
    std::vector<int> que(V);
    for (int i = 0; i < n; i++) {
        std::vector<T> ndp(V + 1);
        for (int j = 0; j < v[i]; j++) {
            int head = 1, tail = 0;
            for (int k = j; k <= V; k += v[i]) {
                while (head <= tail && que[head] < k - cnt[i] * v[i]) head++;
                if (head <= tail) ndp[k] = std::max(dp[k], dp[que[head]] + (k - que[head]) / v[i] * w[i]);
                while (head <= tail && dp[k] >= dp[que[tail]] + (k - que[head]) / v[i] * w[i]) tail--;
                que[++tail] = k;
            }
        }
        dp = std::move(ndp);
    }
    return dp.back();
}
```

2.5 unboundedKnapSack

```
template <typename T>
T entireKnapSack(const std::vector<int>& v, const std::vector<T>& w, int V) {
    int n = (int) v.size();
    std::vector<T> dp(V + 1);
    for (int i = 0; i < n; i++) {
        for (int j = v[i]; j <= V; j++) {
            dp[j] = std::max(dp[j], dp[j - v[i]] + w[i]);
        }
    }
    return dp.back();
}
```

2.6 hybridKnapSack

```
template <typename T>
T hybridKnapSack(const std::vector<int>& v, const std::vector<T>& w, const std::vector<int>& cnt, int V) {
    int n = (int) v.size();
    std::vector<T> dp(V + 1);
    std::vector<std::tuple<int, T, int> > bags;
    for (int i = 0; i < n; i++) {
        if (cnt[i] < 0) {
            bags.push_back(std::make_tuple(v[i], w[i], -1));
        } else if (cnt[i] == 0) {
            bags.push_back(std::make_tuple(v[i], w[i], 0));
```

```cpp
        } else {
            for (int k = 1; k <= cnt[i]; k <<= 1) {
                cnt[i] -= k;
                bags.push_back(std::make_tuple(v[i] * k, w[i] * k, -1));
            }
            if (cnt[i] > 0) {
                bags.push_back(std::make_tuple(v[i] * cnt[i], w[i] * cnt[i], -1));
            }
        }
    }
    for (auto& bag : bags) {
        int v, op; T w;
        std::tie(v, w, op) = bag;
        if (op == -1) {
            for (int j = V; j >= v; j--) {
                dp[j] = std::max(dp[j], dp[j - v] + w);
            }
        } else {
            for (int j = v; j <= V; j++) {
                dp[j] = std::max(dp[j], dp[j - v] + w);
            }
        }
    }
    return dp.back();
}
```

2.7 groupKnapSack

```cpp
template <typename T>
T groupKnapSack(const std::vector<std::vector<std::array<int, 2>>>& groups, int V) {
    std::vector<T> dp(V + 1);
    for (auto& group : groups) {
        for (int j = V; j >= 0; j--) {
            for (auto& [v, w] : group) {
                if (j >= v) {
                    dp[j] = std::max(dp[j], dp[j - v] + w);
                }
                /*
                 if (j >= v) {
                    // from groups[i], ensure dp[i][j-v] has at least one item because dp[i][j-v] is not -1
                    if (~dp[i][j - v]) dp[i][j] = std::max(dp[i][j], dp[i][j - v] + w);
                    if (~dp[i - 1][j - v]) dp[i][j] = std::max(dp[i][j], dp[i - 1][j - v] + w);
                    // from groups[i - 1], ensure dp[i-1][j-v] has at least one item because dp[i-1][j-v] is not -1
                    // the order is not swap
                 }
                 */
            }
        }
    }
    return dp.back();
}
```

2.8 KnapSack2d

```cpp
template <typename T>
T twoDimensionalKnapSack(const std::vector<int>& v, const std::vector<int>& m, const std::vector<T>& w, int V, int M) {
    int n = (int) v.size();
    std::vector<std::vector<T>> dp(V + 1, std::vector<T>(M + 1));
    for (int i = 0; i < n; i++) {
        for (int j = V; j >= v[i]; j--) {
            for (int k = M; k >= m[i]; k--) {
                dp[j][k] = std::max(dp[j][k], dp[j - v[i]][k - m[i]] + w[i]);
            }
        }
    }
    return dp.back().back();
}
```

2.9 treeKnapSack

```cpp
template <typename T>
T treeKnapSack(const forest<T>& g) {
    std::vector<int> siz(n);
    std::vector<std::vector<T>> dp(n);

    std::function<void(int, int)> dfs = [&](int u, int fa) {
        siz[u] = 1;
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (to == fa) continue ;
            dfs(to, u);
            int now = min(siz[u] + siz[v] + 1, M);
            int t[MAX_M]; for (int i = 0; i <= M; i++) t[i] = INF/-INF;//初始化
            for (int i = 0; i <= siz[u]; i++)
                for (int j = 0; j <= siz[v] && i + j <= M; j++) {
                    //...转移方程
                }
            for (int i = 0; i <= now; i++) f[u][i] = min/max(f[u][i], t[i]);
            siz[u] = now;
        }
    };
}
```

2.10 LIS

```cpp
template <typename T>
int lis(const std::vector<T>& a) {
    std::vector<T> u;
    for (const T& x : a) {
        auto it = upper_bound(u.begin(), u.end(), x);
        if (it == u.end()) {
            u.push_back(x);
        } else {
            *it = x;
        }
    }
    return (int) u.size();
}
```

2.11 digitDP

```
// the numbers of exist 49 in 0~n
template <typename T>
T digitDP(T n) {
    T x = n;
    std::vector<int> digit;
    while (x) {
        digit.push_back(x % 10);
        x /= 10;
    }

    std::vector<std::vector<T>> dp(digit.size(), std::vector<T>(10));
    std::function<T(int, int, bool limit)> dfs = [&](int pos, int pre, bool limit) -> int {
        if (pos == -1) return 1;
        if (!limit && dp[pos][pre]) return dp[pos][pre];
        int up = limit ? digit[pos] : 9;
        T ans = 0;
        for (int i = 0; i <= up; i++) {
            if (pre == 4 && i == 9) {
                continue;
            }
            ans += dfs(pos - 1, i, limit && i == digit[pos]);
        }
        if (!limit) dp[pos][pre] = ans;
    };

    return dfs(digit.size() - 1, 0, 1);
}
```

2.12 bitmaskingDP

```
// 杭电1565

int n;
int a[22][22];
int dp[22][1 << 18]; // 第一维是行数，第二位是该行的方案数，继承了前面所有行数的方案数
int tot[1 << 18]; // 方案数

int calc(int i, int k)
{
    int cnt = 1, res = 0;
    while(k)
    {
        if(k & 1) res += a[i][cnt];
        k >>= 1;
        cnt++;
    }
    return res;
}

void solve()
{
    while(cin >> n) {
        mem(dp, 0);
        int cnt = 0;

        for (int i = 0; i <= (1 << n) - 1; i++) { // 预处理
            if ((i & (i >> 1)) == 0) // 判断i这个二进制是否满足相邻没有两个1的条件
                tot[++cnt] = i;
        }

        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                cin >> a[i][j];

        for (int i = 1; i <= n; i++) { // 行遍历
            for (int k = 1; k <= cnt; k++) { // 第i行k的二进制排列的数，与下面的j进行&
                int val = calc(i, tot[k]); // 计算k的二进制中1所在a数组里的权值
                for (int j = 1; j <= cnt; j++) { // 第i-1行j的二进制排列的数，与上面的k进行&并进行状态转移
                    if ((tot[j] & tot[k]) == 0)
                        dp[i][k] = max(dp[i][k], dp[i - 1][j] + val);
                }
            }
        }

        int ans = -1;
        for (int j = 1; j <= cnt; j++)
            ans = max(ans, dp[n][j]);

        cout << ans << endl;
    }
}
```

2.13 quadrilateralOptimization

```
//四边形优化区间dp(n^3 -> n^2)
//a < b < c < d, f[l][r] = min(f[l][k] + f[k + 1][r] + cost(l, r))
//1. cost(b, c) <= cost(a, d)
//2. cost(a, c) + cost(b, d) <= cost(a, d) + cost(b, c), 即交叉小于包含

template <typename T>
void quadrilateralOptimization() {
    for (int len = 2; len <= n; len++) {
        for (int l = 1, r; l + len - 1 <= n; l++) {
            r = l + len - 1;
            mn[l][r] = 0x3f3f3f3f;
            for (int k = m[l][r - 1]; k <= m[l + 1][r]; k++)
                if (mn[l][k] + mn[k + 1][r] + cost(l, r) < mn[l][r]) {
                    mn[l][r] = mn[l][k] + mn[k + 1][r] + cost(l, r);
                    m[l][r] = k;
                }
        }
    }
}
```

2.14 baseRingTreeDP

```
int flag, S, E;//flag是否找到环，SE为环上两个点

void findCircle(int u, int fa) {
    vis[u] = 1;
    for (int i = head[u], v; i; i = e[i].nxt)
```

```
        if ((v = e[i].to) != fa) {
            if (vis[v]) flag = 1, S = u, E = v;
            else findCircle(v, u);
        }
}

void dp(int u, int fa) {
    //dp过程

    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa && v) {
            dp(v, u);

        }
}

ll calc(int u) {
    flag = 0;
    findCircle(u, 0);
    if (flag) {
        for (int i = head[S], v; i; i = e[i].nxt)
            if ((v = e[i].to) == E) {
                e[i].to = e[i ^ 1].to = 0;//删边操作，注意e[tot]中tot从2开始
                break;
            }
        ll res = 0;
        dp(S, 0); res = max(res, ...);
        dp(E, 0); res = max(res, ...);
        return res;
    }
    else {
        dp(u, 0);
        return ...;
    }
}
```

2.15 segmentTreeDP

```
#define lc u << 1
#define rc u << 1 | 1
#define mid (t[u].l + t[u].r) / 2

const int N = 3e5 + 10;
const int K = 100 + 10;
int n, k;
int dp[N][K];

struct Tree {
    int l, r;
    int mx;
    int tag;
}t[N << 2];

inline void push_up(int u) {
    t[u].mx = max(t[lc].mx, t[rc].mx);
}

inline void push_down(int u) {
    if(!t[u].tag) return ;
    t[lc].tag += t[u].tag;
    t[rc].tag += t[u].tag;
    t[lc].mx  += t[u].tag;
    t[rc].mx  += t[u].tag;
    t[u].tag = 0;
}

void build(int u, int l, int r, int k) {
    t[u].l = l; t[u].r = r;
    t[u].tag = t[u].mx = 0;
    if(l == r) {
        t[u].mx = dp[l][k];
        return ;
    }
    int m = (l + r) / 2;
    build(lc, l, m, k);
    build(rc, m + 1, r, k);
    push_up(u);
}

void modify(int u, int ql, int qr, int val) {
    if(ql <= t[u].l && t[u].r <= qr) {
        t[u].mx += val;
        t[u].tag += val;
        return ;
    }
    int ans = -INF;
    push_down(u);
    if(ql <= mid) modify(lc, ql, qr, val);
    if(qr  > mid) modify(rc, ql, qr, val);
    push_up(u);
}

int query(int u, int ql, int qr) {
    if(ql <= t[u].l && t[u].r <= qr) return t[u].mx;
    push_down(u);
    int ans = 0;
    if(ql <= mid) ans = max(ans, query(lc, ql, qr));
    if(qr  > mid) ans = max(ans, query(rc, ql, qr));
    return ans;
}

void solve() {
    cin >> n >> k;
    vector<int> a(n + 1), pre(n + 1), from(n + 1);
    set<int> s;
    for(int i = 1;i <= n; i++) {
        cin >> a[i];
        s.insert(a[i]);
        from[i] = pre[a[i]]; pre[a[i]] = i;
        dp[i][1] = (int)s.size();
    }
    for(int i = 2;i <= k; i++) {
        build(1, 1, n, i - 1);
        for(int j = i;j <= n; j++) {
            modify(1, from[j], j - 1, 1);
            dp[j][i] = max(dp[j - 1][i - 1] + 1, query(1, i - 1, j - 1));
        }
    }
    cout << dp[n][k] << endl;
}
```

2.16 LCS

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int n; cin >> n;
  vector<int> p(n + 1), q(n + 1);
  for(int i = 1;i <= n; i++) cin >> p[i];
  for(int i = 1;i <= n; i++) cin >> q[i];

  vector<int> id(n + 1);
  for(int i = 1;i <= n; i++) id[q[i]] = i;
  vector<int> lis;
  // 将b中与a中的元素相关的位置，从大到小push进lis中
  for(int i = 1;i <= n; i++) {
    vector<int> d;
    for(int j = p[i];j <= n; j += p[i]) {
      d.emplace_back(id[j]);
    }
    sort(d.begin(), d.end(), greater<int>());
    for(auto x : d) lis.emplace_back(x);
  }

  vector<int> f;
  f.push_back(lis[0]);
  for(int i = 1;i < (int)lis.size(); i++) {
    if(lis[i] > f.back()) f.push_back(lis[i]);
    else {
      int pos = lower_bound(f.begin(), f.end(), lis[i]) - f.begin();
      f[pos] = lis[i];
    }
  }

  cout << f.size() << endl;
}
```

3 数据结构

3.1 BTree

```
template<class T>

struct TreeNode {
    T value;
    TreeNode *left;
    TreeNode *right;
};

template<class T>
TreeNode<T> *createTree(const T *pre, const T *in, const int len) {
    TreeNode<T> *t = NULL;
    if (len > 0) {
        t = new TreeNode<T>;
        t->value = pre[0];
        int index;
        for (index = 0; index < len; index++) {
            if (in[index] == pre[0]) {
                break;
            }
        }
        if (index == len) {
            index = -1;
        }
        t->left = createTree(pre + 1, in, index);
        t->right = createTree(pre + index + 1, in + index + 1, len - index - 1);
    }
    return t;
}

template<class T>
int preOrder(TreeNode<T> *root, queue<T> &out) {
    if (root) {
        int count = 1;
        out.push(root->value);
        count += preOrder(root->left, out);
        count += preOrder(root->right, out);
        return count;
    } else {
        return 0;
    }
}

template<class T>
int inOrder(TreeNode<T> *root, queue<T> &out) {
    if (root) {
        int count = 1;
        count += inOrder(root->left, out);
        out.push(root->value);
        count += inOrder(root->right, out);
        return count;
    } else {
        return 0;
    }
}

```

```
template<class T>
void postOrder(TreeNode<T> *root, queue<T> &out) {
    if (root) {
        postOrder(root->left, out);
        postOrder(root->right, out);
        out.push(root->value);
    } else {
        return;
    }
}

template<class T>
T *convertQueueToArray(queue<T> &out, int len) {
    T *list = new T[len];
    int now = 0;
    while (!out.empty() && now < len) {
        list[now] = out.front();
        out.pop();
        now++;
    }
    return list;
}

template<class T>
void destroyTree(TreeNode<T> *root) {
    if (root) {
        destroyTree(root->left);
        destroyTree(root->right);
        delete root;
    } else return;
}

template<class T>
void insertIntoBSTree(TreeNode<T> *root, const T &value) {
    if (!root) {
        return;
    }
    if (value < root->value) {
        if (root->left) {
            insertIntoTree(root->left, value);
        } else {
            root->left = new TreeNode<T>;
            root->left->value = value;
            root->left->left = NULL;
            root->left->right = NULL;
        }
    } else if (value > root->value) {
        if (root->right) {
            insertIntoTree(root->right, value);
        } else {
            root->right = new TreeNode<T>;
            root->right->value = value;
            root->right->left = NULL;
            root->right->right = NULL;
        }
    }
}

template<class T>
```

```
TreeNode<T> *createBSTree(T *list, int len) {
    if (len < 1) {
        return NULL;
    }
    TreeNode<T> *root = new TreeNode<char>;
    root->value = list[0];
    root->left = NULL;
    root->right = NULL;
    for (int i = 1; i < len; i++) {
        insertIntoBSTree(root, list[i]);
    }
    return root;
}
```

3.2 pbds-tree

```
// RBTree 红黑树
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
// 红黑树
__gnu_pbds::tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> t;
// null_type无映射(低版本g++为null_mapped_type)
// 类似multiset
__gnu_pbds::tree<int, null_type, less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update> t;
find_by_order(size_t order);
// 结点更新
tree_order_statistics_node_update
insert(p);
erase(it);
// 求k在树中是第几小,假设插入当前值判断当前值是第几小,最小为第0小
order_of_key(p);
// 找到第order小的迭代器
find_by_order(order);
// 前驱
lower_bound(p);
// 后驱
upper_bound(p);
// 合并
a.join(b);
// 分割 key小于等于v的元素属于a, 其余的属于b
a.split(v, b);

// 优先队列
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp>
// 操作类似于stl的优先队列
typedef __gnu_pbds::priority_queue<node, greater<node>, __gnu_pbds::thin_heap_tag> heap
    ;
heap::point_iterator; // 指向元素的指针
```

3.3 fenwick

```
template <typename T>
class fenwick {
public:
    std::vector<T> fenw;
    int n;

    fenwick(int _n) : n(_n) {
        fenw.resize(n);
    }

    void modify(int x, T v) {
        while (x < n) {
            fenw[x] += v;
            x |= (x + 1);
        }
    }

    T get(int x) {
        T v{};
        while (x >= 0) {
            v += fenw[x];
            x = (x & (x + 1)) - 1;
        }
        return v;
    }
};
```

3.4 fenwick2d

```
template <typename T>
class Fenwick2d {
public:
    std::vector<std::vector<T>> fenw;
    const int n, m;

    Fenwick2d(int _n, int _m) : n(_n), m(_m) {
        fenw.resize(n);
        for (int i = 0; i < n; i++) {
            fenw[i].resize(m);
        }
    }

    inline void modify(int i, int j, T v) {
        int x = i;
        while (x < n) {
            int y = j;
            while (y < m) {
                fenw[x][y] += v;
                y |= (y + 1);
            }
            x |= (x + 1);
        }
    }

    inline T get(int i, int j) {
        T v{};
        int x = i;
        while (x >= 0) {
            int y = j;
            while (y >= 0) {
                v += fenw[x][y];
                y = (y & (y + 1)) - 1;
            }
            x = (x & (x + 1)) - 1;
        }
        return v;
    }
};
```

3.5 SegmentTree

```
struct Info {

};

Info operator+(const Info& a, const Info& b) {

}

template<class Info,
    class Merge = std::plus<Info>>
struct SegmentTree {
    const int n;
    const Merge merge;
    std::vector<Info> info;
    SegmentTree(int n) : n(n), merge(Merge()), info(4 * n + 10) {}
    SegmentTree(std::vector<Info> init) : SegmentTree(init.size()) {
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = merge(info[2 * p], info[2 * p + 1]);
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
};
```

3.6 SegmentTree2d

```
#define lc u << 1
#define rc u << 1 | 1
#define m (l + r) / 2

const int N = 1e3 + 10;

struct Tree_y {
    int mx, mn;

    Tree_y operator + (const Tree_y &rhs) {
        Tree_y ans;
        ans.mx = max(mx, rhs.mx);
        ans.mn = min(mn, rhs.mn);
        return ans;
    }
};

int leafx[N], leafy[N];

struct Tree_x {
    Tree_y ty[N << 2];

    void build(int u, int l, int r) {
        ty[u].mx = -INF; ty[u].mn = INF;
        if(l == r) {
            leafy[l] = u;
            return ;
        }
        build(lc, l, m);
        build(rc, m + 1, r);
    }

    Tree_y query(int u, int l, int r, int ql, int qr) {
        if(qr < l || r < ql) return (Tree_y) {-INF, INF};
        if(ql <= l && r <= qr) return ty[u];
        return query(lc, l, m, ql, qr) + query(rc, m + 1, r, ql, qr);
    }
}tx[N << 2];

int n;

void build(int u, int l, int r) {
    tx[u].build(1, 1, n);
    if(l == r) {
        leafx[l] = u;
        return ;
    }
    build(lc, l, m);
    build(rc, m + 1, r);
}

// (x,y)单点更新，首先更新叶子节点，然后向上合并父亲节点
void modify(int x, int y, int val) {
    int valx = leafx[x];
    int valy = leafy[y];
    tx[valx].ty[valy].mn = tx[valx].ty[valy].mx = val;
    for(int i = valx; i; i >>= 1) {
        for(int j = valy; j; j >>= 1) {
            if(i == valx && j == valy) continue ;
            if(j == valy) {
                // 如果当前更新的列就是需要更新的叶子节点，那么由当前行的两个儿子节点来更新
                tx[i].ty[j] = tx[i << 1].ty[j] + tx[i << 1 | 1].ty[j];
            }
            else {
                tx[i].ty[j] = tx[i].ty[j << 1] + tx[i].ty[j << 1 | 1];
            }
        }
    }
}

Tree_y query(int u, int l, int r, int ql, int qr, int qx, int qy) {
    if(qr < l || r < ql) return (Tree_y) {-INF, INF};
    if(ql <= l && r <= qr) return tx[u].query(1, 1, n, qx, qy);
    return query(lc, l, m, ql, qr, qx, qy) + query(rc, m + 1, r, ql, qr, qx, qy);
}
```

3.7 ValueSegmentTree

```
// 权值线段树，相当于一个桶，每个节点用来表示一个区间的数***出现的次数***。

#include <bits/stdc++.h>

using namespace std;

#define lc u << 1
#define rc u << 1 | 1
#define m (l + r) / 2
#define mid (t[u].l + t[u].r) / 2

const int N = 1e5 + 10;

int t[N << 2];

int a[N];

void push_up(int u) {
    t[u] = t[lc] + t[rc];
}

void build(int u, int l, int r) {
    if(l == r) {
        t[u] = a[l]; // a[l]表示数为l的个数
        return ;
    }
    build(lc, l, m);
    build(rc, m + 1, r);
    push_up(u);
}

void update(int u, int l, int r, int k, int cnt) { // k这个数的个数增加cnt
    if(l == r) {
        t[u] += cnt;
        return ;
    }
    if(k <= m)
        update(lc, l, m, k, cnt);
    else
        update(rc, m + 1, r, k, cnt);
    push_up(u);
}

int query(int u, int l, int r, int k) { // 查询k这个数的个数
    if(l == r) {
        return t[u];
    }
    if(k <= m)
        return query(lc, l, m, k);
    else
        return query(rc, m + 1, r, k);
}

int k_max_th(int u, int l, int r, int k) { // 查询第k大的值
    if(l == r) {
        return l;
    }
    if(t[lc] >= k) return k_max_th(lc, l, m, k);
    else return k_max_th(rc, m + 1, r, k - t[lc]);
}
```

3.8 线段树动态开点合并分裂

```
#define mid (l+r)/2
static const int MAX_N = N * 40;
int rt[N], now;
int lc[MAX_N], rc[MAX_N];
ll sum[MAX_N];
int tot, rub[MAX_N];
int newNode() { return rub[0] ? rub[rub[0]--] : ++tot; }
void remove(int &u) {
    lc[u] = rc[u] = sum[u] = 0;
    rub[++rub[0]] = u;
    u = 0;
}
void push_up(int u) { sum[u] = sum[lc[u]] + sum[rc[u]]; }
void build(int &u, int l, int r) {
    u = newNode();
    if (l == r) {
        sum[u] = cnt[l];
        return;
    }
    build(lc[u], l, mid); build(rc[u], mid + 1, r);
    push_up(u);
}
void update(int &u, int l, int r, int p, ll k) {
    if (!u) u = newNode();
    if (l == r) {
        sum[u] += k;
        return;
    }
    if (p <= mid) update(lc[u], l, mid, p, k);
    else update(rc[u], mid + 1, r, p, k);
    push_up(u);
}
ll querySum(int u, int l, int r, int ql, int qr) {
    if (!u) return 0;
    if (ql <= l && r <= qr) return sum[u];
    ll res = 0;
    if (ql <= mid) res += querySum(lc[u], l, mid, ql, qr);
    if (qr > mid) res += querySum(rc[u], mid + 1, r, ql, qr);
    return res;
}
int queryKth(int u, int l, int r, ll k) {
    if (l == r) return l;
    if (k <= sum[lc[u]]) return queryKth(lc[u], l, mid, k);
    else return queryKth(rc[u], mid + 1, r, k - sum[lc[u]]);
}
void merge(int u, int v, int l, int r) {
    if (l == r) {
        sum[u] += sum[v];
        return;
    }
    if (lc[u] && lc[v]) merge(lc[u], lc[v], l, mid), remove(lc[v]);
    else if (lc[v]) lc[u] = lc[v], lc[v] = 0;
    if (rc[u] && rc[v]) merge(rc[u], rc[v], mid + 1, r), remove(rc[v]);
    else if (rc[v]) rc[u] = rc[v], rc[v] = 0;
    push_up(u);
}
void split(int &newp, int &u, int l, int r, int ql, int qr) {//分裂出[ql, qr]间的点
    if (!u) return;
    if (ql <= l && r <= qr) {
        newp = u;
        u = 0;
        return;
    }
    if (!newp) newp = newNode();
    if (ql <= mid) split(lc[newp], lc[u], l, mid, ql, qr);
    if (qr > mid) split(rc[newp], rc[u], mid + 1, r, ql, qr);
    push_up(u);
    push_up(newp);

}
#undef mid
```

3.9 线段树维护 LIS 方案数

```
// 线段树维护序列总LIS的长度mx.fi和方案数mx.se
// 以及对于每个点，可以存在于多少个LIS种

// https://nanti.jisuanke.com/t/39611

namespace Tree_LIS {
    const int N = 1e6 + 10;

#define lc t[u].l
#define rc t[u].r
#define mid (l + r) / 2

    struct Tree {
        int l, r;
        ll len; // 长度
        ll sum; // 个数
    }t[N << 2];

    int root, cnt;
    void init() {
        mem(t, 0);
        cnt = root = 0;
    }

    void push_up(int u) {
        if(t[lc].len == t[rc].len) {
            t[u].len = t[lc].len;
            t[u].sum = (t[lc].sum + t[rc].sum) % mod;
        }
        else if(t[lc].len < t[rc].len) {
            t[u].len = t[rc].len;
            t[u].sum = t[rc].sum;
        }
        else {
            t[u].len = t[lc].len;
            t[u].sum = t[lc].sum;
        }
    }

    void modify(int &u, int l, int r, int p, int le, int su) {
        if(!u) u = ++cnt;
        if(l == r) {
            if(t[u].len == le) t[u].sum = (t[u].sum + su) % mod;
            else if(t[u].len < le) {
                t[u].len = le;
                t[u].sum = su;
            }
            return ;
        }
        if(!lc) lc = ++cnt;
        if(!rc) rc = ++cnt;
        if(p <= mid) modify(lc, l, mid, p, le, su);
        else modify(rc, mid + 1, r, p, le, su);
        push_up(u);
    }

    pll query(int u, int l, int r, int ql, int qr) {
        if(ql <= l && r  <= qr) return pll{t[u].len, t[u].sum};
        pll lson = {0, 0}, rson = {0, 0};
        if(!lc) lc = ++cnt;
        if(!rc) rc = ++cnt;
        if(ql <= mid) lson = query(lc, l, mid, ql, qr);
        if(qr  > mid) rson = query(rc, mid + 1, r, ql, qr);
        if(lson.fi == rson.fi) return pll{lson.fi, (lson.se + rson.se) % mod};
        else if(lson.fi < rson.fi) return rson;
        else return lson;
    }
};

using namespace Tree_LIS;

ll quick_pow(ll a, ll b) {
    ll ans = 1;
    while(b) {
        if(b & 1) ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans % mod;
}

void solve() {
    int n; cin >> n;
    int L = 0, R = 1e9 + 7;
    vector<int> a(n + 1);
    vector<pll> l(n + 1), r(n + 1);
    for(int i = 1;i <= n; i++) cin >> a[i];
    init();
    modify(root, L, R, a[1], 1, 1);
    l[1] = {1, 1};
    for(int i = 2;i <= n; i++) {
        pll temp = query(root, L, R, 0, a[i] - 1);
        if(temp.fi == 0) temp = {0, 1};
        modify(root, L, R, a[i], temp.fi + 1, temp.se);
        l[i] = {temp.fi + 1, temp.se};
    }
    pll mx = query(root, L, R, 0, R);
    init();
    modify(root, L, R, R - a[n], 1, 1);
    r[n] = {1, 1};
    for(int i = n - 1;i >= 1; i--) {
        pll temp = query(root, L, R, 0, R - a[i] - 1);
        if(temp.fi == 0) temp = {0, 1};
        modify(root, L, R, R - a[i], temp.fi + 1, temp.se);
        r[i] = {temp.fi + 1, temp.se};
    }
    for(int i = 1;i <= n; i++) {
        if(r[i].fi + l[i].fi - 1 == mx.fi) {
            cout << (r[i].se * l[i].se % mod * quick_pow(mx.se, mod - 2) % mod + mod) % mod << " ";
        }
        else cout << 0 << " ";
    }
    cout << endl;
}
```

3.10 线段树维护最小字典序 LIS

```
// 线段树维护LIS输出字典序最小的路径
const int N = 1e5 + 10;

#define lc u << 1
#define rc u << 1 | 1
#define mid (t[u].l + t[u].r) / 2

struct Tree {
    int l, r;
    int mx;
    int id;
}t[N << 2];

inline void push_up(int u) {
    if (t[lc].mx > t[rc].mx)
        t[u].mx = t[lc].mx, t[u].id = t[lc].id;
    else if (t[lc].mx < t[rc].mx)
        t[u].mx = t[rc].mx, t[u].id = t[rc].id;
    else
        t[u].mx = t[lc].mx, t[u].id = min(t[lc].id, t[rc].id);
}

void build(int u, int l, int r) {
    t[u].l = l;
    t[u].r = r;
    t[u].mx = t[u].id = 0;
    if (l == r)
        return;
    int m = (l + r) >> 1;
    build(lc, l, m);
    build(rc, m + 1, r);
    push_up(u);
}

void modify(int u, int ql, int qr, int val, int id) {
    if (ql <= t[u].l && t[u].r <= qr) {
        if (t[u].mx < val || (t[u].mx == val && t[u].id > id)) {
            t[u].mx = val;
            t[u].id = id;
        }
        return;
    }
    if (ql <= mid)
        modify(lc, ql, qr, val, id);
    if (qr > mid)
        modify(rc, ql, qr, val, id);
    push_up(u);
}

pii query(int u, int ql, int qr) {
    if (ql <= t[u].l && t[u].r <= qr)
        return pii{t[u].mx, t[u].id};
    pii lson = {-1, -1}, rson = {-1, -1};
    if (ql <= mid)
        lson = query(lc, ql, qr);
    if (qr > mid)
        rson = query(rc, ql, qr);
    if (lson.x > rson.x)
        return lson;
    else if (lson.x < rson.x)
        return rson;
    else
        return {lson.x, min(lson.y, rson.y)};
}

void solve() {
    int n;
    cin >> n;
    assert(1 <= n && n <= 1e5);
    build(1, 1, 1e5);
    vector<int> a(n + 1), ans(n + 1), fa(n + 1);
    pii res = {0, 0};
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        assert(1 <= a[i] && a[i] <= 1e5);
        if (a[i] == 1) {
            fa[i] = 0;
            ans[i] = 1;
            modify(1, a[i], a[i], 1, i);
            continue;
        }
        pii temp = query(1, 1, a[i] - 1);
        ans[i] = temp.x + 1;
        fa[i] = temp.y;
        modify(1, a[i], a[i], ans[i], i);
        if (res.x < ans[i])
            res = pii{ans[i], i};
    }
    vector<int> v;
    int tt = res.second;
    while (tt) {
        v.push_back(tt);
        tt = fa[tt];
    }
    cout << v.size() << endl;
    for (int i = v.size() - 1; i >= 0; i--) {
        cout << v[i] << (i == 0 ? endl : " ");
    }
}


// 线段树维护LIS方案数
```

3.11 线段树维护插队问题

```
// n个人，每个人a_i要顺序坐在pos_i，问最终的序列如何
// 最后一个人一定坐在自己喜欢坐的位置，去掉该位置，倒数第二个人成为最后一个人，所以就是查找空位置的第pos位置

const int N = 4e5 + 10;

#define lc u << 1
#define rc u << 1 | 1
#define mid (l + r) / 2
int sum[N << 2], ans[N];

void push_up(int u) {
    sum[u] = sum[lc] + sum[rc];
}

void build(int u, int l, int r) {
    if(l == r) {
        sum[u] = 1;
        ans[l] = 0;
        return ;
    }
    build(lc, l, mid);
    build(rc, mid + 1, r);
    push_up(u);
}

void modify(int u, int l, int r, int k, int val) {
    if(l == r) {
        ans[l] = val;
        sum[u] = 0;
        return ;
    }
    if(sum[lc] >= k) modify(lc, l, mid, k, val);
    else modify(rc, mid + 1, r, k - sum[lc], val);
    push_up(u);
}

void solve() {
    int n;
    while(~scanf("%d", &n)) {
        vector<pii> p(n + 1);
        for(int i = 1;i <= n; i++) {
            scanf("%d%d",&p[i].fi, &p[i].se);
        }
        build(1, 1, n);
        for(int i = n;i >= 1; i--) {
            modify(1, 1, n, p[i].fi + 1, p[i].se);
        }
        for(int i = 1;i <= n; i++) {
            printf("%d ", ans[i]);
        }
        printf("\n");
    }
}
```

3.12 线段树维护连续区间异或值

```
// 当[l,r]^x时，很不幸异或出来的结果不一定连续，而是分成多个连续区间，所以需要用线段树来构造一个区间异或x之后还是连续区间

// [0, 7]可以分成[0, 3]和[4, 7]，这样区间异或x还是连续区间

// 主要操作:

/*
把低pos位全为0
int ql = (l ^ val) & (((1 << 30) - 1) ^ (1 << pos) - 1);
int qr = ql + (1 << pos) - 1;
把低pos位全为1

```

```
高pos不管
*/

vector<pii> g[N], len;
int l[N], r[N];

void modify(int pos, int l, int r, int L, int R, int val) {
    if(L <= l && r <= R) {
        // 把低pos设置为0
        int ql = (l ^ val) & (((1 << 30) - 1) ^ (1 << pos) - 1);
        int qr = ql + (1 << pos) - 1;
        len.push_back(pii{ql, qr});
        return ;
    }
    int mid = (l + r) / 2;
    if(L <= mid) modify(pos - 1, l, mid, L, R, val);
    if(R  > mid) modify(pos - 1, mid + 1, r, L, R, val);
}

void dfs(int u, int fa, int w) {
    modify(30, 0, (1 << 30) - 1, l[u], r[u], w);
    for(auto e : g[u]) {
        if(e.fi == fa) continue ;
        dfs(e.fi, u, e.se ^ w);
    }
}
```

3.13 线段树维护区间异或

```
const int N = 2e5 + 10;

#define lc u << 1
#define rc u << 1 | 1

struct Tree {
    int sum, tag;
}t[21][N << 2];
int a[N];

void push_up(int id, int u) {
    t[id][u].sum = t[id][lc].sum + t[id][rc].sum;
}

void push_down(int id, int u, int l, int r) {
    if(!t[id][u].tag) return ;
    int m = (l + r) / 2;
    t[id][lc].sum = (m - l + 1) - t[id][lc].sum;
    t[id][rc].sum = (r - m) - t[id][rc].sum;
    t[id][lc].tag ^= 1;
    t[id][rc].tag ^= 1;
    t[id][u].tag = 0;
}

void build(int id, int u, int l, int r) {
    if(l == r) {
        t[id][u].sum = (a[l] >> id) & 1;
        t[id][u].tag = 0;
        return ;
```

```cpp
    }
    int m = (l + r) / 2;
    build(id, lc, l, m);
    build(id, rc, m + 1, r);
    push_up(id, u);
}

void modify(int id, int u, int l, int r, int ql, int qr) {
    if(ql <= l && r <= qr) {
        t[id][u].sum = (r - l + 1) - t[id][u].sum;
        t[id][u].tag ^= 1;
        return ;
    }
    push_down(id, u, l, r);
    int m = (l + r) / 2;
    if(ql <= m) modify(id, lc, l, m, ql, qr);
    if(qr  > m) modify(id, rc, m + 1, r, ql, qr);
    push_up(id, u);
}

int query(int id, int u, int l, int r, int ql, int qr) {
    if(ql <= l && r <= qr) return t[id][u].sum;
    push_down(id, u, l, r);
    int ans = 0;
    int m = (l + r) / 2;
    if(ql <= m) ans += query(id, lc, l, m, ql, qr);
    if(qr  > m) ans += query(id, rc, m + 1, r, ql, qr);
    return ans;
}

void solve() {
    int n, m; cin >> n >> m;
    for(int i = 1;i <= n; i++) cin >> a[i];
    for(int i = 0;i <= 20; i++) {
        build(i, 1, 1, n);
    }
    while(m--) {
        int opt; cin >> opt;
        if(opt == 1) {
            int l, r; cin >> l >> r;
            ll ans = 0;
            for(int i = 0;i <= 20; i++) {
                ans += query(i, 1, 1, n, l, r) * (1ll << i);
            }
            cout << ans << endl;
        }
        else {
            int l, r, k; cin >> l >> r >> k;
            for(int i = 0;i <= 20; i++) {
                if((k >> i) & 1) modify(i, 1, 1, n, l, r);
            }
        }
    }
}
```

3.14 LazySegmentTree

```cpp
struct Info {

};

struct Tag {

};

Info operator+(const Info& a, const Info& b) {

}

void apply(Info &a, const Tag &b) {

}

void apply(Tag &a, const Tag &b) {

}

template<class Info, class Tag,
    class Merge = std::plus<Info>>
struct LazySegmentTree {
    const int n;
    const Merge merge;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree(int n) : n(n), merge(Merge()), info(4 * n + 10), tag(4 * n + 10) {}
    LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = merge(info[2 * p], info[2 * p + 1]);
    }
    void apply(int p, const Tag &v) {
        ::apply(info[p], v);
        ::apply(tag[p], v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = info[p] + v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
};
```

3.15 SparseTable

```
// 倍增思想加DP优化

#include <bits/stdc++.h>
using namespace std;

const int N = 1e5 + 10;

int a[N];


template <typename T, class F = std::function<T(const T&, const T&)>>
class SparseTable {
```

```cpp
public:
    int n;
    std::vector<std::vector<T>> mat;
    F func;

    SparseTable(const std::vector<T>& a, const F& f) : func(f) {
        n = static_cast<int>(a.size());
        int max_log = 32 - __builtin_clz(n);
        mat.resize(max_log);
        mat[0] = a;
        for (int j = 1; j < max_log; j++) {
            mat[j].resize(n - (1 << j) + 1);
            for (int i = 0; i <= n - (1 << j); i++) {
                mat[j][i] = func(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
            }
        }
    }

    T get(int from, int to) const {
        assert(0 <= from && from <= to && to <= n - 1);
        int lg = 32 - __builtin_clz(to - from + 1) - 1;
        return func(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
    }
};

// 二维ST表
int f[105][105][105];

void Two_ST() {
    int n, m;
    cin >> n >> m;
    for(int i = 1;i <= n; i++)
        for(int j = 1;j <= m; j++){
            cin >> a[i];
            f[i][j][0] = a[i];
        }

// 大矩阵分成四个小矩阵求最值

    for(int k = 1;k < log2(n); k++) {
        for(int i = 1;i <= n; i++) {
            for(int j = 1;j <= m; j++) {
                if((i + (1 << (k - 1)) <= n) && (j + (1 << (k - 1)) <= m))
                    f[i][j][k] = max(max(f[i][j + (1 << (k - 1))][k - 1], f[i + (1 << (k - 1))][j][k - 1]), max(f[i][j][k - 1], f[i + (1 << (k - 1))][j + (1 << (k - 1))][k - 1]));
            }
    }

    int T;
    cin >> T;
    while(T--) {
        int l, r, k;
        cin >> l >> r >> k;
        int len = log2(k);
        int s = max(max(f[l][r][len], f[l + k - (1 << len)][r + k - (1 << len)][len]), max(f[l + k - (1 << len)][r][len], f[l][r + k - (1 << len)][len]));
    }

}
```

3.16 CartesianTree

```
// 笛卡尔树是一种由数列构造的特殊二叉搜索树，每个节点都有两个键值，first为下标，second为权值
// 笛卡尔树满足两个性质，在下标递增的情况下就是一个大/小根堆

// 笛卡尔树，静态建树，区间最值跳转
struct CartesianTree {
    int rt; // 根节点
    pii ch[N]; // 左右儿子
    int st[N]; // 单调栈

    void build(int n, int p[]) {
        rt = 0;
        int t = 0;
        for (int i = 1; i <= n; i++) {
            ch[i] = {0, 0};
            // 决定了大于还是小于
            while (t && p[st[t]] > p[i]) --t;
            if (t) {
                // 上一个点的右儿子作为自己的左儿子
                // 成为上一个点的右儿子
                ch[i].first = ch[st[t]].second;
                ch[st[t]].second = i;
            } else { // 自己作为根节点
                ch[i].first = rt;
                rt = i;
            }
            st[++t] = i;
        }
    }
} dika;
```

3.17 DancingLinks

```
// Dancing Links
struct DLX {
    int n, m, size;
    int U[MaxNode], D[MaxNode], L[MaxNode], R[MaxNode], Row[MaxNode], Col[MaxNode];
    int H[MaxN], S[MaxM];
    int ansd, ans[MaxN];

    void init(int _n, int _m) {
        n = _n;
        m = _m;
        for (int i = 0; i <= m; i++) {
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i - 1;
            R[i] = i + 1;
        }
        R[m] = 0;
        L[0] = m;
        size = m;
        for (int i = 0; i <= n; i++) {
            H[i] = -1;
        }
    }

    void Link(int r, int c) {
        ++S[Col[++size] = c];
        Row[size] = r;
        D[size] = D[c];
        U[D[c]] = size;
        U[size] = c;
        D[c] = size;
        if (H[r] < 0) {
            H[r] = L[size] = R[size] = size;
        } else {
            R[size] = R[H[r]];
            L[R[H[r]]] = size;
            L[size] = H[r];
            R[H[r]] = size;
        }
    }

    void remove(int c) {
        L[R[c]] = L[c];
        R[L[c]] = R[c];
        for (int i = D[c]; i != c; i = D[i]) {
            for (int j = R[i]; j != i; j = R[j]) {
                U[D[j]] = U[j];
                D[U[j]] = D[j];
                --S[Col[j]];
            }
        }
    };

    void resume(int c) {
        for (int i = U[c]; i != c; i = U[i])
            for (int j = L[i]; j != i; j = L[j])
                ++S[Col[U[D[j]] = D[U[j]] = j]];
        L[R[c]] = R[L[c]] = c;
    }

    bool Dance(int d) {
        if (R[0] == 0) {
            for (int i = 0; i < d; i++) {
                printf("%d%c", ans[i], " \n"[i == d - 1]);
            }
            return true;
        }
        int c = R[0];
        for (int i = R[0]; i != 0; i = R[i]) if (S[i] < S[c]) c = i;
        remove(c);
        for (int i = D[c]; i != c; i = D[i]) {
            ans[d] = Row[i];
            for (int j = R[i]; j != i; j = R[j])remove(Col[j]);
            if (Dance(d + 1))return true;
            for (int j = L[i]; j != i; j = L[j])resume(Col[j]);
        }
        resume(c);
        return false;
    }
};
```

3.18 ChthollyTree

```cpp
// 要先Split右端点(r+1)，在Split左端点(l)

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

struct node {
    int l, r;
    mutable ll v;
    node (int L, int R = -1, ll V = 0) : l(L), r(R), v(V) {}
    bool operator < (const node &rhs) const {
        return l < rhs.l;
    }
};

set<node> s;

auto Split(int pos) {
    auto it = s.lower_bound(node(pos));
    if(it != s.end() && it -> l == pos) return it;
    --it;
    int L = it -> l, R  = it -> r;
    ll V = it -> v;
    s.erase(it);
    s.insert(node(L, pos - 1, V));
    return s.insert(node(pos, R, V)).first;
}

void assign_val(int l, int r, ll val) { // 推平操作
    auto itr = Split(r + 1);
    auto itl = Split(l);
    s.erase(itl, itr);
    s.insert(node(l, r, val));
}

void add(int l, int r, ll val) { // 区间加
    auto itr = Split(r + 1);
    auto itl = Split(l);
    for( ;itl != itr; ++itl) {
        itl -> v += val;
    }
}

ll kth(int l, int r, int k) { // 区间第k小
    vector<pair<ll, int> > v;
    auto itr = Split(r + 1);
    auto itl = Split(l);
    for( ;itl != itr; ++itl) {
        v.push_back(pair<ll, int>{itl -> v, itl -> r - itl -> l + 1});
    }
    sort(v.begin(), v.end());
    for(auto it = v.begin();it != v.end(); ++it) {
        k -= it -> second;
        if(k <= 0) return it -> first;
    }
}

ll quick_pow(ll a, ll b, ll p) ;

ll qpow(int l, int r, int ex, int p) {
    auto itr = Split(r + 1);
    auto itl = Split(l);
    ll ans = 0;
    for( ;itl != itr; ++itl)
        ans = (ans + ll(itl -> r - itl -> l + 1) * quick_pow(itl -> v, ll(ex), ll(p)) % ll(p)) % ll(p);
    return ans % ll(p);
}
int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; ++i){
        ll x;
        cin >> x;
        s.insert(node(i,i,x));
    }
    s.insert(node(n + 1, n + 1, 0));
    while(m--) {
        int opt, l, r, x;
        cin >> opt >> l >> r >> x;
        if(opt == 1) add(l, r, x);
        else if(opt == 2) assign_val(l, r, x);
        else if(opt == 3) cout << kth(l, r, x)  << endl;
        else {
            int y;
            cin >> y;
            cout << qpow(l, r, x, y) << endl;
        }
    }
}
```

3.19 monotonousQueue

```
template <typename T>
struct monotonousQueue {
    std::vector<T> a;
    monotonousQueue(const std::vector<T>& init) : a(init) {}

    std::vector<T> Max(int k) {
        int n = (int) a.size();
        int head = 0, tail = -1;
        std::queue<int> que(n);
        std::vector<T> ans;
        for (int i = 0; i < n; i++) {
            while (head <= tail && a[que[tail]] <= a[i]) tail--;
            que[++tail] = i;
            while (que[head] + k <= i) head++;
            if (i >= k - 1) ans.push_back(a[que[head]]);
        }
        return ans;
    }

    std::vector<T> Min(int k) {
        int n = (int) a.size();
        int head = 0, tail = -1;
        std::queue<int> que(n);
        std::vector<T> ans;
        for (int i = 0; i < n; i++) {
            while (head <= tail && a[que[tail]] >= a[i]) tail--;
            que[++tail] = i;
            while (que[head] + k <= i) head++;
            if (i >= k - 1) ans.push_back(a[que[head]]);
        }
        return ans;
    }
};
```

3.20 monotonousStack

```
#include <bits/stdc++.h>
using namespace std;

struct Monotone_stack {
    static const int N = 1e5 + 10;
    int a[N];
    stack<int> s;
    int n;

    void read() {
        cin >> n;
        for(int i = 1;i <= n; i++) cin >> a[i];
    }

    void Monotone_min() {
        for(int i = 1;i <= n; i++) {
            if(s.empty() || s.top() >= a[i])
                s.push(a[i]);
            else {
                while(!s.empty() && s.top() < a[i]) {
                    cout << s.top() << endl;
                    s.pop();
                }
                s.push(a[i]);
            }
        }
        while(!s.empty()) {
            cout << s.top() << endl;
            s.pop();
        }
    }

    void Monotone_max() {
        for(int i = 1;i <= n; i++) {
            if(s.empty() || s.top() <= a[i])
                s.push(a[i]);
            else {
                while(!s.empty() && s.top() > a[i]) {
                    cout << s.top() << endl;
                    s.pop();
                }
                s.push(a[i]);
            }
        }
        while(!s.empty()) {
            cout << s.top() << endl;
            s.pop();
        }
    }

}Worker;
```

3.21 difference

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

using namespace std;

template <typename T>
struct difference {
    int n;
    std::vector<T> d;

    difference(int _n) : n(_n), d(_n + 1) {}
    difference(std::vector<T>& init) : difference(init.size()) {
        d[0] = init[0];
        for (int i = 1; i < n; i++) {
            d[i] = init[i] - init[i - 1];
        }
    }

    void modify(int l, int r, T v) {
        assert(0 <= l && l <= n - 1 && 0 <= r && r <= n - 1);
        d[l] += v;
        d[r + 1] -= v;
    }

    void solve() {
        for (int i = 1; i < n; i++) {
            d[i] += d[i - 1];
        }
    }
};

template <typename T>
struct difference2d {
    int n, m;
    std::vector<std::vector<T>> d;

    difference2d(int _n, int _m) : n(_n), m(_m), d(_n + 1, std::vector<T>(_m + 1)) {}
    difference2d(std::vector<std::vector<T>>& init) : difference2d(init.size(), init[0].size()) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (i == 0 || j == 0) d[i][j] = init[i][j];
                else d[i][j] = init[i][j] - init[i - 1][j] - init[i][j - 1] + init[i - 1][j - 1];
            }
        }
    }
```

```cpp

    void modify(int x1, int y1, int x2, int y2, T v) {
        assert(0 <= x1 <= n - 1 && 0 <= y1 <= n - 1 && 0 <= x2 <= n - 1 && 0 <= y2 <= n - 1);
        d[x1][y1] += v;
        d[x1][y2 + 1] -= v;
        d[x2 + 1][y1] -= v;
        d[x2 + 1][y2 + 1] += v;
    }
};
```

3.22 trie

```cpp
class Trie {
private :
    Trie* next[26] = {nullptr};
    int val;
public :
    Trie() {}

    void insert(std::string& s) {
        Trie* root = this;
        for (char &c : s) {
            if (root -> next[c] == nullptr) {
                root -> next[c] = new Trie();
            }
            root = root -> next[c];
        }
        root -> val ++;
    }

    void del(std::string& s) {
        Trie* root = this;
        for (char &c : s) {
            root = root -> next[c];
        }
        root -> val --;
    }

    int search(std::string& s) {
        int ans = 0;
        Trie* root = this;
        for (char& c : s) {
            if (!root -> next[c]) break ;
            root = root -> next[c];
            ans += root -> val;
        }
        return ans;
    }
};
```

3.23 HashTable

```cpp
template<typename T>
class HashTable{
private :
    const int maxn;
    std::vector<std::vector<T>> key, val;

public :
    HashTable(int n) : maxn(n), key(n), val(n) {}

    int hash(int x){
        return (((long long)x * (x + 1)) ^ x) % maxn;
    }
    void insert(int x){
        int u = hash(x);
        for(int v = 0; v < (int)key[u].size(); ++v)
            if(key[u][v] == x){
                ++val[u][v];
                return;
            }
        key[u].push_back(x), val[u].push_back(1);
    }
    T query(int x){
        int u = hash(x);
        for(int v = 0; v < (int)key[u].size(); ++v)
            if(key[u][v] == x)
                return val[u][v];
        return 0;
    }
};
```

3.24 2-4 维前缀和

```
// 统计(a,b)到(c,d)这个矩阵中的所有0子矩阵

const int N = 50 + 10;
int sum[N][N];
int Q[N][N][N][N];

void solve() {
    int n, m, q; cin >> n >> m >> q;
    for(int i = 1;i <= n; i++) {
        string s; cin >> s;
        for(int j = 1;j <= m; j++) {
            sum[i][j] = (s[j - 1] - '0') + sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1];
        }
    }

    for(int a = 1;a <= n; a++) {
        for(int b = 1;b <= m; b++) {
            for(int c = a;c <= n; c++) {
                for(int d = b;d <= m; d++) {
                    if(sum[c][d] - sum[a - 1][d] - sum[c][b - 1] + sum[a - 1][b - 1] == 0) {
                        Q[a][b][c][d]++;
                    }
                }
            }
        }
    }

    for(int a = n;a >= 1; a--) {
        for(int b = m;b >= 1; b--) {
            for(int c = 1;c <= n; c++) {
                for(int d = 1;d <= m; d++) {
                    Q[a][b][c][d] += Q[a + 1][b][c][d];
                }
            }
        }
    }

    for(int a = n;a >= 1; a--) {
        for(int b = m;b >= 1; b--) {
            for(int c = 1;c <= n; c++) {
                for(int d = 1;d <= m; d++) {
                    Q[a][b][c][d] += Q[a][b + 1][c][d];
                }
            }
        }
    }

    for(int a = n;a >= 1; a--) {
        for(int b = m;b >= 1; b--) {
            for(int c = 1;c <= n; c++) {
                for(int d = 1;d <= m; d++) {
                    Q[a][b][c][d] += Q[a][b][c - 1][d];
                }
            }
        }
    }

    for(int a = n;a >= 1; a--) {
        for(int b = m;b >= 1; b--) {
            for(int c = 1;c <= n; c++) {
                for(int d = 1;d <= m; d++) {
                    Q[a][b][c][d] += Q[a][b][c][d - 1];
                }
            }
        }
    }

    while(q--) {
        int a, b, c, d; cin >> a >> b >> c >> d;
        cout << Q[a][b][c][d] << endl;
    }
}
```

3.25 simpleDSU

```
class DSU {
private :
    std::vector<int> f, siz;
    std::vector<int> dep;
public :
    DSU(int n) : f(n), dep(n), siz(n, 1) { iota(f.begin(), f.end(), 0); }
    int find(int x) {
        while(x != f[x]) x = f[x] = f[f[x]];
        return x;
    }
    bool same(int x, int y) { return find(x) == find(y); }
    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if(x == y) return false;
        if (dep[x] > dep[y]) std::swap(x, y);
        siz[y] += siz[x];
        f[x] = y;
        dep[y] = std::max(dep[y], dep[x] + 1);
        return true;
    }
    int size(int x) { return siz[find(x)]; }
};
```

3.26 valueDSU

```
class DSU {
private :
    std::vector<int> f, siz, val;

public :
    DSU(int n) : f(n), val(n), siz(n, 1) { iota(f.begin(), f.end(), 0); }

    int find(int x) {
        if (x != f[x]) {
            int fa = f[x];
            f[x] = find(f[x]);
            val[x] += val[fa];
        }
        return f[x];
    }
    bool same(int x, int y) { return find(x) == find(y); }
    bool merge(int x, int y, int v) {
        int nx = find(x);
        int ny = find(y);
        if(nx == ny) return false;
        siz[nx] += siz[ny];
        f[ny] = nx;
        val[ny] = val[x] + v - val[y];
        return true;
    }
    int size(int x) { return siz[find(x)]; }
};
```

3.27 modifyDSU

```
struct node {
    int x, y, z;
};

struct UnionFind {
private:
    int rk[N], pre[N], siz[N], totNode;//N为最大点数
    stack<node> st;//node记录上次修改的内容
public:
    void init(int tot) {
        totNode = tot;
        for (int i = 1; i <= totNode; i++)
```

```cpp
            pre[i] = i, siz[i] = rk[i] = 1;
    }
    int find(int x) { while (x ^ pre[x]) x = pre[x]; return x; }
    void merge(int x, int y) {//按秩合并
        x = find(x), y = find(y);
        if (x == y) return;
        if (rk[x] < rk[y]) swap(x, y);
        st.push(node{ y, rk[x], siz[y] });
        pre[y] = x, rk[x] += rk[x] == rk[y], siz[x] += siz[y];
    }
    int start() { return st.size(); }
    void end(int last) {//撤回merge操作
        while (st.size() > last) {
            node tp = st.top();
            rk[pre[tp.x]] -= tp.y, siz[pre[tp.x]] -= tp.z;
            pre[tp.x] = tp.x;
            st.pop();
        }
    }
    bool judge() { return siz[find(1)] == totNode; }
};
```

3.28 varietyDSU

```cpp
class DSU {
private :
    std::vector<int> f, siz;

public :
    DSU(int n) : f(n), siz(n, 1) { iota(f.begin(), f.end(), 0); }
    int find(int x) {
        while(x != f[x]) x = f[x] = f[f[x]];
        return x;
    }
    bool same(int x, int y) { return find(x) == find(y); }
    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if(x == y) return false;
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }
    int size(int x) { return siz[find(x)]; }
};

int main() {
    int n, q;
    cin >> n >> q;
    for(int i = 1;i <= 2 * n; i++) f[i] = i;// **

    while(q--) {
        int flag, x, y;
        cin >> flag >> x >> y;
        if(flag) { // 敌人
            merge(x + n, y);
            merge(y + n, x);
        }
        else {
            merge(x, y); // 同伴
        }
    }
    int ans = 0;
    for(int i = 1;i <= n; i++) {
        if(f[i] == i) ans ++;
    }
    cout << ans << endl;
}
```

3.29 KD 求矩阵权值和

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 200005;
const double alpha = 0.75;
//查包含在x1,y1,x2,y2为左下角和右上角的矩形里面权值之和
//K-D Tree 二维划分树
int n, ans, rt, WD, tot, top, rub[MAX];

struct node {
    int x[2], w;
} p[MAX];

struct K_D_tree {
    int ls, rs, siz, mn[2], mx[2], sum;
    //mn[0], mx[0] -> x的取值范围
    //mn[1], mx[1] -> y的取值范围
    node tmp;
} t[MAX];

int operator < (const node &a, const node &b) { return a.x[WD] < b.x[WD]; }

int newnode() {
    if (top) return rub[top--];
    else return ++tot;
}

void push_up(int u) {
    for (int i = 0; i <= 1; i++) {//更新x，y的取值范围
        t[u].mn[i] = t[u].mx[i] = t[u].tmp.x[i];
        if (t[u].ls) {//左子树的最大最小值
            t[u].mn[i] = min(t[u].mn[i], t[t[u].ls].mn[i]);
            t[u].mx[i] = max(t[u].mx[i], t[t[u].ls].mx[i]);
        }
        if (t[u].rs) {//右子树的最大最小值
            t[u].mn[i] = min(t[u].mn[i], t[t[u].rs].mn[i]);
            t[u].mx[i] = max(t[u].mx[i], t[t[u].rs].mx[i]);
        }
    }
    t[u].sum = t[t[u].ls].sum + t[t[u].rs].sum + t[u].tmp.w;
    t[u].siz = t[t[u].ls].siz + t[t[u].rs].siz + 1;
}

int build(int l, int r, int wd) {
    if (l > r) return 0;
    int u = newnode();
    int m = (l + r) >> 1;
    WD = wd; nth_element(p + l, p + m, p + r + 1);
    t[u].tmp = p[m];
    t[u].ls = build(l, m - 1, wd ^ 1);
    t[u].rs = build(m + 1, r, wd ^ 1);
    push_up(u);
    return u;
}

void pia(int u, int num) {//拍扁回炉重做
    if (t[u].ls) pia(t[u].ls, num);
    p[t[t[u].ls].siz + num + 1] = t[u].tmp, rub[++top] = u;
    if (t[u].rs) pia(t[u].rs, t[t[u].ls].siz + num + 1);
}

void check(int &u, int wd) {//检查是否平衡，不平衡则需要重建
    if (t[u].siz * alpha < t[t[u].ls].siz || t[u].siz * alpha < t[t[u].rs].siz) pia(u, 0), u = build(1, t[u].siz, wd);
}

void insert(int &u, node tp, int wd) {//插入点
    if (!u) {
        u = newnode();
        t[u].ls = t[u].rs = 0, t[u].tmp = tp;
        push_up(u);
        return;
    }
    if (tp.x[wd] <= t[u].tmp.x[wd]) insert(t[u].ls, tp, wd ^ 1);
    else insert(t[u].rs, tp, wd ^ 1);
    push_up(u);
    check(u, wd);
}

bool in(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) {//完全被包含
    return (x1 <= X1 && X2 <= x2 && y1 <= Y1 && Y2 <= y2);
}

bool out(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) {//完全无交集
    return (x1 > X2 || x2 < X1 || y1 > Y2 || y2 < Y1);
}

int query(int u, int x1, int y1, int x2, int y2) {
    if (!u) return 0;
    int res = 0;
    if (in(x1, y1, x2, y2, t[u].mn[0], t[u].mn[1], t[u].mx[0], t[u].mx[1])) return t[u].sum;
    if (out(x1, y1, x2, y2, t[u].mn[0], t[u].mn[1], t[u].mx[0], t[u].mx[1])) return 0;
    if (in(x1, y1, x2, y2, t[u].tmp.x[0], t[u].tmp.x[1], t[u].tmp.x[0], t[u].tmp.x[1])) res += t[u].tmp.w;
    res += query(t[u].ls, x1, y1, x2, y2) + query(t[u].rs, x1, y1, x2, y2);
    return res;
}

void init() {
    ans = rt = top = tot = 0;
}
```

```
void solve() {
    scanf("%d", &n);
    while(1) {
        int opt; scanf("%d",&opt);
        if(opt == 1) {
            int x, y, w; scanf("%d%d%d",&x,&y,&w);
            insert(rt, node{x ^ ans, y ^ ans, w ^ ans}, 0);
        }
        else if(opt == 2) {
            int x1, y1, x2, y2; scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
            ans = query(rt, x1 ^ ans, y1 ^ ans, x2 ^ ans, y2 ^ ans);
            printf("%d\n",ans);
        }
        else break;
    }
}
```

3.30 KD 求最近点对距离

```
// 二维平面里最近点对距离
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const int N = 1e5 + 10;
struct node {
    ll x[2];
}a[N], b[N];

int now, n;
ll ans;
map<pair<ll, ll>, int> mp;

bool cmp(node a, node b) { return a.x[now] < b.x[now]; }

ll sqr(int x) { return (ll)x * x; }
ll dis(node a, node b) { return sqr(a.x[0] - b.x[0]) + sqr(a.x[1] - b.x[1]); }

void build(int l, int r, int d) {
    if(l >= r) return ;
    int m = (l + r) >> 1;
    now = d;
    nth_element(a + l, a + m, a + r, cmp);
    build(l, m, d ^ 1);
    build(m + 1, r, d ^ 1);
}

void query(int l, int r, int d, node p) {
    if(l >= r) return ;
    int m = (l + r) >> 1;
    int ql = l, qr = m;
    ll res = dis(a[m], p);
    if(ans == 0 || res && ans > res) ans = res;
    if(p.x[d] > a[m].x[d]) ql = m + 1, qr = r;
    query(ql, qr, d ^ 1, p);
    if(ans > sqr(a[m].x[d] - p.x[d]))
        query(l + m - ql + 1, m + r - qr, d ^ 1, p);
}
```

```

void solve() {
    scanf("%d",&n);
    ll sum = 5e18;
    for(int i = 0;i < n; i++) {
        scanf("%lld %lld",&a[i].x[0],&a[i].x[1]);
        if(mp[{a[i].x[0], a[i].x[1]}]) sum = 0;
        else mp[{a[i].x[0], a[i].x[1]}]++;
        b[i] = a[i];
    }
    build(0, n, 0);
    for(int i = 0;i < n; i++) {
        ans = 0;
        query(0, n, 0, b[i]);
        sum = min(ans, sum);
    }
    printf("%.4lf\n",sqrt(1.0 * sum));
}
```

3.31 CDQ 分治

```
// 原问题：“任意两个元素之间的贡献”

// 降维成的子问题：“左段元素对右段每一个元素的贡献”

// 分而治之，最基础的应用为归并排序
// 将一个区间分成两个区间[l,mid]和[mid+1,r]，然后排序使其有序
// 注意事项：
// 1、离散化
// 2、如果有多个点的参量完全相同，由于顺着添加，查出来的最后一个答案才是正确的
// 对于y排序，我们可以直接对两段区间Sort，然而归并排序本身就是分治，我们可以在CDQ的过程中进行归并排序，要比Sort少一个log

// 千万不要忘记离散化！！！！！！！！！！！！！！！！！！
```

3.32 cdq 处理逆序数

```
const int N = 1e5 + 10;
int ans[N], cnt[N];

struct star {
    int x, y, id;
}a[N], tmp[N];

bool cmp(star a, star b) {
    if(a.y == b.y) return a.x < b.x;
    return a.y < b.y;
}

void cdq(int l, int r) {
    if(l == r) return ;
    int m = (l + r) / 2;
    cdq(l, m);
    cdq(m + 1, r);
    int p = l, q = m + 1;
    for(int i = l;i <= r; i++) {
        if((p <= m && a[p].x <= a[q].x) || q > r) {
                tmp[i] = a[p++];
            }
            else {
                ans[a[q].id] += p - l;
                tmp[i] = a[q++];
            }
        }
        for(int i = l;i <= r; i++) a[i] = tmp[i];
}

void solve() {
    int n; cin >> n;
    for(int i = 1;i <= n; i++) cin >> a[i].x >> a[i].y, a[i].id = i;
    sort(a + 1, a + n + 1, cmp);
    cdq(1, n);
    for(int i = 1;i <= n; i++) cnt[ans[i]]++;
    for(int i = 0;i < n; i++) cout << cnt[i] << endl;
}
```

3.33 cdq 处理二维偏序

```

const int N = 1e5 + 10;
int ans[N], cnt[N];

struct star {
    int x, y, id;
}a[N], tmp[N];

bool cmp(star a, star b) {
    if(a.x == b.x) return a.y < b.y;
    return a.x < b.x;
}

void cdq(int l, int r) {
    if(l == r) return ;
    int m = (l + r) / 2;
    cdq(l, m);
    cdq(m + 1, r);
    int p = l, q = m + 1;
    for(int i = l;i <= r; i++) {
        if((p <= m && a[p].x <= a[q].x) || q > r) {
            tmp[i] = a[p++];
        }
        else {
            ans[a[q].id] += i - l;
            tmp[i] = a[q++];
        }
    }
    for(int i = l;i <= r; i++) a[i] = tmp[i];
}

void solve() {
    int n; cin >> n;
    for(int i = 1;i <= n; i++) cin >> a[i].x >> a[i].y;
    sort(a + 1, a + n + 1, cmp);
    cdq(1, n);
    for(int i = 1;i <= n; i++) cnt[ans[i]]++;
    for(int i = 0;i < n; i++) cout << cnt[i] << endl;
}
```

3.34 cdq 套 cdq 处理三维偏序

```
// 一维排序、二维cdq、三维树状数组

const int N = 1e5 + 10;
struct node {
    int x, y, z;
    int id;
    int tag;
    bool operator < (const node &a)const{
        if(x != a.x) return x < a.x;
        if(y != a.y) return y < a.y;
        return z < a.z;
    }
    bool operator == (const node &a)const{
        return x == a.x && y == a.y && z == a.z;
    }
}a[N], b[N], tmp[N];

int ans[N];
int n;

void cdq2(int l, int r) {
    if(l == r) return ;
    int mid = (l + r) / 2;
    cdq2(l, mid); cdq2(mid + 1, r);
    int p = l, q = mid + 1, cnt = 0;
    for(int i = l;i <= r; i++) {
        if(q > r || (p <= mid && b[p].z <= b[q].z)) {
            if(b[p].tag == 0) cnt++;
            tmp[i] = b[p++];
        }
        else {
            if(b[q].tag == 1) ans[b[q].id] += cnt;
            tmp[i] = b[q++];
        }
    }
    for(int i = l;i <= r; i++) b[i] = tmp[i];
}

void cdq1(int l, int r) {
    if(l == r) return ;
    int mid = (l + r) / 2;
    cdq1(l, mid); cdq1(mid + 1, r);
    int p = l, q = mid + 1;
    /* 因为是计算左端元素对右端元素的影响，所以需要打个标记tag来记录他是左端还是右端元素 */
    for(int i = l;i <= r; i++) {
        if(q > r || (p <= mid && a[p].y <= a[q].y)) {
            a[p].tag = 0;
            b[i] = a[p++];
        }
        else {
            a[q].tag = 1;
            b[i] = a[q++];
        }
    }
    for(int i = l;i <= r; i++) a[i] = b[i];
    cdq2(l, r);
}

void solve() {
    n = read();
    for(int i = 1;i <= n; i++) {
        a[i].x = read(), a[i].y = read(), a[i].z = read();
        a[i].id = i;
    }
    sort(a + 1, a + n + 1);
    for(int i = n - 1;i >= 1; i--) {
        if(a[i + 1] == a[i]) ans[a[i].id] = ans[a[i + 1].id] + 1;
    }
    cdq1(1, n);
    for(int i = 1;i <= n; i++) cout << ans[i] << endl;
}
```

3.35 cdq 套树状数组处理三维偏序

```
// 一维排序、二维cdq、三维树状数组

const int N = 1e5 + 10;
struct node {
    int x, y, z;
    int id;
    bool operator < (const node &a)const{
        if(x != a.x) return x < a.x;
        if(y != a.y) return y < a.y;
        return z < a.z;
    }
    bool operator == (const node &a)const{
        return x == a.x && y == a.y && z == a.z;
    }
}a[N], b[N];
int n;


int ans[N];

struct BIT {
#define lowbit(x) (x & (-x))
    int n;
    int t[N];

    void init(int _n) {
        mem(t, 0);
        n = _n;
    }

    void update(int x, int val) {
        while (x <= n) {
            t[x] += val;
            x += lowbit(x);
        }
    }

    int query(int x) {
        int ans = 0;
        while (x) {
            ans += t[x];
            x -= lowbit(x);
        }
        return ans;
    }
}bit;

void cdq(int l, int r) {
    if(l == r) return ;
    int mid = (l + r) / 2;
    cdq(l, mid);
    cdq(mid + 1, r);
    int p = l, q = mid + 1;
    for(int i = l;i <= r; i++) {
        if(q > r || (p <= mid && a[p].y <= a[q].y)) {
            bit.update(a[p].z, 1);
            b[i] = a[p++];
        }
        else {
            ans[a[q].id] += bit.query(a[q].z);
            b[i] = a[q++];
        }
    }
    for(int i = l;i <= mid; i++) bit.update(a[i].z, -1);
    for(int i = l;i <= r; i++) a[i] = b[i];
}

void solve() {
    n = read();
    int mx = 0;
    for(int i = 1;i <= n; i++) {
        a[i].x = read(), a[i].y = read(), a[i].z = read();
        a[i].id = i;
        mx = max(mx, a[i].z);
    }
    bit.init(mx);
    sort(a + 1, a + n + 1);
    for(int i = n - 1;i >= 1; i--) {
        if(a[i] == a[i + 1]) ans[a[i].id] = ans[a[i + 1].id] + 1;
    }
    cdq(1, n);
    for(int i = 1;i <= n; i++) cout << ans[i] << endl;
}
```

3.36 cdq 维护矩阵内二维数点

```
// 求二维平面上(x1,y1)到(x2,y2)的矩阵中数点

// 利用前缀和思想，把问题划分成[x2,y2] - [x1-1,y2] - [x2,y2-1] + [x1-1,y1-1]

// 所有要建立4个虚点为查询点，而原本实点为修改点

const int N = 3e6 + 10;

struct node {
    int x, y, opt, id;
    // opt为操作类型，1为修改，0为查询
    bool operator < (const node& o) const {
        return x == o.x ? (y == o.y ? opt : y < o.y) : x < o.x;
    }
    // 注意排序顺序，坐标相同时，要使opt放前面，因为要先修改

    bool operator == (const node &o) const {
        return x == o.x && y == o.y;
    }
}a[N], tmp[N];

int ans[N];

void cdq(int l, int r) {
    if(l == r) return ;
    int mid = (l + r) / 2;
    cdq(l, mid); cdq(mid + 1, r);
    int p = l, q = mid + 1, cnt = 0;
    for(int i = l;i <= r; i++) {
        if(q > r || (p <= mid && a[p].y <= a[q].y)) {
            cnt += a[p].opt;
            tmp[i] = a[p++];
        }
        else {
            ans[a[q].id] += cnt;
            tmp[i] = a[q++];
        }
    }
    for(int i = l;i <= r; i++) a[i] = tmp[i];
}

void solve() {
    int n = read(), m = read();
    for(int i = 1;i <= n; i++) {
        a[i].x = read(), a[i].y = read(), a[i].opt = 1;
    }
    int _n = 0;
    for(int i = 1;i <= m; i++) {
        int x1 = read(), y1 = read(), x2 = read(), y2 = read();
        a[++n] = (node){x2, y2, 0, ++_n};
        a[++n] = (node){x2, y1 - 1, 0, ++_n};
        a[++n] = (node){x1 - 1, y2, 0, ++_n};
        a[++n] = (node){x1 - 1, y1 - 1, 0, ++_n};
    }
    sort(a + 1, a + n + 1);
    cdq(1, n);
    for(int i = 1;i + 3 <= _n; i += 4) {
        cout << ans[i] - ans[i + 1] - ans[i + 2] + ans[i + 3] << endl;
    }
}
```

3.37 advanced01trie

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e7 + 10;
```

```

int t[N][2];
int cnt, root[N], sz[N][2];
int a[N];

void insert(int pre, int &now, int i, int x) {
    if(i < 0) return ;
    now = ++cnt;
    int d = x >> i & 1;
    t[now][d ^ 1] = t[pre][d ^ 1];
    sz[now][d ^ 1] = sz[pre][d ^ 1]; sz[now][d] = sz[pre][d] + 1;
    insert(t[pre][d], t[now][d], i - 1, x);
}

int query(int l, int r, int i, int x) {
    if(i < 0) return 0;
    int d = x >> i & 1;
    int tmp = sz[r][d ^ 1] - sz[l][d ^ 1];
    if(tmp > 0) return query(t[l][d ^ 1], t[r][d ^ 1], i - 1, x) + (1 << i);
    else return query(t[l][d], t[r][d], i - 1, x);
}

int main() {
    int n, m;
    cin >> n >> m;
    for(int i = 1;i <= n; i++) {
        int x;
        cin >> x;
        insert(root[i - 1], root[i], 30, x);
    }
    while(m--) {
        int l, r, x;
        cin >> l >> r >> x;
        cout << query(root[l - 1], root[r], 30, x) << endl;
    }
}
```

3.38 advancedArray

```

#include <bits/stdc++.h>
using namespace std;

const int N = 1e5 + 10;

struct Node {
    int l, r, val;
}hjt[N * 40];
int cnt, root[N];
int a[N];

inline void build(int &now, int l, int r) {
    now = ++cnt;
    if(l == r) {
        hjt[now].val = a[l];
        return ;
    }
    int m = (l + r) >> 1;
    build(hjt[now].l, l, m);
    build(hjt[now].r, m + 1, r);
}

inline void modify(int ver, int &now, int l, int r, int pos, int value) {
    hjt[now = ++cnt] = hjt[ver];
    if(l == r) {
        hjt[now].val = value;
        return ;
    }
    int m = (l + r) >> 1;
    if(pos <= m) modify(hjt[ver].l, hjt[now].l, l, m, pos, value);
    else modify(hjt[ver].r, hjt[now].r, m + 1, r, pos, value);
}

inline int query(int now, int l, int r, int pos) {
    if(l == r) return hjt[now].val;
    int m = (l + r) >> 1;
    if(pos <= m) return query(hjt[now].l, l, m, pos);
    else return query(hjt[now].r, m + 1, r, pos);
}

int main() {
    int n, m;
    cin >> n >> m;
    for(int i = 1;i <= n; i++) cin >> a[i];
    build(root[0], 1, n);
    for(int i = 1;i <= m; i++) {
        int ver, opt;
        cin >> ver >> opt;
        if(opt == 1) {
            int pos, value;
            cin >> pos >> value;
            modify(root[ver], root[i], 1, n, pos, value);
        }
        else {
            int pos;
            cin >> pos;
            root[i] = root[ver];
            cout << query(root[i], 1, n, pos) << endl;
        }
    }
}
```

3.39 anvancedDSU

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e5 + 10;

struct Node {
    int l, r, val;
}hjt[N * 40 * 2];

int cnt, rootfa[N], rootdep[N], tot;
int n;

```

```
inline void build(int &now, int l, int r) {
    now = ++cnt;
    if(l == r) {
        hjt[now].val = ++tot;
        return ;
    }
    int m = (l + r) >> 1;
    build(hjt[now].l, l, m);
    build(hjt[now].r, m + 1, r);
}

inline void modify(int ver, int &now, int l, int r, int pos, int value) {
    hjt[now = ++cnt] = hjt[ver];
    if(l == r) {
        hjt[now].val = value;
        return ;
    }
    int m = (l + r) >> 1;
    if(pos <= m) modify(hjt[ver].l, hjt[now].l, l, m, pos, value);
    else modify(hjt[ver].r, hjt[now].r, m + 1, r, pos, value);
}

inline int query(int now, int l, int r, int pos) {
    if(l == r) return hjt[now].val;
    int m = (l + r) >> 1;
    if(pos <= m) return query(hjt[now].l, l, m, pos);
    else return query(hjt[now].r, m + 1, r, pos);
}

inline int find(int ver, int x) {
    int fx = query(rootfa[ver], 1, n, x);
    return fx == x ? x : find(ver, fx);
}

inline void merge(int ver, int x, int y) {
    x = find(ver - 1, x);
    y = find(ver - 1, y);
    if(x == y) {
        rootfa[ver] = rootfa[ver - 1];
        rootdep[ver] = rootdep[ver - 1];
    }
    else {
        int depx = query(rootdep[ver - 1], 1, n, x);
        int depy = query(rootdep[ver - 1], 1, n, y);
        if(depx < depy) {
            modify(rootfa[ver - 1], rootfa[ver], 1, n, x, y);
            rootdep[ver] = rootdep[ver - 1];
        }
        else if(depx > depy) {
            modify(rootfa[ver - 1], rootfa[ver], 1, n, y, x);
            rootdep[ver] = rootdep[ver - 1];
        }
        else {
            modify(rootfa[ver - 1], rootfa[ver], 1, n, x, y);
            modify(rootdep[ver - 1], rootdep[ver], 1, n, y, depy + 1);
        }
    }
}

```

```
int main() {
    int m;
    cin >> n >> m;
    build(rootfa[0], 1, n);
    for(int ver = 1;ver <= m; ver++) {
        int opt, x, y;
        cin >> opt;
        if(opt == 1) {
            cin >> x >> y;
            merge(ver, x, y);
        }
        else if(opt == 2) {
            cin >> x;
            rootfa[ver] = rootfa[x];
            rootdep[ver] = rootdep[x];
        }
        else {
            cin >> x >> y;
            rootfa[ver] = rootfa[ver - 1];
            rootdep[ver] = rootdep[ver - 1];
            int u = find(ver, x);
            int v = find(ver, y);
            if(u == v) cout << 1 << endl;
            else cout << 0 << endl;
        }
    }
}
```

3.40 扫描线求面积并

```
// 横向扫描
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 10;

#define lc u << 1
#define rc u << 1 | 1
#define mid (t[u].l + t[u].r) >> 1

int n, cnt;

double v[N];
struct L {
    double x, y1, y2;
    int state;
    bool operator < (L rhs) {return x < rhs.x; }
}line[N << 2];

struct Node {
    int l, r, cover;
    double len;
}t[N << 2];

inline void push_up(int u) {
    if(t[u].cover) t[u].len = v[t[u].r + 1] - v[t[u].l];
    else if(t[u].l == t[u].r) t[u].len = 0;
    else t[u].len = t[lc].len + t[rc].len;
}

void build(int u, int l, int r) {
    t[u].l = l; t[u].r = r;
    if(l == r) {
        t[l].cover = t[l].len = 0;
        return ;
    }
    int m = (l + r) >> 1;
    build(lc, l, m);
    build(rc, m + 1, r);
}

void modify(int u, int ql, int qr, int state) {
    if(ql <= t[u].l && t[u].r <= qr) {
        t[u].cover += state;
        push_up(u);
        return ;
    }
    if(ql <= mid) modify(lc, ql, qr, state);
    if(qr > mid)  modify(rc, ql, qr, state);
    push_up(u);
}

void init() {
    cin >> n;
    for(int i = 1;i <= n; i++) {
        double x1, y1, x2, y2;
        scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
        line[i] = L{x1, y1, y2, 1}; v[i] = y1;
        line[n + i] = L{x2, y1, y2, -1}; v[n + i] = y2;
    }
    n <<= 1;
    sort(line + 1, line + n + 1);
    sort(v + 1, v + n + 1);
    cnt = unique(v + 1, v + n + 1) - (v + 1);
    build(1, 1, cnt);
}

void solve() {
    double ans = 0;
    for(int i = 1;i <= n; i++) {
        int ql = lower_bound(v + 1, v + cnt + 1, line[i].y1) - v;
        int qr = lower_bound(v + 1, v + cnt + 1, line[i].y2) - v - 1;
        modify(1, ql, qr, line[i].state);
        ans += t[1].len * (line[i + 1].x - line[i].x);
    }
    cout << ans << endl;
}

int main() {
    init();
    solve();
}
```

3.41 扫描线求周长并

```
// 纵向扫描
```

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 10;

#define INF 0x3fffff
#define lc u << 1
#define rc u << 1 | 1
#define mid (t[u].l + t[u].r) >> 1

int n;

struct L {
    int y, x1, x2;
    int state;
    bool operator < (L rhs) {return y < rhs.y; }
}line[N << 2];

struct Node {
    int l, r, cover;
    bool ls, rs;
    int num;
    int len;
}t[N << 2];

inline void push_up(int u) {
    if(t[u].cover) {
        t[u].len = t[u].r - t[u].l + 1;
        t[u].ls = t[u].rs = 1;
        t[u].num = 1;
    }
    else if(t[u].l == t[u].r) {
        t[u].ls = t[u].rs = 0;
        t[u].len = t[u].num = 0;
    }
    else {
        t[u].len = t[lc].len + t[rc].len;
        t[u].ls = t[lc].ls; t[u].rs = t[rc].rs;
        t[u].num = t[lc].num + t[rc].num - (t[lc].rs & t[rc].ls);
    }
}

void build(int u, int l, int r) {
    t[u].l = l; t[u].r = r;
    if(l == r) {
        t[u].len = t[u].cover = t[u].ls = t[u].rs = t[u].num = 0;
        return ;
    }
    int m = (l + r) >> 1;
    build(lc, l, m);
    build(rc, m + 1, r);
}

void modify(int u, int ql, int qr, int state) {
    if(ql <= t[u].l && t[u].r <= qr) {
        t[u].cover += state;
        push_up(u);
        return ;
    }
    if(ql <= mid) modify(lc, ql, qr, state);
    if(qr > mid)  modify(rc, ql, qr, state);
    push_up(u);
}

void init() {
    cin >> n;
    int mx = -INF, mn = INF;
    for (int i = 1; i <= n; i++) {
        int x1, x2, y1, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        mx = max(mx, max(x1, x2));
        mn = min(mn, min(x1, x2));
        line[i] = L{y1, x1, x2, 1};
        line[n + i] = L{y2, x1, x2, -1};
    }
    n <<= 1;
    sort(line + 1, line + n + 1);
    build(1, mn, mx);
}

void solve() {
    int ans = 0;
    int last = 0;
    for(int i = 1;i <= n; i++) {
        modify(1, line[i].x1, line[i].x2 - 1, line[i].state);
        ans += abs(t[1].len - last); // 横线
        ans += (line[i + 1].y - line[i].y) * 2 * t[1].num; // 竖线
        last = t[1].len;
    }
    printf("%d\n",ans);
}

int main() {
    init();
    solve();
}
```

3.42 区间第 k 小

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e5 + 10;

vector<int> v;
struct Node {
    int l, r;
    int val;
}hjt[N * 40];
int root[N], cnt;

int get_id(int x) { return lower_bound(v.begin(), v.end(), x) - v.begin() + 1; }

void insert(int pre, int &now, int l, int r, int p) {
    hjt[++cnt] = hjt[pre];
    now = cnt;
    hjt[now].val++;
    if(l == r) return ;
    int m = (l + r) >> 1;
    if(p <= m) insert(hjt[pre].l, hjt[now].l, l, m, p);
    else insert(hjt[pre].r, hjt[now].r, m + 1, r, p);
}

int query(int L, int R, int l, int r, int k) {
    if(l == r) return l;
    int m = (l + r) >> 1;
    int tmp = hjt[hjt[R].l].val - hjt[hjt[L].l].val;
    if(k <= tmp) return query(hjt[L].l, hjt[R].l, l, m, k);
    else return query(hjt[L].r, hjt[R].r, m + 1, r, k - tmp);
}

int main() {
    int n, q;
    cin >> n >> q;
    vector<int> a(n + 1);
    for(int i = 1;i <= n; i++) { cin >> a[i]; v.push_back(a[i]); }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());

    for(int i = 1;i <= n; i++) {
        insert(root[i - 1], root[i], 1, n, get_id(a[i]));
    }

    while(q--) {
        int l, r, k;
        cin >> l >> r >> k;
        cout << v[query(root[l - 1], root[r], 1, n, k) - 1] << endl;
    }
}
```

3.43 区间前 k 大

```

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const int N = 1e6 + 10;
int a[N];
vector<int> v;
int cnt, root[N];

struct Node {
    int l, r;
    ll sum;
    int num;
    int val;
}hjt[N * 40];

int getid(int x) { return lower_bound(v.begin(), v.end(), x) - v.begin() + 1; }

void insert(int pre, int &now, int l, int r, int p, int val) {
    now = ++cnt;
    hjt[now] = hjt[pre];
    hjt[now].num++; hjt[now].sum += val;
    if(l == r) {
        hjt[now].val = val;
        return ;
    }
    int m = (l + r) >> 1;
    if(p <= m) insert(hjt[pre].l, hjt[now].l, l, m, p, val);
    else insert(hjt[pre].r, hjt[now].r, m + 1, r, p, val);
}

ll query(int L, int R, int l, int r, int k) {
    if(l == r) return hjt[R].val * k;
    int m = (l + r) >> 1;
    int tmp = hjt[hjt[R].r] .num - hjt[hjt[L].r].num;
    if(k <= tmp) return query(hjt[L].r, hjt[R].r, m + 1, r, k);
    else return hjt[hjt[R].r].sum - hjt[hjt[L].r].sum + query(hjt[L].l, hjt[R].l, l, m, k - tmp);
}

void init(int n) {
    v.clear();
    cnt = 0;
    for(int i = 1;i <= n; i++) {
        scanf("%d",&a[i]);
        v.push_back(a[i]); root[i] = 0;
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
}

int main() {
    int _;
    scanf("%d",&_);
    while(_--) {
        int n;
        scanf("%d",&n);
        init(n);
        for(int i = 1;i <= n; i++) {
            insert(root[i - 1], root[i], 1, n, getid(a[i]), a[i]);
        }
        int q;
        scanf("%d",&q);
        while(q--) {
            int l, r, k;
            scanf("%d%d%d",&l,&r,&k);
            int t = r - l + 1;
            ll ans = query(root[l - 1], root[r], 1, n, k);
            printf("%lld\n",1ll * t * (t + 1) * (2 * t + 1) / 6 + ans);
        }
    }
}
```

3.44 树套树维护三维偏序

```
const int N = 4e6 + 10;

int n, k;
struct node {
    int a, b, c;
    int operator < (const node &o) const {
        return a != o.a ? (a < o.a) : (b != o.b ? (b < o.b) : (c < o.c));
    }

    int operator == (const node &o) const {
        return a == o.a && b == o.b && c == o.c;
    }
}p[N];


struct Tree1 {
    int l, r;
}t1[N << 2];

struct Tree2 {
    int l, r;
    int num;
}t2[N << 2];

int root, root2[N];
int cnt1, cnt2;

void vec_insert(int &u, int l, int r, int pos, int val) {
    if(!u) u = ++cnt2;
    t2[u].num += val;
    if(l == r) return ;
    int m = (l + r) / 2;
    if(pos <= m) vec_insert(t2[u].l, l, m, pos, val);
    else vec_insert(t2[u].r, m + 1, r, pos, val);
}

int vec_query(int u, int l, int r, int ql, int qr) {
    if(!u) return 0;
    if(ql <= l && r <= qr) return t2[u].num;
    int ans = 0;
    int mid = (l + r) / 2;
    if(ql <= mid) ans += vec_query(t2[u].l, l, mid, ql, qr);
    if(qr  > mid) ans += vec_query(t2[u].r, mid + 1, r, ql, qr);
    return ans;
}

// 在第一维权值线段树在[1,k]根据p[x].b插入，第二维权值线段树在[1,k]根据p[x].c插入
void tree_insert(int &u, int l, int r, int x, int val) {
    if(!u) u = ++cnt1;
    vec_insert(root2[u], 1, k, p[x].c, val);
    if(l == r) return ;
    int m = (l + r) / 2;
    if(p[x].b <= m) tree_insert(t1[u].l, l, m, x, val);
    else tree_insert(t1[u].r, m + 1, r, x, val);
}

int tree_query(int u, int l, int r, int x) {
    if(!u) return 0;
    if(1 <= l && r <= p[x].b) return vec_query(root2[u], 1, k, 1, p[x].c);
    int mid = (l + r) / 2;
    int ans = 0;
    if(1 <= mid) ans += tree_query(t1[u].l, l, mid, x);
    if(p[x].b > mid) ans += tree_query(t1[u].r, mid + 1, r, x);
    return ans;
```

```
}

void solve() {
    cin >> n >> k;
    for(int i = 1;i <= n; i++) cin >> p[i].a >> p[i].b >> p[i].c;
    sort(p + 1, p + n + 1);
    vector<int> ans(n + 1);
    int sum = 1;
    for(int i = 1;i <= n; i++) {
        // 因为这些个都一样，如果不这样操作，会使后面的不会对前面的有贡献
        if(p[i + 1] == p[i]) {
            sum++;
            continue;
        }
        tree_insert(root, 1, k, i, sum);
        int res = tree_query(root, 1, k, i);
        ans[res] += sum;
        sum = 1;
    }
    for(int i = 1;i <= n; i++) cout << ans[i] << endl;
}
```

3.45 线段树套主席树-二维区间不同数

```
const int N = 2e5 + 10;

int n, m, l, r, a, b, num[N], Last[N], pre[N], Hash[N], ans[N][2];
int cnt, root[N], sum[N * 20], lc[N * 20], rc[N * 20];
struct Query {
    int a, b, l, id;
};

struct data {
    int a, v;

    bool operator<(const data &b) const {
        return a < b.a;
    }
} d[N];

vector<Query> q[N * 4];

void addquery(int o, int l, int r, int L, int R, Query qry) {
    if (L <= l && R >= r) {
        q[o].push_back(qry);
        return;
    }
    int mid = (l + r) / 2;
    if (L <= mid) {
        addquery(o * 2, l, mid, L, R, qry);
    }
    if (R > mid) {
        addquery(o * 2 + 1, mid + 1, r, L, R, qry);
    }
}

void build(int y, int &x, int l, int r, int k) {
    x = ++cnt;
    sum[x] = sum[y] + 1;
    lc[x] = lc[y];
    rc[x] = rc[y];
    if (l == r) {
        return;
    }
    int mid = (l + r) / 2;
    if (k <= mid) {
        build(lc[y], lc[x], l, mid, k);
    } else {
        build(rc[y], rc[x], mid + 1, r, k);
    }
}

int query(int y, int x, int l, int r, int k) {
    if (!x || l == r) {
        return 0;
    }
    int mid = (l + r) / 2;
    if (k <= mid) {
        return query(lc[y], lc[x], l, mid, k);
    } else {
        return sum[lc[x]] - sum[lc[y]] + query(rc[y], rc[x], mid + 1, r, k);
    }
}

void insert(int o, int l, int r) {
    if (q[o].size()) {
        Hash[0] = 0;
        for (int i = l; i <= r; i++) {
            Hash[++Hash[0]] = num[i];
        }
        sort(Hash + 1, Hash + Hash[0] + 1);
        int s = 0;
        for (int i = l; i <= r; i++) {
            d[++s].a = lower_bound(Hash + 1, Hash + Hash[0] + 1, num[i]) - Hash;
            d[s].v = pre[i];
        }
        sort(d + 1, d + s + 1);
        for (int i = 1; i <= s; i++) {
            build(root[i - 1], root[i], 0, n, d[i].v);
        }
        int a, b;
        for (int i = 0; i < q[o].size(); i++) {
            a = lower_bound(Hash + 1, Hash + Hash[0] + 1, q[o][i].a) - Hash;
            b = upper_bound(Hash + 1, Hash + Hash[0] + 1, q[o][i].b) - Hash - 1;
            ans[q[o][i].id][0] += sum[root[b]] - sum[root[a - 1]];
            ans[q[o][i].id][1] += query(root[a - 1], root[b], 0, n, q[o][i].l);
        }
        memset(root, 0, sizeof(int) * (Hash[0] + 1));
        memset(sum, 0, sizeof(int) * (cnt + 1));
        memset(lc, 0, sizeof(int) * (cnt + 1));
        memset(rc, 0, sizeof(int) * (cnt + 1));
        cnt = 0;
    }
    if (l == r) {
        return;
    }
    int mid = (l + r) / 2;
    insert(o * 2, l, mid);
    insert(o * 2 + 1, mid + 1, r);
}

void solve() {
    int _;
    cin >> _;
    while (_--) {
        for (int i = 0; i < N * 4; i++) q[i].clear();
        cnt = 0;
        for (int i = 0; i < N; i++) ans[i][0] = ans[i][1] = pre[i] = Last[i] = 0;
        cin >> n >> m;
        for (int i = 1; i <= n; i++) {
            cin >> num[i];
            num[i]++;
            pre[i] = Last[num[i]];
            Last[num[i]] = i;
        }
        for (int i = 1; i <= m; i++) {
            cin >> l >> a >> r >> b;
            a++, b++;
            addquery(1, 1, 1e5 + 1, l, r, (Query) {a, b, l, i});
        }
        insert(1, 1, 1e5 + 1);
        for (int i = 1; i <= m; i++) {
            cout << ans[i][1] << endl;
        }
    }
}
```

3.46 Scapegoat

```
// 无旋转平衡，暴力拍扁重构

#include <bits/stdc++.h>
using namespace std;

namespace Scapegoat_Tree {
#define MAXN (100000 + 10)
    const double alpha = 0.75;
    struct Node {
    Node * ch[2];
    int key, size, cover; // size为有效节点的数量，cover为节点总数量
    bool exist; // 是否存在（即是否被删除）
    void PushUp(void) {
        size = ch[0]->size + ch[1]->size + (int)exist;
        cover = ch[0]->cover + ch[1]->cover + 1;
    }
    bool isBad(void) { // 判断是否需要重构
        return ((ch[0]->cover > cover * alpha + 5) ||
                (ch[1]->cover > cover * alpha + 5));
        }
    };
    struct STree {
    protected:
        Node mem_poor[MAXN]; //内存池，直接分配好避免动态分配内存占用时间
        Node *tail, *root, *null; // 用null表示NULL的指针更方便，tail为内存分配指针，root为根
        Node *bc[MAXN]; int bc_top; // 储存被删除的节点的内存地址，分配时可以再利用这些地址

        Node * NewNode(int key) {
            Node * p = bc_top ? bc[--bc_top] : tail++;
            p->ch[0] = p->ch[1] = null;
            p->size = p->cover = 1; p->exist = true;
            p->key = key;
            return p;
        }
        void Travel(Node * p, vector<Node *>&v) {
            if (p == null) return;
            Travel(p->ch[0], v);
            if (p->exist) v.push_back(p); // 构建序列
            else bc[bc_top++] = p; // 回收
            Travel(p->ch[1], v);
        }
        Node * Divide(vector<Node *>&v, int l, int r) {
            if (l >= r) return null;
            int mid = (l + r) >> 1;
            Node * p = v[mid];
            p->ch[0] = Divide(v, l, mid);
            p->ch[1] = Divide(v, mid + 1, r);
            p->PushUp(); // 自底向上维护，先维护子树
            return p;
        }
        void Rebuild(Node * &p) {
            static vector<Node *>v; v.clear();
            Travel(p, v); p = Divide(v, 0, v.size());
        }
        Node ** Insert(Node *&p, int val) {
            if (p == null) {
                p = NewNode(val);
                return &null;
            }
            else {
                p->size++; p->cover++;

                // 返回值储存需要重构的位置，若子树也需要重构，本节点开始也需要重构，以本节点为根重构
                Node ** res = Insert(p->ch[val >= p->key], val);
                if (p->isBad()) res = &p;
                return res;
            }
        }
        void Erase(Node *p, int id) {
            p->size--;
            int offset = p->ch[0]->size + p->exist;
            if (p->exist && id == offset) {
                p->exist = false;
                return;
            }
            else {
                if (id <= offset) Erase(p->ch[0], id);
                else Erase(p->ch[1], id - offset);
            }
        }
    public:
        void Init(void) {
            tail = mem_poor;
            null = tail++;
            null->ch[0] = null->ch[1] = null;
            null->cover = null->size = null->key = 0;
            root = null; bc_top = 0;
        }
        STree(void) { Init(); }

        void Insert(int val) {
            Node ** p = Insert(root, val);
            if (*p != null) Rebuild(*p);
        }
        int Rank(int val) {
            Node * now = root;
            int ans = 1;
            while (now != null) { // 非递归求排名
                if (now->key >= val) now = now->ch[0];
                else {
                    ans += now->ch[0]->size + now->exist;
                    now = now->ch[1];
                }
            }
            return ans;
        }
        int Kth(int k) {
            Node * now = root;
            while (now != null) { // 非递归求第K大
                if (now->ch[0]->size + 1 == k && now->exist) return now->key;
                else if (now->ch[0]->size >= k) now = now->ch[0];
                else k -= now->ch[0]->size + now->exist, now = now->ch[1];
            }
        }
        void Erase(int k) {
            Erase(root, Rank(k));
            if (root->size < alpha * root->cover) Rebuild(root);
        }
        void Erase_kth(int k) {
            Erase(root, k);
            if (root->size < alpha * root->cover) Rebuild(root);
        }
    }sTree;
#undef MAXN

}

int main() {
    Scapegoat_Tree::sTree.Init();
    int _; cin >> _;
    while(_--) {
        int opt, x;
        cin >> opt >> x;
        if(opt == 1) Scapegoat_Tree::sTree.Insert(x);
        else if(opt == 2) Scapegoat_Tree::sTree.Erase(x);
        else if(opt == 3) cout << Scapegoat_Tree::sTree.Rank(x) << endl;
        else if(opt == 4) cout << Scapegoat_Tree::sTree.Kth(x) << endl;
        else if(opt == 5) cout << Scapegoat_Tree::sTree.Kth(Scapegoat_Tree::sTree.Rank(
    x) - 1) << endl;
        else if(opt == 6) cout << Scapegoat_Tree::sTree.Kth(Scapegoat_Tree::sTree.Rank(
    x + 1)) << endl;
    }
}
```

3.47 Splay

```
class node {
public:
    int id;
    node *l;
    node *r;
    node *p;
    bool rev;
    int sz;
    // declare extra variables

    node(int _id) {
        id = _id;
        l = r = p = nullptr;
        rev = false;
        sz = 1;
        // init extra variables
    }

    void unsafe_reverse() {
        rev ^= 1;
        std::swap(l, r);
        pull();
    }

    // apply changes:
    void unsafe_apply() {
    }

    void push() {
        if (rev) {
            if (l != nullptr) {
                l -> unsafe_reverse();
            }
            if (r != nullptr) {
                r -> unsafe_reverse();
            }
            rev = 0;
        }
        // now push everything else:
    }

    void pull() {
        sz = 1;
        // now init from self:

        if (l != nullptr) {
            l -> p = this;
            sz += l -> sz;
            // now pull from l:

        }
        if (r != nullptr) {
            r -> p = this;
            sz += r -> sz;
            // now pull from r:

        }
    }
};

void debug_node(node* v, std::string pref = "") {
#ifdef LOCAL
    if (v != nullptr) {
        debug_node(v -> r, pref + " ");
        std::cerr << pref << "-" << " " << v -> id << '\n';
        debug_node(v -> l, pref + " ");
    } else {
        std::cerr << pref << "-" << " " << "NULL" << '\n';
    }
#endif
}

namespace splay_tree {

bool is_bst_root(node* v) {
    if (v == nullptr) {
        return false;
    }
    return (v -> p == nullptr || (v -> p -> l != v && v -> p -> r != v));
}
void rotate(node* v) {
    node* u = v -> p;
    assert(u != nullptr);
    u -> push();
    v -> push();
    v -> p = u -> p;
    if (v -> p != nullptr) {
        if (v -> p -> l == u) {
            v -> p -> l = v;
        }
        if (v -> p -> r == u) {
            v -> p -> r = v;
        }
    }
    if (v == u -> l) {
        u -> l = v -> r;
        v -> r = u;
    } else {
        u -> r = v -> l;
        v -> l = u;
    }
    u -> pull();
    v -> pull();
}

void splay(node* v) {
    if (v == nullptr) {
        return ;
    }
    while (!is_bst_root(v)) {
        node* u = v -> p;
        if (!is_bst_root(u)) {
            if ((u -> l == v) ^ (u -> p -> l == u)) {
                rotate(v);
            } else {
                rotate(u);
            }
        }
        rotate(v);
    }
}

std::pair<node*, int> find(node* v, const std::function<int(node*)> &go_to) {
    // go_to returns: 0 -- found; -1 -- go left; 1 -- go right
    // find returns the last vertex on the descent and its go_to
    if (v == nullptr) {
        return {nullptr, 0};
    }
    splay(v);
    int dir;
    while (true) {
        v -> push();
        dir = go_to(v);
        if (dir == 0) {
            break ;
        }
        node* u = (dir == -1 ? v -> l : v -> r);
        if (u == nullptr) {
            break ;
        }
        v = u;
    }
    splay(v);
    return {v, dir};
}

node* get_leftmost(node* v) {
    return find(v, [&](node*) { return -1; }).first;
}

node* get_rightmost(node* v) {
    return find(v, [&](node*) { return 1;  }).first;
}

node* get_kth(node* v, int k) { // 0-indexed
    std::pair<node*, int> p = find(v, [&](node* u) {
        if (u -> l != nullptr) {
            if (u -> l -> sz > k) {
                return -1;
            }
            k -= u -> l -> sz;
        }
        if (k == 0) {
            return 0;
        }
        k--;
        return 1;
    });
    return (p.second == 0 ? p.first : nullptr);
}

int get_position(node* v) { // 0-indexed
    splay(v);
    return (v -> l != nullptr ? v -> l -> sz : 0);
}
```

```

node* get_bst_root(node* v) {
    splay(v);
    return v;
}

std::pair<node*, node*> split(node* v, const std::function<bool(node*)> &is_right) {
    if (v == nullptr) {
        return {nullptr, nullptr};
    }
    std::pair<node*, int> p = find(v, [&](node* u) { return is_right(u) ? -1 : 1; });
    v = p.first;
    v -> push();
    if (p.second == -1) {
        node* u = v -> l;
        if (u == nullptr) {
            return {nullptr, v};
        }
        v -> l = nullptr;
        u -> p = v -> p;
        u = get_rightmost(u);
        v -> p = u;
        v -> pull();
        return {u, v};
    } else {
        node* u = v -> r;
        if (u == nullptr) {
            return {v, nullptr};
        }
        v -> r = nullptr;
        v -> pull();
        return {v, u};
    }
}

std::pair<node*, node*> split_leftmost_k(node* v, int k) {
    return split(v, [&](node* u) {
        int left_and_me = (u -> l != nullptr ? u -> l -> sz : 0) + 1;
        if (k >= left_and_me) {
            k -= left_and_me;
            return false;
        }
        return true;
    });
}

node* merge(node* v, node* u) {
    if (v == nullptr) {
        return u;
    }
    if (u == nullptr) {
        return v;
    }
    v = get_rightmost(v);
    assert(v -> r == nullptr);
    splay(u);
    v -> push();
    v -> r = u;
    v -> pull();
    return v;
}

int count_left(node* v, const std::function<bool(node*)> &is_right) {
    if (v == nullptr) {
        return 0;
    }
    std::pair<node*, int> p = find(v, [&](node* u) { return is_right(u) ? -1 : 1; });
    node* u = p.first;
    return (u -> l != nullptr ? u -> l -> sz : 0) + (p.second == 1);
}

node* add(node* r, node* v, const std::function<bool(node*)> &go_left) {
    std::pair<node*, node*> p = split(r, go_left);
    return merge(p.first, merge(v, p.second));
}

node* remove(node* v) { // returns the new root
    splay(v);
    v -> push();
    node* x = v -> l;
    node* y = v -> r;
    v -> l = v -> r = nullptr;
    node* z = merge(x, y);
    if (z != nullptr) {
        z -> p = v -> p;
    }
    v -> p = nullptr;
    v -> push();
    v -> pull(); //  now v might be reusable
    return z;
}

node* next(node* v) {
    splay(v);
    v -> push();
    if (v -> r == nullptr) {
        return nullptr;
    }
    v = v -> r;
    while (v -> l != nullptr) {
        v -> push();
        v = v -> l;
    }
    splay(v);
    return v;
}

node* prev(node* v) {
    splay(v);
    v -> push();
    if (v -> l == nullptr) {
        return nullptr;
    }
    v = v -> l;
    while (v -> r != nullptr) {
        v -> push();
        v = v -> r;
    }
    splay(v);
    return v;
}

int get_size(node* v) {
    splay(v);
    return (v != nullptr ? v -> sz : 0);
}

template<typename... T>
void apply(node* v, T... args) {
    splay(v);
    v -> unsafe_apply(args...);
}

void reverse(node* v) {
    splay(v);
    v -> unsafe_reverse();
}

} // namespace splay_tree
```

3.48 01trie

```
template<typename T>
class Trie {
private :
    Trie* next[2] = {nullptr};
    int val;
    const int maxl = 32;
public :
    Trie() {}

    void insert(T x) {
        Trie* root = this;
        for(int i = maxl; i >= 0; i--) {
            int u = x >> i & 1;
            if(root -> next[u] == nullptr) root -> next[u] = new Trie();
            root = root -> next[u];
            root -> val ++;
        }
    }

    void del(T x) {
        Trie* root = this;
        for (int i = maxl; i >= 0; i--) {
            root = root -> next[x >> i & 1];
            root -> val --;
        }
    }

    T search(T x) {
        T ans = 0;
        Trie* root = this;
        for (int i = maxl; i >= 0; i--) {
            int u = x >> i & 1;
            if (root -> next[!u] && root -> next[!u] -> val) {
                ans += 1 << i;
                root = root -> next[!u];
            } else {
                root = root -> next[u];
            }
        }
        return ans;
    }
};
```

3.49 Treap

```
struct Tree {
    Tree *l;
    Tree *r;
    int x;
    int siz;
    Tree(Tree *v) { *this = *v; }
    Tree(int x = 0) : l(nullptr), r(nullptr), x(x), siz(1) {}
    void pull() {
        siz = 1;
        if (l != nullptr) {
            siz += l->siz;
        }
        if (r != nullptr) {
            siz += r->siz;
        }
    }
};

int cnt = 0;
constexpr int N = 1e7;
Tree pool[N];

std::mt19937 rnd(std::chrono::steady_clock::now().time_since_epoch().count());

template<class... T>
Tree *newTree(T... x) {
    Tree *t = &pool[cnt++];
    *t = Tree(x...);
    return t;
}

Tree *merge(Tree *a, Tree *b) {
    if (a == nullptr) {
        return b;
    }
    if (b == nullptr) {
        return a;
    }
    Tree *t;
    if (int(rnd() % (a->siz + b->siz)) < a->siz) {
        t = newTree(a);
        t->r = merge(t->r, b);
    } else {
        t = newTree(b);
        t->l = merge(a, t->l);
    }
    t->pull();
    return t;
}

std::pair<Tree *, Tree *> split(Tree *t, int k) {
    if (t == nullptr || k == 0) {
        return {nullptr, t};
    }
    if (t->siz == k) {
        return {t, nullptr};
    }
    int szl = t->l == nullptr ? 0 : t->l->siz;
    Tree *u = newTree(t);
    if (k <= szl) {
        auto [a, b] = split(t->l, k);
        u->l = b;
        u->pull();
        return {a, u};
    } else {
        auto [a, b] = split(t->r, k - 1 - szl);
        u->r = a;
        u->pull();
        return {u, b};
    }
}
```

4 博弈论

4.1 bash

```cpp
template <typename T>
bool bash(const T& a, const T& b) {
    return a % (b + 1);
}
```

4.2 fibonacci

```cpp
template <typename T>
bool Fibonacci(const T& x) {
    std::unordered_map<int, bool> was;
    std::vector<int> fib(51);
    fib[1] = 1; fib[2] = 2;
    for (int i = 3; i <= 50; i++) {
        fib[i] = fib[i - 1] + fib[i - 2];
        was[fib[i]] = true;
    }
    return !was[fib[x]];
}
```

4.3 nim

```cpp
template <typename T>
bool nimGame(const std::vector<T>& stones) {
    int res = 0;
    for (int &x : stones) {
        res ^= x;
    }
    return res;
}
```

4.4 wythoff

```cpp
template <typename T>
bool wythoff(T& a, T& b) {
    if (a > b) std::swap(a, b);
    T delta = b - a;
    T res = delta * (1.0 + sqrt(5.0)) / 2;
    return !(res == a);
}

template <typename T>
bool wythoff_exp(T& a, T& b, T& k) {
    k++;
    if (a > b) std::swap(a, b);
    T delta = (b - a) / k;
    T res1 = delta * (2 - k + sqrt(4.0 + k * k)) / 2;
    T res2 = delta * (2 + k + sqrt(4.0 + k * k)) / 2;
    return !(res1 == a && res2 == b);
}
```

4.5 sgFunction

```
// SG函数
#define N 1001
//f[]: 可以取走的石子个数
//sg[]:0~n的SG函数值
int f[N], sg[N], mex[N];

void getSG(int n) {
    int i, j;
    memset(sg, 0, sizeof(sg));
    for (i = 1; i <= n; i++) {
        memset(mex, 0, sizeof(mex));
        for (j = 1; f[j] <= i; j++)
            mex[sg[i - f[j]]] = 1;
        for (j = 0; j <= n; j++) { //求mes{}中未出现的最小的非负整数
            if (mex[j] == 0) {
                sg[i] = j;
                break;
            }
        }
    }
}
```

5 树与森林

5.1 forest

```
template <typename T>
class forest : public graph<T> {
public:
    using graph<T>::edges;
    using graph<T>::g;
    using graph<T>::n;

    forest(int _n) : graph<T>(_n) {}

    int add (int from, int to, T cost = 1) {
        assert(0 <= from && from < n && 0 <= to && to < n);
        int id = (int) edges.size();
        assert(id < n - 1);
        g[from].push_back(id);
        g[to].push_back(id);
        edges.push_back({from, to, cost});
        return id;
    }
};
```

5.2 dfs-forest

```
template <typename T>
class dfs_forest : public forest<T> {
public:
    using forest<T>::edges;
    using forest<T>::g;
    using forest<T>::n;

    std::vector<int> pv;
    std::vector<int> pe;
    std::vector<int> order;
    std::vector<int> pos;
    std::vector<int> end;
    std::vector<int> sz;
    std::vector<int> root;
    std::vector<int> depth;
    std::vector<T> dist;

    dfs_forest(int _n) : forest<T>(_n) {}

    void init() {
        pv = std::vector<int>(n, -1);
        pe = std::vector<int>(n, -1);
        order.clear();
        pos = std::vector<int>(n, -1);
        end = std::vector<int>(n, -1);
        sz = std::vector<int>(n, 0);
        root = std::vector<int>(n, -1);
        depth = std::vector<int>(n, -1);
        dist = std::vector<T>(n);
    }

    void clear() {
        pv.clear();
        pe.clear();
        order.clear();
        pos.clear();
        end.clear();
        sz.clear();
        root.clear();
        depth.clear();
        dist.clear();
    }

private:
    void do_dfs(int v) {
        pos[v] = (int) order.size();
        order.push_back(v);
        sz[v] = 1;
        for (int id : g[v]) {
            if (id == pe[v]) {
                continue;
            }
            auto &e = edges[id];
            int to = e.from ^ e.to ^ v;
            depth[to] = depth[v] + 1;
            dist[to] = dist[v] + e.cost;
            pv[to] = v;
            pe[to] = id;
            root[to] = (root[v] != -1 ? root[v] : to);
            do_dfs(to);
            sz[v] += sz[to];
        }
        end[v] = (int) order.size() - 1;
    }

    void do_dfs_from(int v) {
        depth[v] = 0;
        dist[v] = T{};
        root[v] = v;
        pv[v] = pe[v] = -1;
        do_dfs(v);
    }

public:
    void dfs(int v, bool clear_order = true) {
        if (pv.empty()) {
            init();
        } else {
            if (clear_order) {
                order.clear();
            }
        }
        do_dfs_from(v);
    }

    void dfs_all() {
        init();
        for (int v = 0; v < n; v++) {
            if (depth[v] == -1) {
                do_dfs_from(v);
            }
        }
        assert((int) order.size() == n);
    }
};
```

5.3 lca-forest

```
template <typename T>
class lca_forest : public dfs_forest<T> {
public:
    using dfs_forest<T>::edges;
    using dfs_forest<T>::g;
    using dfs_forest<T>::n;
    using dfs_forest<T>::pv;
    using dfs_forest<T>::pos;
    using dfs_forest<T>::end;
    using dfs_forest<T>::depth;

    int h;
    std::vector<std::vector<int>> pr;

    lca_forest(int _n) : dfs_forest<T>(_n) {}

    inline void build_lca() {
        assert(!pv.empty());
        int max_depth = 0;
        for (int i = 0; i < n; i++) {
            max_depth = std::max(max_depth, depth[i]);
        }
        h = 1;
        while ((1 << h) <= max_depth) {
            h++;
        }
        pr.resize(n);
        for (int i = 0; i < n; i++) {
            pr[i].resize(h);
            pr[i][0] = pv[i];
        }
        for (int j = 1; j < h; j++) {
            for (int i = 0; i < n; i++) {
                pr[i][j] = (pr[i][j - 1] == -1 ? -1 : pr[pr[i][j - 1]][j - 1]);
            }
        }
    }

    inline bool anc(int x, int y) {
        return (pos[x] <= pos[y] && end[y] <= end[x]);
    }

    inline int go_up(int x, int up) {
        assert(!pr.empty());
        up = std::min(up, (1 << h) - 1);
        for (int j = h - 1; j >= 0; j--) {
            if (up & (1 << j)) {
                x = pr[x][j];
                if (x == -1) {
                    break;
                }
            }
        }
        return x;
    }

    inline int lca(int x, int y) {
        assert(!pr.empty());
        if (anc(x, y)) {
            return x;
        }
        if (anc(y, x)) {
            return y;
        }
        for (int j = h - 1; j >= 0; j--) {
            if (pr[x][j] != -1 && !anc(pr[x][j], y)) {
                x = pr[x][j];
            }
        }
        return pr[x][0];
    }

    inline int dist(int x, int y) {
        return depth[x] + depth[y] - depth[lca(x, y)] * 2;
    }
};
```

5.4 重链剖分

```
const int maxn = 4e5 + 10;

struct Edge {
    int v, next;
}e[maxn << 1];

int head[maxn * 2], cnt;

inline void add(int u, int v) {
    e[++cnt].v = v;
    e[cnt].next = head[u];
    head[u] = cnt;
}

int fa[maxn], dep[maxn], siz[maxn], son[maxn];

void dfs1(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    for(int i = head[u]; ~i; i = e[i].next) {
        int v = e[i].v;
        if(v == par) continue;
        dfs1(v, u);
        siz[u] += siz[v];
        if(!son[u] || siz[v] > siz[son[u]])
            son[u] = v;
    }
}

int dfn[maxn], top[maxn], nodeof[maxn], tim;

```

```cpp
void dfs2(int u, int topf) {
    nodeof[dfn[u] = ++tim] = u;
    top[u] = topf;
    if(!son[u]) return ;
    dfs2(son[u], topf);
    for(int i = head[u]; ~i; i = e[i].next) {
        int v = e[i].v;
        if(v == fa[u] || v == son[u]) continue;
        dfs2(v, v);
    }
}

int w[maxn];

#define lc u << 1
#define rc u << 1 | 1
#define mid (t[u].l + t[u].r) / 2
struct Tree {
    int l, r, sum, tag;
}t[maxn << 2];
inline void push_up(int u) ;
inline void push_down(int u) ;
void build(int u, int l, int r) ;
void modify(int u, int ql, int qr, int v) ;
int query(int u, int ql, int qr) ;

void modify_chain(int x, int y, int val) {
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        modify(1, dfn[top[x]], dfn[x], val);
        x = fa[top[x]];
    }
    if(dep[x] > dep[y]) swap(x, y);
    modify(1, dfn[x], dfn[y], val);
}

int query_chain(int x, int y) {
    int ans = 0;
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        ans += query(1, dfn[top[x]], dfn[x]);
        x = fa[top[x]];
    }
    if(dep[x] > dep[y]) swap(x, y);
    ans += query(1, dfn[x], dfn[y]);
    return ans;
}

signed main() {
    memset(head, -1, sizeof(head));
    int n; cin >> n;
    for(int i = 1;i <= n; i++) cin >> w[i];
    for(int i = 1;i <= n - 1; i++) {
        int u, v; cin >> u >> v;
        add(u, v);
        add(v, u);
    }
    dfs1(1, 0);
    dfs2(1, 1);
    build(1, 1, n);
    int m; cin >> m;
    while(m--) {
        int opt; cin >> opt;
        if(opt == 1) {
            int x, y, val; cin >> x >> y >> val;
            modify_chain(x, y, val);
        }
        else if(opt == 2) {
            int x, val; cin >> x >> val;
            modify(1, dfn[x], dfn[x] + siz[x] - 1, val);
        }
        else if(opt == 3) {
            int x, y; cin >> x >> y;
            cout << query_chain(x, y) << endl;
        }
        else if(opt == 4) {
            int x; cin >> x;
            cout << query(1, dfn[x], dfn[x] + siz[x] - 1) << endl;
        }
    }
}
```

5.5 tree-diameter

```
template <typename T>
std::vector<int> find_tree_diameter(const forest<T>& g, T& diameter) {
    diameter = 0;
    int st = 0, ed = 0;
    std::vector<T> dis(g.n);
    std::vector<int> pre(g.n);

    std::function<void(int, int)> dfs1 = [&](int u, int parent) {
        if (dis[u] > dis[st]) st = u;
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (to == parent) continue ;
            dis[to] = dis[u] + e.cost;
            dfs1(to, u);
        }
    };

    std::function<void(int, int)> dfs2 = [&](int u, int parent) {
        if (dis[u] > dis[ed]) ed = u;
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (to == parent) continue ;
            pre[to] = u;
            dis[to] = dis[u] + e.cost;
            dfs2(to, u);
        }
    };

    dfs1(0, -1);
    dis.assign(g.n, 0);
    dfs2(st, -1);
```

```cpp
    std::vector<int> vertexs{ed};
    int now = ed;
    do {
        vertexs.push_back(pre[now]);
        now = pre[now];
    } while (now != st);
    reverse(vertexs.begin(), vertexs.end());
    return vertexs;
}

template <typename T>
std::vector<int> find_tree_all_diameters(const forest<T>& g, T& diameter) {
    diameter = 0;
    std::vector<std::array<T, 2>> dp(g.n);
    std::vector<int> down(g.n);
    std::vector<T> up(g.n);

    std::function<void(int, int)> dfs1 = [&](int u, int parent) {
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (to == parent) continue ;
            dfs1(to, u);
            if (dp[to][0] + e.cost > dp[u][0]) {
                dp[u][1] = dp[u][0];
                dp[u][0] = dp[to][0] + e.cost;
                down[u] = to;
            } else if (dp[to][0] + e.cost > dp[u][1]) {
                dp[u][1] = dp[to][0] + e.cost;
            }
            diameter = std::max(diameter, dp[u][1] + dp[u][0]);
        }
    };

    std::function<void(int, int)> dfs2 = [&](int u, int parent) {
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (to == parent) continue;
            up[to] = up[u] + e.cost;
            if (down[u] == to) up[to] = std::max(up[to], dp[u][1] + e.cost);
            else up[to] = std::max(up[to], dp[u][0] + e.cost);
            dfs2(to, u);
        }
    };

    dfs1(0, -1);
    dfs2(0, -1);
    std::vector<int> vertexs;
    for (int i = 0; i < g.n; i++) {
        std::vector<int> dis{ dp[i][1], dp[i][0], up[i] };
        sort(dis.begin(), dis.end());
        if (dis[1] + dis[2] == diameter) vertexs.push_back(i);
    }
    return vertexs;
};
```

5.6 树的重心

```
const int N = 1e5 + 10;

struct Edge {
    int v, next;
}e[N * 2];

int cnt, head[N * 2];

int d[N], R[2], root;
int n;

int balance;

inline void add(int u, int v) {
    e[++cnt].v = v;
    e[cnt].next = head[u];
    head[u] = cnt;
}

void DFS(int u, int fa) {
    d[u] = 1;
    int res = 0;
    for(int i = head[u] ; i != -1 ; i = e[i].next) {
        int v = e[i].v;
        if(v == fa) continue;
        DFS(v, u);
        d[u] += d[v];
        res = max(res, d[v]);
    }
    res = max(res, n - d[u]);
    if(res < balance) {
        R[root++] = u;
        balance = res;
    }
    else if(res == balance) {
        R[root++] = u;
    }
}

int main() {
    cin >> n;
    balance = n / 2;
    for(int i = 1;i < n; i++) {
        int u, v;
        cin >> u >> v;
        add(u, v);
        add(v, u);
    }
    DFS(1, 0);
    if(R[0]) cout << R[0] << endl;
    if(R[1]) cout << R[1] << endl;
}
```

5.7 树的最大匹配

```
// 设状态为f[u][1/0]表示以u为根的子树与儿子连边/不连边的最大匹配
```

```cpp

const int N = 1e5 + 10;
vector<int> g[N];
int f[N][2];

void dfs(int u, int fa) {
    int mn = INF;
    for(auto v : g[u]) {
        if(v == fa) continue ;
        dfs(v, u);
        f[u][0] += f[v][1]; // u不与儿子连边，即加上所有与儿子连边的v
        f[u][1] += f[v][1]; // u与儿子连边，即加上一个不与儿子连边的v和其他所有与儿子连边的v
        mn = min(mx, f[v][1] - f[v][0]);
    }
    if(mn != INF) f[u][1] = dp[u][1] - mx + 1;
}
```

5.8 树分治-点分治

```cpp
// 题意：n个节点的树，存在边权，范围1e18
// 求任意两点之间点集的子集中两点之间路径异或和为0的个数
// u<v,u'<v',(u',v') □  path(u,v),求path(u', v')异或和==0

struct Edge {
    int to, nxt;
    ll w;
};
const int N = int(1e5 + 10);
const int M = N << 1;

struct Grahp {
    int head[N];
    Edge eg[M];
    int tot;

    void init(int n) {
        memset(head, -1, sizeof(int) * ++n);
    }

    inline void addEdge(int u, int v, ll w) {
        eg[tot] = {v, head[u], w};
        head[u] = tot++;
    }
} gh;

bool vis[N];
// q队列，fa祖先，sz是子树大小，smx是子树最大
int q[N], fa[N], sz[N], smx[N];

int froot(int s) {
    int l, r, mn = N, rt = 0;
    q[l = r = 1] = s;
    while (l <= r) {
        int u = q[l++];
        sz[u] = 1;
        smx[u] = 0;
        for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
            int v = gh.eg[i].to;
            if (v == fa[u] || vis[v]) continue;
            fa[v] = u;
            q[++r] = v;
        }
    }
    // 反向遍历所有点算size
    while (--l) {
        int u = q[l];
        int mx = max(smx[u], r - sz[u]);
        if (mx < mn) mn = mx, rt = u;
        if (l == 1) break; // 根节点没有fa
        sz[fa[u]] += sz[u];
        smx[fa[u]] = max(smx[fa[u]], sz[u]);
    }
    return rt;
}

// sons子树方向节点个数，val根到该节点异或和，gc边后继方向的节点个数
int sons[N], gc[M];
ll val[N];
ll ans = 0;
int n;

const int MOD = int(1e9 + 7);

ll nums[N];
int cnt[N];

void go(int s, int rt) {
    fa[s] = rt;
    val[s] = 0;
    int l, r;
    // 不计算s
    q[l = r = 0] = s;
    int m = 0;
    while (l <= r) {
        int u = q[l++];
        nums[m++] = val[u];
        for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
            int v = gh.eg[i].to;
            if (v == fa[u] || vis[v]) continue;
            fa[v] = u;
            q[++r] = v;
            val[v] = val[u] ^ gh.eg[i].w;
            // 这个点方向后面有多少点
            sons[v] = gc[i];
        }
    }
    sort(nums, nums + m);
    m = unique(nums, nums + m) - nums;
    mst(cnt, 0, m);
    // 遍历分支
    for (int j = gh.head[s]; ~j; j = gh.eg[j].nxt) {
        // 分支的根
        int du = gh.eg[j].to;
        if (vis[du]) continue;
        q[l = r = 1] = du;
        while (l <= r) {
            int u = q[l++];
            int k = lower_bound(nums, nums + m, val[u]) - nums;
            (ans += 1ll * sons[u] * cnt[k] % MOD) %= MOD;
            if (val[u] == 0) {
                (ans += 1ll * sons[u] * (n - gc[j]) % MOD) %= MOD;
            }
            for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
                int v = gh.eg[i].to;
                if (v == fa[u] || vis[v]) continue;
                q[++r] = v;
            }
        }
        // 增加这个方向的值
        while (--l) {
            int u = q[l];
            int k = lower_bound(nums, nums + m, val[u]) - nums;
            (cnt[k] += sons[u]) %= MOD;
        }
    }
}

void work(int u) {
    // 换根
    u = froot(u);
    vis[u] = true;
    go(u, 0);
    for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
        int v = gh.eg[i].to;
        if (vis[v]) continue;
        work(v);
    }
}

// 预处理边后继节点个数
int pdfs(int u, int f) {
    int fg_id = -1;
    int s = 1;
    for (int i = gh.head[u]; ~i; i = gh.eg[i].nxt) {
        int v = gh.eg[i].to;
        if (v == f) { // 记录父边ID
            fg_id = i;
            continue;
        }
        int c = pdfs(v, u);
        gc[i] = c;
        s += c;
    }
    // 存在父边
    if (~fg_id) gc[fg_id] = n - s;
    return s;
}

void solve() {
    while (cin >> n) {
        gh.init(n);
        for (int i = 2; i <= n; i++) {
            int u, v;
            ll w;
            u = i;
            cin >> v >> w;
            gh.addEdge(u, v, w);
            gh.addEdge(v, u, w);
        }
        mst(vis, false, n + 1);
        pdfs(1, 0);
        ans = 0;
        work(1);
        cout << ans << endl;
    }
}
```

5.9 树上 dsu-维护路径信息

```
const int N = 1e5 + 10;

vector<int> g[N];
int siz[N], dep[N], son[N], dfn[N], nodeof[N], tim;

void calc(int u, int w) {
    // ....对u这一节点进行单独处理
    if(w > 0) // ....计算贡献
    else // ....撤销影响
}

void dfs1(int u, int fa) {
    dep[u] = dep[fa] + (siz[u] = 1);
    nodeof[dfn[u] = ++tim] = u;
    for(auto v : g[u]) {
        if(v == fa) continue ;
        dfs1(v, u);
        siz[u] += siz[v];
        if(!son[u] || siz[v] > siz[son[u]]) son[u] = v;
    }
}

void dfs2(int u, int fa, bool keep) {
    for(auto v : g[u]) {
        if(v == fa || v == son[u]) continue ;
        dfs2(v, u, 0);
    }
    if(son[u]) {
        dfs2(son[u], u, 1);
    }
    for(auto v : g[u]) {
        if(v == fa || v == son[u]) continue ;
        for(int j = 0;j < siz[v]; j++) {
            // ....更新答案
        }
        for(int j = 0;j < siz[v]; j++) {
            calc(nodeof[dfn[v] + j], 1);
        }
    }
    calc(u, 1);
    // ....更新答案
    if(!keep) {
        for(int i = 0;i < siz[u]; i++) calc(nodeof[dfn[u] + i], -1);
    }
}
```

```

int main() {
    int n; cin >> n;
    for(int i = 1;i < n; i++) {
        int u, v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs1(1, 0);
    dfs2(1, 0, 0);
}
```

5.10 树上 dsu-维护子树信息

```
const int N = 2e5 + 10;

vector<int> g[N];

int siz[N], son[N], col[N];
int ans[N], cnt[N];
bool vis[N];
int maxx, sum;
// maxx为每棵子树里出现最多的颜色，sum为编号和


void calc(int u, int fa, int val) {
    /*
    针对不同问题，采取的操作
    */
    else if(val > 0 && cnt[col[u]] == maxx) sum += col[u];
    for(auto v : g[u]) {
        if(v != fa && !vis[v]) calc(v, u, w);
    }
}

void dfs1(int u, int fa) {
    siz[u] = 1;
    for(auto v : g[u]) {
        if(v == fa) continue ;
        dfs1(v, u);
        siz[u] += siz[v];
        if(!son[u] || siz[v] > siz[son[u]]) son[u] = v;
    }
}

void dfs2(int u, int fa, bool keep) {
    for(auto v : g[u]) {
        if(v != fa && v != son[u]) {
            dfs2(v, u, 0);
        }
    }
    if(son[u]) {
        dfs2(son[u], u, 1);
        vis[son[u]] = 1;
    }
    calc(u, fa, 1);
    ans[u] = sum;
    if(son[u]) vis[son[u]] = 0;
    if(!keep) {
        calc(u, fa, -1);
        maxx = sum = 0;
    }
}

int main() {
    int n; cin >> n;
    for(int i = 1;i <= n; i++) cin >> col[i];
    for(int i = 1;i < n; i++) {
        int u,  v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs0(1, 0);
    dfs1(1, 0, false);
    for(int i = 1;i <= n; i++) cout << ans[i] << endl;
}
```

5.11 树上 K 祖先

```
//倍增KFA，空间大点，但是好写
vector<int> g[N];

int anc[N][20];
void dfs(int u, int fa) {
    anc[u][0] = fa;
    for (int i = 1; i <= 19; i++) anc[u][i] = anc[anc[u][i - 1]][i - 1];
    for (auto &v: g[u])
        if (v != fa) dfs(v, u);
}

int kthFa(int u, int k) {
    int bit = 0;
    while (k) {
        if (k & 1) u = anc[u][bit];
        k >>= 1;
        bit++;
    }
    return u;
}


//树剖KFA
int siz[N], son[N], dep[N], fa[N], top[N];
int id[N], nodeOf[N], cnt;
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    for (auto &v: g[u])
        if (v != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (!son[u] || siz[v] > siz[son[u]])
                son[u] = v;
        }
}

void dfs2(int u, int topf) {
    nodeOf[id[u] = ++cnt] = u, top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (auto &v: g[u])
        if (v != fa[u] && v != son[u]) dfs2(v, v);
}

int kthFa(int u, int k) {
    while (k >= id[u] - id[top[u]] + 1 && u) {
        k -= id[u] - id[top[u]] + 1;
        u = fa[top[u]];
    }
    return nodeOf[id[u] - k];
}
```

5.12 virtualTree

```
//虚树可以处理多次询问，并且每次询问只需要树上的K个关键点
//建立的虚树能保证点数 < 2 * K
//如果对虚树做dp，总体复杂度和∑K有关
//考虑dp的时候，需要同时考虑非关键点对答案的影响

int n;

struct edge {
    int nxt, to;
} e[N << 1];
int head[N], tot;
void add(int u, int v) { e[++tot] = edge{ head[u], v }, head[u] = tot;}

int dep[N], fa[N], topfa[N], siz[N], son[N], dfn[N], cnt;
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    int max_son = -1;
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (max_son < siz[v]) son[u] = v, max_son = siz[v];
        }
}
void dfs2(int u, int topf) {
    topfa[u] = topf, dfn[u] = ++cnt;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa[u] && v != son[u]) dfs2(v, v);
}
int LCA(int x, int y) {
    while (topfa[x] != topfa[y]) {
        if (dep[topfa[x]] < dep[topfa[y]]) swap(x, y);
        x = fa[topfa[x]];
    }
    return dep[x] < dep[y] ? x : y;
}
int getDis(int x, int y) { return dep[x] + dep[y] - 2 * dep[LCA(x, y)]; }

//建立虚树
```

```cpp
int tag[N];//tag[u] = 1 <=> 关键点
vector<int> g[N];//虚树边
void add_edge(int u, int v) { g[u].push_back(v); }
int st[N], top, rt;//rt为虚树根
void insert(int u) {
    if (top == 1) {
        st[++top] = u;
        return;
    }
    int lca = LCA(u, st[top]);
    if (lca != st[top]) {
        while (top > 1 && dfn[st[top - 1]] >= dfn[lca])
            add_edge(st[top - 1], st[top]), top--;
        if (lca != st[top]) add_edge(lca, st[top]), st[top] = lca;
    }
    st[++top] = u;
}
bool cmp(const int &x, const int &y) { return dfn[x] < dfn[y]; }
void build(vector<int> &v) {
    st[top = 1] = rt;
    sort(v.begin(), v.end(), cmp);
    for (auto &i: v) {
        tag[i] = 1;
        if (i != rt) insert(i);
    }
    while (top > 1) add_edge(st[top - 1], st[top]), top--;
}


void dp(int u) {
    //...
}
void clear(int u) {//清空虚树边和标记，也可以和dp合并
    for (auto &v: g[u]) clear(v);
    g[u].clear(); tag[u] = 0;
}
void solve() {
    //...
    dp(rt); clear(rt);
    //...
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i < n; i++) {
        int u, v; scanf("%d%d", &u, &v);
        add(u, v); add(v, u);
    }
    //此处距离为1，所以用dep替代dis，dis[fa[rt] = 0] = -1
    dep[0] = -1, rt = 1;
    dfs(rt, 0); dfs2(rt, rt);


    int Q; scanf("%d", &Q);
    while (Q--) {
        int K; scanf("%d", &K);//读取关键点
        for (int i = 1; i <= K; i++) scanf("%d", &a[i]);
        //构建虚树
        build(a);
        solve();
    }

    return 0;
}
```

5.13 LCT

```
int ch[N][2], fa[N], rev[N], siz[N];//基本内容
int sum[N], val[N], tag[N];//另外要维护的
#define lc  ch[u][0]
#define rc  ch[u][1]
#define identify(u) (ch[fa[u]][1] == u)
#define isRoot(u)   (u != ch[fa[u]][0] && u != ch[fa[u]][1])
void flip(int u) { swap(lc, rc); rev[u] ^= 1; }
void push_up(int u) {
    siz[u] = siz[lc] + siz[rc] + 1;
    //...
}
void push_down(int u) {
    if (rev[u]) {
        if (lc) flip(lc);
        if (rc) flip(rc);
        rev[u] = 0;
    }
    //...
}
void update(int u) {//当前点之上的所有点都push_down
    if (!isRoot(u)) update(fa[u]);
    push_down(u);
}
void rotate(int u) {
    int f = fa[u], fc = identify(u);
    int g = fa[f], gc = identify(f);
    int uc = fc ^ 1, c = ch[u][uc];
    if (!isRoot(f))
        ch[g][gc] = u; fa[u] = g;
    ch[f][fc] = c, fa[c] = f;
    ch[u][uc] = f, fa[f] = u;
    push_up(f); push_up(u);
}
void splay(int u) {//将u变为u所在的Splay的根
    update(u);
    for (int f; f = fa[u], !isRoot(u); rotate(u))
        if (!isRoot(f)) rotate(identify(f) ^ identify(u) ? u : f);
}
int access(int u) {//将(rt, u)之间的路径变为实链
    int pre = 0;
    for (; u; u = fa[pre = u])
        splay(u), rc = pre, push_up(u);
    return pre;
}
void makeRoot(int u) {//将u变为整棵树的根(注意:不一定是当前splay的根)
    u = access(u);
    flip(u);
}
int findRoot(int u) {
    access(u), splay(u);
    while (lc) push_down(u), u = lc;
    splay(u);
    return u;
}
void link(int u, int v) {
    makeRoot(u); splay(u);
    if (findRoot(v) != u) fa[u] = v;
}
void split(int u, int v) {
    makeRoot(u);
    access(v); splay(v);//加了这个就将v变为splay的根
}
void cut(int u, int v) {
    makeRoot(u); splay(u);
    if (findRoot(v) == u && fa[v] == u && !ch[v][0]) {
        fa[v] = ch[u][1] = 0;
        push_up(u);
    }
}
void fix(int u, int k) {
    splay(u); val[u] = k;
}
```

6 数学

6.1 Mint

```cpp
template <typename T>
T inverse(T a, T m) {
    T u = 0, v = 1;
    while (a != 0) {
        T t = m / a;
        m -= t * a; std::swap(a, m);
        u -= t * v; std::swap(u, v);
    }
    assert(m == 1);
    return u;
}

template <typename T>
class Modular {
public:
    using Type = typename std::decay<decltype(T::value)>::type;

    constexpr Modular() : value() {}
    template <typename U>
    Modular(const U& x) {
        value = normalize(x);
    }

    template <typename U>
    static Type normalize(const U& x) {
        Type v;
        if (-mod() <= x && x < mod()) v = static_cast<Type>(x);
        else v = static_cast<Type>(x % mod());
        if (v < 0) v += mod();
        return v;
    }

    const Type& operator()() const { return value; }
    template <typename U>
    explicit operator U() const { return static_cast<U>(value); }
    constexpr static Type mod() { return T::value; }

    template <typename U> Modular& operator*=(const U& other) { return *this *= Modular(other); }
    template <typename U> Modular& operator/=(const U& other) { return *this /= Modular(other); }
    Modular& operator+=(const Modular& other) { if ((value += other.value) >= mod()) value -= mod(); return *this; }
    Modular& operator-=(const Modular& other) { if ((value -= other.value) < 0) value += mod(); return *this; }
    template <typename U> Modular& operator+=(const U& other) { return *this += Modular(other); }
    template <typename U> Modular& operator-=(const U& other) { return *this -= Modular(other); }
    Modular& operator++() { return *this += 1; }
    Modular& operator--() { return *this -= 1; }
    Modular operator++(int) { Modular result(*this); *this += 1; return result; }
    Modular operator--(int) { Modular result(*this); *this -= 1; return result; }
    Modular operator-() const { return Modular(-value); }

    template <typename U = T>
    typename std::enable_if<std::is_same<typename Modular<U>::Type, int>::value, Modular>::type& operator*=(const Modular& rhs) {
#ifdef _WIN32
    uint64_t x = static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value);
    uint32_t xh = static_cast<uint32_t>(x >> 32), xl = static_cast<uint32_t>(x), d, m;
    asm(
        "divl %4; \n\t"
        : "=a" (d), "=d" (m)
        : "d" (xh), "a" (xl), "r" (mod())
    );
    value = m;
#else
    value = normalize(static_cast<int64_t>(value) * static_cast<int64_t>(rhs.value));
#endif
    return *this;
    }
    template <typename U = T>
    typename std::enable_if<std::is_same<typename Modular<U>::Type, long long>::value, Modular>::type& operator*=(const Modular& rhs) {
        long long q = static_cast<long long>(static_cast<long double>(value) * rhs.value / mod());
        value = normalize(value * rhs.value - q * mod());
        return *this;
    }
    template <typename U = T>
    typename std::enable_if<!std::is_integral<typename Modular<U>::Type>::value, Modular>::type& operator*=(const Modular& rhs) {
        value = normalize(value * rhs.value);
        return *this;
    }

    Modular& operator/=(const Modular& other) { return *this *= Modular(inverse(other.value, mod())); }

    friend const Type& abs(const Modular& x) { return x.value; }

    template <typename U>
    friend bool operator==(const Modular<U>& lhs, const Modular<U>& rhs);

    template <typename U>
    friend bool operator<(const Modular<U>& lhs, const Modular<U>& rhs);

    template <typename V, typename U>
    friend V& operator>>(V& stream, Modular<U>& number);

private:
    Type value;
};

template <typename T> bool operator==(const Modular<T>& lhs, const Modular<T>& rhs) { return lhs.value == rhs.value; }
template <typename T, typename U> bool operator==(const Modular<T>& lhs, U rhs) { return lhs == Modular<T>(rhs); }
template <typename T, typename U> bool operator==(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) == rhs; }

template <typename T> bool operator!=(const Modular<T>& lhs, const Modular<T>& rhs) { return !(lhs == rhs); }
template <typename T, typename U> bool operator!=(const Modular<T>& lhs, U rhs) { return !(lhs == rhs); }
template <typename T, typename U> bool operator!=(U lhs, const Modular<T>& rhs) { return !(lhs == rhs); }

template <typename T> bool operator<(const Modular<T>& lhs, const Modular<T>& rhs) { return lhs.value < rhs.value; }

template <typename T> Modular<T> operator+(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) += rhs; }
template <typename T, typename U> Modular<T> operator+(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) += rhs; }
template <typename T, typename U> Modular<T> operator+(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) += rhs; }

template <typename T> Modular<T> operator-(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) -= rhs; }
template <typename T, typename U> Modular<T> operator-(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) -= rhs; }
template <typename T, typename U> Modular<T> operator-(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) -= rhs; }

template <typename T> Modular<T> operator*(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) *= rhs; }
template <typename T, typename U> Modular<T> operator*(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) *= rhs; }
template <typename T, typename U> Modular<T> operator*(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) *= rhs; }

template <typename T> Modular<T> operator/(const Modular<T>& lhs, const Modular<T>& rhs) { return Modular<T>(lhs) /= rhs; }
template <typename T, typename U> Modular<T> operator/(const Modular<T>& lhs, U rhs) { return Modular<T>(lhs) /= rhs; }
template <typename T, typename U> Modular<T> operator/(U lhs, const Modular<T>& rhs) { return Modular<T>(lhs) /= rhs; }

template<typename T, typename U>
Modular<T> power(const Modular<T>& a, const U& b) {
    assert(b >= 0);
    Modular<T> x = a, res = 1;
    U p = b;
    while (p > 0) {
        if (p & 1) res *= x;
        x *= x;
        p >>= 1;
    }
    return res;
}

template <typename T>
bool IsZero(const Modular<T>& number) {
    return number() == 0;
}

template <typename T>
std::string to_string(const Modular<T>& number) {
    return to_string(number());
}

```

```
// U == std::ostream? but done this way because of fastoutput
template <typename U, typename T>
U& operator<<(U& stream, const Modular<T>& number) {
    return stream << number();
}

// U == std::istream? but done this way because of fastinput
template <typename U, typename T>
U& operator>>(U& stream, Modular<T>& number) {
    typename std::common_type<typename Modular<T>::Type, long long>::type x;
    stream >> x;
    number.value = Modular<T>::normalize(x);
    return stream;
}

/*
using ModType = int;

struct VarMod { static ModType value; };
ModType VarMod::value;
ModType& md = VarMod::value;
using Mint = Modular<VarMod>;
*/

constexpr int md = 998244353;
using Mint = Modular<std::integral_constant<std::decay<decltype(md)>::type, md>>;

std::vector<Mint> fact(1, 1);
std::vector<Mint> inv_fact(1, 1);

/*Mint C(int n, int k) {
    if (k < 0 || k > n) {
        return 0;
    }
    while ((int) fact.size() < n + 1) {
        fact.push_back(fact.back() * (int) fact.size());
        inv_fact.push_back(1 / fact.back());
    }
    return fact[n] * inv_fact[k] * inv_fact[n - k];
}*/
```

6.2 Z

```
constexpr int P = 998244353;
using i64 = long long;
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    Z(i64 x) : x(norm(x % P)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
    friend std::istream &operator>>(std::istream &is, Z &a) {
        i64 v;
        is >> v;
        a = Z(v);
        return is;
    }
    friend std::ostream &operator<<(std::ostream &os, const Z &a) {
        return os << a.val();
    }
};
```

6.3 exgcd

```
template <typename T>
std::array<T, 3> exgcd(T a, T b) {
    if (b == 0) {
        return {a, 1, 0};
    }
    auto [g, x, y] = exgcd(b, a % b);
    return {g, y, x - a / b * y};
    /*
    auto [g, x, y] = exgcd<long long>(a, b);
    assert(1LL * a * x + 1LL * b * y == g);
    if (c % g != 0) {
        std::cout << -1 << std::endl;
        continue ;
    }
    */
}
```

6.4 factorizer

```
namespace factorizer {

template <typename T>
struct FactorizerVarMod { static T value; };
template <typename T>
T FactorizerVarMod<T>::value;

template <typename T>
bool IsPrime(T n, const std::vector<T>& bases) {
    if (n < 2) {
        return false;
    }
    std::vector<T> small_primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    for (const T& x : small_primes) {
        if (n % x == 0) {
            return n == x;
        }
    }
    if (n < 31 * 31) {
        return true;
    }
    int s = 0;
    T d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }
    FactorizerVarMod<T>::value = n;
    for (const T& a : bases) {
        if (a % n == 0) {
            continue;
        }
        Modular<FactorizerVarMod<T>> cur = a;
        cur = power(cur, d);
        if (cur == 1) {
            continue;
        }
        bool witness = true;
        for (int r = 0; r < s; r++) {
            if (cur == n - 1) {
                witness = false;
                break;
            }
            cur *= cur;
        }
        if (witness) {
            return false;
        }
    }
    return true;
}

bool IsPrime(int64_t n) {
    return IsPrime(n, {2, 325, 9375, 28178, 450775, 9780504, 1795265022});
}

bool IsPrime(int32_t n) {
    return IsPrime(n, {2, 7, 61});
}

// but if you really need uint64_t version...
/*
bool IsPrime(uint64_t n) {
    if (n < 2) {
        return false;
    }
    std::vector<uint32_t> small_primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    for (uint32_t x : small_primes) {
        if (n == x) {
            return true;
        }
        if (n % x == 0) {
            return false;
        }
    }
    if (n < 31 * 31) {
        return true;
    }
    uint32_t s = __builtin_ctzll(n - 1);
    uint64_t d = (n - 1) >> s;
    function<bool(uint64_t)> witness = [&n, &s, &d](uint64_t a) {
        uint64_t cur = 1, p = d;
        while (p > 0) {
            if (p & 1) {
                cur = (__uint128_t) cur * a % n;
            }
            a = (__uint128_t) a * a % n;
            p >>= 1;
        }
        if (cur == 1) {
            return false;
        }
        for (uint32_t r = 0; r < s; r++) {
            if (cur == n - 1) {
                return false;
            }
            cur = (__uint128_t) cur * cur % n;
        }
        return true;
    };
    std::vector<uint64_t> bases_64bit = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    for (uint64_t a : bases_64bit) {
        if (a % n == 0) {
            return true;
        }
        if (witness(a)) {
            return false;
        }
    }
    return true;
}
*/

std::vector<int> least = {0, 1};
std::vector<int> primes;
int precalculated = 1;

void RunLinearSieve(int n) {
    n = std::max(n, 1);
    least.assign(n + 1, 0);
    primes.clear();
    for (int i = 2; i <= n; i++) {
        if (least[i] == 0) {
            least[i] = i;
            primes.push_back(i);
        }
        for (int x : primes) {
            if (x > least[i] || i * x > n) {
                break;
            }
            least[i * x] = x;
        }
    }
    precalculated = n;
}

void RunSlowSieve(int n) {
    n = std::max(n, 1);
    least.assign(n + 1, 0);
    for (int i = 2; i * i <= n; i++) {
        if (least[i] == 0) {
            for (int j = i * i; j <= n; j += i) {
                if (least[j] == 0) {
                    least[j] = i;
                }
            }
        }
    }
    primes.clear();
    for (int i = 2; i <= n; i++) {
        if (least[i] == 0) {
            least[i] = i;
            primes.push_back(i);
        }
    }
    precalculated = n;
}

void RunSieve(int n) {
    RunLinearSieve(n);
}

template <typename T>
std::vector<std::pair<T, int>> MergeFactors(const std::vector<std::pair<T, int>>& a, const std::vector<std::pair<T, int>>& b) {
    std::vector<std::pair<T, int>> c;
    int i = 0;
    int j = 0;
    while (i < (int) a.size() || j < (int) b.size()) {
        if (i < (int) a.size() && j < (int) b.size() && a[i].first == b[j].first) {
            c.emplace_back(a[i].first, a[i].second + b[j].second);
            ++i;
            ++j;
            continue;
        }
        if (j == (int) b.size() || (i < (int) a.size() && a[i].first < b[j].first)) {
            c.push_back(a[i++]);
        } else {
            c.push_back(b[j++]);
        }
    }
    return c;
}

template <typename T>
std::vector<std::pair<T, int>> RhoC(const T& n, const T& c) {
    if (n <= 1) {
        return {};
    }
    if ((n & 1) == 0) {
        return MergeFactors({{2, 1}}, RhoC(n / 2, c));
    }
    if (IsPrime(n)) {
        return {{n, 1}};
    }
    FactorizerVarMod<T>::value = n;
    Modular<FactorizerVarMod<T>> x = 2;
    Modular<FactorizerVarMod<T>> saved = 2;
    T power = 1;
    T lam = 1;
    while (true) {
        x = x * x + c;
        T g = __gcd((x - saved)(), n);
        if (g != 1) {
            return MergeFactors(RhoC(g, c + 1), RhoC(n / g, c + 1));
        }
        if (power == lam) {
            saved = x;
            power <<= 1;
            lam = 0;
        }
        lam++;
    }
    return {};
}

template <typename T>
std::vector<std::pair<T, int>> Rho(const T& n) {
    return RhoC(n, static_cast<T>(1));
}

template <typename T>
std::vector<std::pair<T, int>> Factorize(T x) {
    if (x <= 1) {
        return {};
    }
    if (x <= precalculated) {
        std::vector<std::pair<T, int>> ret;
        while (x > 1) {
            if (!ret.empty() && ret.back().first == least[x]) {
                ret.back().second++;
            } else {
                ret.emplace_back(least[x], 1);
            }
            x /= least[x];
        }
        return ret;
    }
    if (x <= static_cast<int64_t>(precalculated) * precalculated) {
        std::vector<std::pair<T, int>> ret;
        if (!IsPrime(x)) {
            for (T i : primes) {
                T t = x / i;
                if (i > t) {
                    break;
                }
                if (x == t * i) {
                    int cnt = 0;
                    while (x % i == 0) {
                        x /= i;
                        cnt++;
                    }
                    ret.emplace_back(i, cnt);
                    if (IsPrime(x)) {
                        break;
                    }
                }
            }
        }
        if (x > 1) {
            ret.emplace_back(x, 1);
        }
        return ret;
```

```cpp
    }
    return Rho(x);
}

template <typename T>
std::vector<T> BuildDivisorsFromFactors(const std::vector<std::pair<T, int>>& factors) {
    std::vector<T> divisors = {1};
    for (auto& p : factors) {
        int sz = (int) divisors.size();
        for (int i = 0; i < sz; i++) {
            T cur = divisors[i];
            for (int j = 0; j < p.second; j++) {
                cur *= p.first;
                divisors.push_back(cur);
            }
        }
    }
    sort(divisors.begin(), divisors.end());
    return divisors;
}

}  // namespace factorizer
```

6.5 comb

```cpp
class Comb {
public :
    const int n;;
    std::vector<Z> fac, inv, ifac;
    Comb(int n) : n(n), fac(n), inv(n), ifac(n) {
        fac[0] = fac[1] = inv[0] = inv[1] = ifac[0] = ifac[1] = 1;
        for(int i = 2;i < n; i++) {
            fac[i] = fac[i - 1] * i;
            inv[i] = (P - P / i) * inv[P % i];
            ifac[i] = ifac[i - 1] * inv[i];
        }
    }
    Z C(int n, int m) {
        if(m < 0 || n < 0 || m > n) return 0;
        return fac[n] * ifac[m] * ifac[n - m];
    }
    Z Lucas(long long m, long long n) { return n ? Lucas(m / P, n / P) * C(m % P, n % P) : 1;}
};
```

6.6 算术基本定理

```cpp
ll get_Count(ll n) {
    ll ans = 1;
    for(int i = 2;i * i <= n; i++) {
        if(n % i == 0) {
            int a = 0;
            while(n % i == 0) {
                a++;
                n /= i;
            }
            ans *= (a + 1);
        }
    }
    if(n > 1)  ans *= 2;
    return ans;
}

ll get_Sum(ll n) {
    ll ans = 1;
    for(int i = 2;i * i <= n; i++) {
        if(n % i == 0) {
            ll a = 1;
            while(n % i == 0) {
                n /= i;
                a *= i;
            }
            ans = ans * (a * i - 1) / (i - 1);
        }
    }
    if(n > 1) ans *= (n + 1);
    return ans;
}
```

6.7 筛 phi

```
int is_prime[N], prime[N], cnt, phi[N];
void makePhi() {
    phi[1] = 1, cnt = 0;
    for (int i = 2; i < N; i++) {
        if (!is_prime[i]) prime[++cnt] = i, phi[i] = i - 1;
        for (int j = 1; j <= cnt && i * prime[j] < N; j++) {
            is_prime[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else phi[i * prime[j]] = phi[i] * phi[prime[j]];
        }
    }
}
```

6.8 筛 mobius

```
const int N = 1e5 + 10;
bool is_prime[N];
int prime[N], mu[N], cnt;

void makeMobius() {
    mu[1] = 1; is_prime[0] = is_prime[1] = true;
    for(int i = 2;i < N; i++) {
        if (!is_prime[i]) {
            mu[i] = -1;
            prime[++cnt] = i;
        }
        for (int j = 1; j <= cnt && i * prime[j] < N; j++) {
            is_prime[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            }
            mu[i * prime[j]] = -mu[i];
        }
    }
}
```

6.9 筛积性函数

```
//只需要计算f(p ^ k)即可
//其余的都可以通过积性函数的性质来计算

int vis[N], prime[N], num;
int f[N], low[N];

void makeF(int siz) {//f为积性函数
    num = 0, low[1] = f[1] = 1;
    for (int i = 2; i <= siz; i++) {
        if (!vis[i]) prime[++num] = i, low[i] = i, f[i] = ...;//这里是f(p)的答案
        for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                low[i * prime[j]] = low[i] * prime[j];
                if(low[i] == i) {//i = prime[j] ^ k
                    //只需要这里算一下
                    //考虑 p ^ 1 , p ^ 2, p ^ 3...
                }
                else f[i * prime[j]] = 1ll * f[i / low[i]] * f[prime[j] * low[i]] % mod
    ;
                break;
            }
            low[i * prime[j]] = prime[j];
            f[i * prime[j]] = 1ll * f[i] * f[prime[j]] % mod;
        }
    }
}
```

6.10 欧拉函数

```
// 求解单个正整数的欧拉函数
int Get_phi(int n) {
    int ans = n;
    for(int i = 2;i * i <= n; i++) {
        if(n % i == 0) {
            ans = ans - ans / i;
            while(n % i == 0)
                n /= i;
        }
    }
    if(n > 1)
        ans = ans - ans / n;
    return ans;
}

// 埃拉托斯特尼筛求欧拉函数
int phi[10005];
```

```

void Euler_sieve(int n) {
    phi[1] = 1;
    for(int i = 2;i <= n; i++) {
        if(!phi[i]) {
            for(int j = i;j <= n; j += i)    {
                if(!phi[j])
                    phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
}

// 欧拉筛求欧拉函数

const int N = 5e6 + 10;
bool is_prime[N];
int prime[N], phi[N], tot;

void Euler() {
    phi[1] = 1; is_prime[1] = true;
    for(int i = 2;i < N; i++){
        if(!is_prime[i]) {
            phi[i] = i - 1;
            prime[++tot] = i;
        }
        for(int j = 1;j <= tot && i * prime[j] < N; j++){
            is_prime[i * prime[j]] = true;
            if(i % prime[j]) {
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            }
            else{
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
        }
    }
}
```

6.11 原根

```
typedef long long ll;

vector<ll> YG;
ll p, n; // p是模数，n是p的欧拉函数值

ll gcd(ll a, ll b) {
    return b ? gcd(b, a % b) : a;
}

ll quick_pow(ll a, ll b, ll p) ;

ll phi(ll n) {
    ll ans = n;
    for(int i = 2;i * i <= n; i++) {
        if(n % i == 0) {
            ans = ans - ans / i;
            while(n % i == 0) {
                n /= i;
            }
        }
    }
    if(n > 1)
        ans = ans - ans / n;
    return ans;
}

vector<ll> PrimeFac(ll n) { // n的素因子
    vector<ll> fac;
    fac.clear();
    for(ll i = 2;i * i <= n; i++) {
        if(n % i == 0) {
            fac.push_back(i);
            while(n % i == 0)
                n /= i;
        }
    }
    if(n > 1)
        fac.push_back(n);
    return fac;
}

bool is_Protogen(ll p) { // 原根p = 2、4、p^k、2*p^k(p为非2的质数，k为任意数)
    if(p == 2 || p == 4) return true;
    if(p <= 1 || p % 4 == 0) return false;
    ll num = 0;
    while(p % 2 == 0) // 2的倍数先筛掉
        p /= 2;
    for(int i = 3;i * i <= p; i++) { // p只能是一个非2的素数的倍数构成，否则没有原根
        if(p % i == 0) {
            num++;
            while(p % i == 0)
                p /= i;
        }
    }
    if(p > 1) num++;
    if(num == 1) return true;
    return false;
}

ll Protogen(ll p) {
    if(!is_Protogen(p)) // 先判断是否存在原根
        return -1;
    n = phi(p);
    if(p == 2) return 1;
    if(p == 3) return 2;
    if(p == 4) return 3;
    vector<ll> fac = PrimeFac(n); // f(p)的素因子
    for(int i = 2;i <= p - 1; i++) {
        if(gcd(i, p) != 1) // n是模p的欧拉函数值，i要和n互质
            continue;
        bool flag = true;
        for(ll j = 0;j < fac.size(); j++) {
            if(quick_pow(i, n / fac[j] , p) == 1)
                flag = 0;
        }
        if(flag) // i就是原根
            return i;
    }
    return -1;
}

void Sum_Protogen(ll k) { // 找出n的所有原根
    YG.push_back(k);
    for(int i = 2;i < n; i++) {
        if(gcd(i, n) == 1) // i要与f(n)互质
            YG.push_back(quick_pow(k, i, p));
    }
}

int main() {
    cin >> p;
    ll k = Protogen(p); // p的原根
    cout << k << endl;
    Sum_Protogen(k);
    for(int i = 0;i < YG.size(); i++) {
        cout << YG[i] << " ";
    }
    cout << endl;
    return 0;
}
```

6.12 原根表

```
mod                                         原根
r*2^k+1   r               k             g
3   1   1   2
5   1   2   2
17  1   4   3
97  3   5   5
193 3   6   5
257 1   8   3
7681    15  9   17
12289   3   12  11
40961   5   13  3
65537   1   16  3
786433  3   18  10
5767169 11  19  3
7340033 7   20  3
23068673    11  21  3
104857601   25  22  3
167772161   5   25  3
469762049   7   26  3
998244353   119 23  3   这个数常用
1004535809  479 21  3   加起来不会爆int
2013265921  15  27  31
2281701377  17  27  3    这个数平方刚好不会爆ll
3221225473  3   30  5
75161927681 35  31  3
77309411329 9   33  7
206158430209    3   36  22
2061584302081   15  37  7
2748779069441   5   39  3
6597069766657   3   41  5
```

```
39582418599937  9   42  5
79164837199873  9   43  5
263882790666241 15  44  7
1231453023109121    35  45  3
1337006139375617    19  46  3
3799912185593857    27  47  5
4222124650659841    15  48  19
7881299347898369    7   50  6
31525197391593473   7   52  3
180143985094819841  5   55  6
1945555039024054273 27  56  5
4179340454199820289 29  57  3
```

6.13 阶乘逆元

```
const int N = 5e6 + 10;
const ll mod = 1e9 + 7;

ll F[N], invn[N], invF[N];

void Init() {
    F[0] = F[1] = invn[0] = invn[1] = invF[0] = invF[1] = 1;
    for(int i = 2;i < N; i++){
        F[i] = F[i - 1] * i % mod;
        invn[i] = (mod - mod / i) * invn[mod % i] % mod;
        invF[i] = invF[i - 1] * invn[i] % mod;
    }
}
```

6.14 常见积性函数

```
//phi
//phi[i * j] = phi[i] * phi[j] * gcd(i, j) / phi[gcd(i, j)]

// d
// d[i * j] = \sum_{x|i} * \sum_{y|j} * [gcd(x, y) = 1]
```

6.15 Miller-Rabin

```
// 二次探测定理：对素数p,满足x^2≡1(modp)的小于p的正整数解x只有1或p-1.

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 7;
const int times = 10;

ll ksc(ll a, ll b, ll mod)  {
    ll ans = 0;
    while(b > 0) {
        if(b & 1) {
            ans = (ans + a) % mod;
        }
        a = (a << 1) % mod;
        b >>= 1;
    }
    return ans;
}

ll quick_pow(ll a, ll b, ll mod) {
    ll ans = 1, base = a;
    while(b != 0) {
        if(b & 1) {
            ans = ans * base % mod;
        }
        base = base * base % mod;
        b >>= 1;
    }
    return ans;
}

bool Miller_Pabin(ll n)//Miller测试的主体结构
{
    if(n < 2) return false;
    if(n == 2) return true;
    if(n & 1 == 0) return false;//对于偶数的优化
    ll k = 0,u = n - 1;//p为Miller测试的k，u为Miller测试的m

    while(u & 1 == 0){ // 把x拆成u*2^k
        u >>= 1;
        k++;
    }
    srand(time(NULL));

    ll x, pre; // pre为上次探测的x的值

    for(int i = 1;i <= times; i++) {
        x = rand() % (n - 1) + 1;
        x = quick_pow(x, u, n); // 先求出x^u(mod n)
        pre = x;
        for(int j = 1;j <= k; j++) {
            x = ksc(x, x, n);
            if(x == 1 && pre != 1 && pre != n - 1)
                return false;
            pre = x;
        }
        if(x != -1)
        return false;
    }
    return true;
}

int main() {
    ll n; cin >> n;
    cout << (Miller_Pabin(n) ? "Prime" : "Not a Prime") << endl;
}
```

6.16 quadraticResidue

```
typedef long long ll;

typedef struct{
    ll x, y; // 把求出来的w作为虚部，则为a + bw
}num;
```

```

ll quick_pow(ll a, ll b, ll p) {
    ll ans = 1;
    while(b) {
        if(b & 1) ans = ans * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return ans % p;
}


num num_mul(num a, num b, ll w, ll p) {// 复数乘法
    num ans = {0, 0};
    ans.x = (a.x * b.x % p + a.y * b.y % p * w % p + p) % p;
    ans.y = (a.x * b.y % p + a.y * b.x % p + p) % p;
    return ans;
}

ll num_pow(num a, ll b, ll w, ll p) { // 复数快速幂
    num ans = {1, 0};
    while(b) {
        if(b & 1)
            ans = num_mul(ans, a, w, p);
        a = num_mul(a, a, w, p);
        b >>= 1;
    }
    return ans.x % p;
}

ll legendre(ll a, ll p) { // 勒让德符号 = {1, -1, 0}
    return quick_pow(a, (p - 1) >> 1, p);
}

ll Cipolla(ll n, ll p) {// 输入a和p，是否存在x使得x^2 = a (mod p)，存在二次剩余返回x，存在二次非剩余返回-1      注意：p是奇质数
    n %= p;
    if(n == 0)
        return 0;
    if(p == 2)
        return 1;
    if(legendre(n, p) + 1 == p) // 二次非剩余
        return -1;

    ll a, w;

    while(true) {// 找出a，求出w，随机成功的概率是50%，所以数学期望是2
        a = rand() % p;
        w = ((a * a - n) % p + p) % p;
        if(legendre(w, p) + 1 == p) // 找到w，非二次剩余条件
            break;
    }
    num x = {a, 1};
    return num_pow(x, (p + 1) >> 1, w, p) % p; // 计算x,一个解是x，另一个解是p-x，这里的w其实要开方，但是由拉格朗日定理可知虚部为0，所以最终答案就是对x的实部用快速幂求解
}

int main()
{
    ll n, p;
    cin >> n >> p;
    srand((unsigned)time(NULL));
    cout << Cipolla(n, p) << endl;
    return 0;
}
```

6.17 bags

```
// 求解a^x = b (mod c)，要求gcd(a, c) = 1， 不要求p为素数，x的范围是0 <= x <= p-1

template <typename T>
struct Hash {
    int n;
    int cnt;
    std::vector<int> head, next, hash, id;
    Hash(int _n) : n(_n), head(_n, -1), next(_n), id(_n), hash(_n), cnt(0) {}

    void insert(T x, T y) {
        T k = x % n;
        hash[cnt] = x;
        id[cnt] = y;
        next[cnt] = head[k];
        head[k] = cnt++;
    }

    T query(T x) {
        for(int i = head[x % n]; i != -1 ; i = next[i]){
            if(hash[i] == x)
                return id[i];
        }
        return -1;
    }
};

template <typename T>
T bsgs(T& a, T& b, T& c) {
    a %= c; b %= c;
    int cnt = 1;
    if (b == 1) return 0;
    Hash<long long> hs(100005);
    T m = std::ceil(sqrt((double) c));
    T x = 1, p = 1;
    for (T j = 0; j < m; j++, p = p * a % c) {
        hs.insert(p * b % c, j);
    }
    for (T i = 1, j; i <= m; i++) {
        x = x * p % c;
        if ((j = hs.query(x)) != -1) {
            return i * m - j;
        }
    }
    return -1;
}
```

6.18 EX-BSGS

```
// a和c不互质
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

ll gcd(ll a, ll b) {
    return b ? gcd(b, a % b) : a;
}

template <typename T>
T ex_bsgs(T& a, T& b, T& c) {
    a %= c; b %= c;
    if (b == 1) return 0;
    T k = 0, tmp = 1, d;
    while (true) {
        d = __gcd(a, c);
        if (d == 1) {
            break ;
        }
        if (b % d) {
            return -1;
        }
        b /= d; c /= d;
        tmp = tmp * (a / d) % c;
        k++;
        if (tmp == b) {
            return k;
        }
    }
    std::unordered_map<T, T> mp;
    T m = std::ceil(sqrt((double) c));
    T x = 1, p = 1;
    for(T j = 0;j < m; j++, p = p * a % c) {
        mp[p * b % c] = j;
    }
    x = tmp % c;
    for(T i = 1 ;i <= m; i++) { // 枚举a^im
        x = x * p % c;
        if(mp[x]) {
            return k + i * m - mp[x];
        }
    }
    return -1;
}
```

6.19 CRT

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
ll m[10005], a[10005], n; //a是余数, b是模数

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

ll INV(ll a, ll mod) {
    ll x, y;
    exgcd(a , mod , x , y);
    x = (x % mod + mod) % mod;
    return x;
}

ll CRT() {
    ll ans = 0, M = 1;
    for(ll i = 1;i <= n; i++) {
        M *= m[i]; // M是所有除数的乘积
    }
    for(ll i = 1;i <= n; i++) {
        ll mm = M / m[i];
        ll ret = INV(mm , m[i]); // 先求逆元
        ans = (ans + a[i] * mm % M * ret % M) % M;
/*
ans = (ans + quick_mul(quick_mul(m , ret , M) , b[i] , M)) % M;
利用快速乘防止爆longlong
*/
    }
    return (ans + M) % M;
}

int main() {
    ll ans = 0;
    scanf("%lld",&n);
    for(ll i = 1;i <= n; i++) {
        scanf("%lld%lld",&m[i],&a[i]);
        a[i] = (a[i] % m[i] + m[i]) % m[i];// 防止b[i]为负
    }
    ans = CRT(); // 精髓
    printf("%lld",ans);
    return 0;
}
```

6.20 EX-CRT

```
#include <iostream>

using namespace std;

typedef long long ll;

ll c[100005], m[100005], n;

ll ksc(ll a, ll b, ll mod) {
    ll ans = 0;
    while(b > 0) {
        if(b & 1) {
            ans = (ans + a) % mod;
        }
        a = (a << 1) % mod;
        b >>= 1;
    }
    return ans;
}

ll gcd(ll a, ll b) {
    return b ? gcd(b, a % b) : a;
}

ll ex_gcd(ll a, ll b, ll &x, ll &y) {
    ll res, t;
    if(!b) {
        x = 1;
        y = 0;
        return a;
    }
    res = ex_gcd(b, a % b, x, y);
    t = x;
    x = y;
    y = t - (a / b) * y;
    return res;
}

ll INV(ll a, ll mod) {
    ll x, y;
    ll d = ex_gcd(a, mod, x, y);
    return d ? (x % mod + mod) % mod : -1;
}

ll EX_CRT() {
    ll x, y;
    ll ans = c[1];
    ll M = m[1];
    for(int i = 2; i <= n; i++) {
        ll C = ((c[i] - ans) % m[i] + m[i]) % m[i];
        ll T = ex_gcd(M, m[i], x, y);
        if((c[i] - ans) % T)
            return -1;
        x = ksc(x, C / T, m[i] / T);
        ans += M * x;
        M *= (m[i] / T);
        ans = (ans % M + M) % M;
    }
    return ans;
}

/*
ll EX_CRT() // 便于理解
{
    for(int i = 2;i <= n; i++)
    {
        ll M1 = m[i - 1], M2 = m[i], C1 = c[i - 1], C2 = c[i];
        ll T = gcd(M1, M2); // gcd(M1, M2)
        if((C2 - C1) % T) // 无解
            return -1;
        m[i] = (M1 * M2) / T; // 合并后新同余方程的模
        c[i] = INV(M1 / T, M2 / T) * (C2 - C1) / T % (M2 / T) * M1 + C1; // 可快速乘优化
        c[i] = (c[i] % m[i] + m[i]) % m[i]; // 合并后新同余方程的余
    }
    return c[n];
}
*/

int main()
{
    cin >> n;
    for(int i = 1;i <= n; i++)
    cin >> c[i] >> m[i];
    cout << EX_CRT() << endl;
}
```

6.21 EX-Lucas

```
// p不为质数，利用中国剩余定理结合求解
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e5 + 10;

ll quick_pow(ll a, ll b, ll P) {
    ll ans = 1;
    while(b) {
        if(b & 1)
            ans = ans * a % P;
        a = a * a % P;
        b >>= 1;
    }
    return ans % P;
}

ll ex_gcd(ll a, ll b, ll &x, ll &y) {
    ll res, t;
    if(!b) {
        x = 1;
        y = 0;
        return a;
    }
    res = ex_gcd(b, a % b, x, y);
    t = x;
    x = y;
    y = t - (a / b) * y;
    return res;
}

ll INV(ll a, ll mod) {
    ll x, y;
    ll d = ex_gcd(a, mod, x, y);
    return d ? (x % mod + mod) % mod : -1;
}

ll fac(ll n, ll P, ll pk) {// 阶乘除去质因子后模质数幂 （n / p^a） % pk
    if(!n) return 1;
    ll ans = 1;
    for(int i = 1;i < pk; i++) {// 第三部分：n!与p互质的乘积
```

```
        if(i % P)
            ans = ans * i % pk;
    }
    ans = quick_pow(ans, n / pk, pk) % pk; // 第三部分：n!与p互质的乘积,ans循环的次数为n/pk
    for(int i = 1;i <= n % pk; i++) {// 第四部分：循环过后n!剩下的部分
        if(i % P) ans = ans * i % pk;
    }
    return ans * fac(n / P, P, pk) % pk;  // 第一部分，p的幂，个数为n/p;   第二部分：(n/p)!
}

ll C(ll m, ll n, ll P, ll pk) {// 组合数模质数幂
    if(n < 0 || m < 0 || n > m) return 0;
    ll f1 = fac(m, P, pk), f2 = fac(n, P, pk), f3 = fac(m - n, P, pk), tmp = 0; // tmp = pk1 - pk2 - pk3
    for(ll i = m; i ; i /= P)     tmp += i / P;
    for(ll i = n; i ; i /= P)     tmp -= i / P;
    for(ll i = m - n; i ; i /= P) tmp -= i / P;
    return f1 * INV(f2, pk) % pk * INV(f3, pk) * quick_pow(P, tmp, pk) % pk;
}

ll p[N], a[N];
int cnt;

ll CRT() {
    ll M = 1, ans = 0;
    for(int i = 1;i <= cnt; i++)  M *= p[i];
    for(int i = 1;i <= cnt; i++) {
        ll m = M / p[i];
        ans = (ans + a[i] * m % M * INV(m, p[i]) % M) % M;
    }
    return (ans % M + M) % M;
}

ll EX_Lucas(ll m, ll n, ll P) {
    for(int i = 2;i * i <= P; i++) {
        if(P % i == 0) {
            ll tmp = 1;
            while(P % i == 0) {
                tmp *= i;
                P /= i;
            }
            p[++cnt] = tmp;
            a[cnt] = C(m, n, i, tmp);
        }
    }
    if(P > 1) {
        p[++cnt] = P;
        a[cnt] = C(m, n, P, P);
    }
    return CRT();
}
int main() {
    ll m, n, P;
    cin >> m >> n >> P;
    cnt = 0;
    cout << EX_Lucas(m, n, P) << endl;
}
```

6.22 min25

```
typedef long long ll;

const int N = 1e5 + 10;


namespace Min25 {
    int prime[N], id1[N], id2[N], flag[N], ncnt, m;

    ll g[N], sum[N], a[N], T, n;

    inline int ID(ll x) {
        return x <= T ? id1[x] : id2[n / x];
    }

    inline ll calc(ll x) {
        return x * (x + 1) / 2 - 1;
    }

    inline ll f(ll x) {
        return x;
    }

    inline void init() {
        ncnt = 0, m = 0;
        T = sqrt(n + 0.5);
        for (int i = 2; i <= T; i++) {
            if (!flag[i]) prime[++ncnt] = i, sum[ncnt] = sum[ncnt - 1] + i;
            for (int j = 1; j <= ncnt && i * prime[j] <= T; j++) {
                flag[i * prime[j]] = 1;
                if (i % prime[j] == 0) break;
            }
        }
        for (ll l = 1; l <= n; l = n / (n / l) + 1) {
            a[++m] = n / l;
            if (a[m] <= T) id1[a[m]] = m; else id2[n / a[m]] = m;
            g[m] = calc(a[m]);
        }
        for (int i = 1; i <= ncnt; i++)
            for (int j = 1; j <= m && (ll)prime[i] * prime[i] <= a[j]; j++)
                g[j] = g[j] - (ll)prime[i] * (g[ID(a[j] / prime[i])] - sum[i - 1]);
    }

    inline ll Solve(ll x) {
        if (x <= 1) return x;
        return n = x, init(), g[ID(n)];
    }

}
```

6.23 BM

```
typedef long long ll;
const ll mod = 1e9 + 7;

typedef vector<ll> VI;

```

```
ll quick_pow(ll a, ll b) ;

namespace linear_seq {
    const ll N = 1e5 + 10;
    ll res[N], base[N], _c[N], _md[N];

    vector<ll> Md;
    void mul(ll *a, ll *b, ll k) {
        for (ll i = 0; i < 2 * k; i++)
            _c[i] = 0;
        for (ll i = 0; i < k; i++) {
            if (a[i]) {
                for (int j = 0; j < k; j++) {
                    _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
                }
            }
        }
        for (ll i = 2 * k - 1; i >= k; i--) {
            if (_c[i]) {
                for (ll j = 0; j < Md.size(); j++) {
                    _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j]]) % mod;
                }
            }
        }
        for (ll i = 0; i < k; i++)
            a[i] = _c[i];
    }

    ll solve(ll n, VI a, VI b) {
        // a 系数 b 初值 b[n + 1] = a[0] * b[n] + ...
        // cout << b.size() << endl;
        ll ans = 0, pnt = 0;
        ll k = a.size();
        assert(a.size() == b.size());
        for (ll i = 0; i < k; i++)
            _md[k - i - 1] = -a[i];
        _md[k] = 1;
        Md.clear();
        for (ll i = 0; i < k; i++) {
            if (_md[i] != 0)
                Md.push_back(i);
        }
        for (ll i = 0; i < k; i++)
            res[i] = base[i] = 0;
        res[0] = 1;
        while ((1ll << pnt) <= n)
            pnt++;
        for (ll p = pnt; p >= 0; p--) {
            mul(res, res, k);
            if ((n >> p) & 1) {
                for (ll i = k - 1; i >= 0; i--)
                    res[i + 1] = res[i];
                res[0] = 0;
                for (ll i = 0; i < Md.size(); i++)
                    res[Md[i]] = (res[Md[i]] - res[k] * _md[Md[i]]) % mod;
            }
        }
        for (ll i = 0; i < k; i++)
            ans = (ans + res[i] * b[i]) % mod;
        return ans;
    }

    VI BM(VI s) {
        VI C(1, 1), B(1, 1);
        ll L = 0, m = 1, b = 1;
        for (ll n = 0; n < s.size(); n++) {
            ll d = 0;
            for (ll i = 0; i < L + 1; i++)
                d = (d + (ll)C[i] * s[n - i]) % mod;
            if (d == 0)
                m++;
            else if (2 * L <= n) {
                VI T = C;
                ll c = mod - d * quick_pow(b, mod - 2) % mod;
                while (C.size() < B.size() + m)
                    C.push_back(0);
                for (int i = 0; i < B.size(); i++)
                    C[i + m] = (C[i + m] + c * B[i]) % mod;
                L = n + 1 - L;
                B = T;
                b = d;
                m = 1;
            }
            else {
                ll c = mod - d * quick_pow(b, mod - 2) % mod;
                while (C.size() < B.size() + m)
                    C.push_back(0);
                for (ll i = 0; i < B.size(); i++)
                    C[i + m] = (C[i + m] + c * B[i]) % mod;
                m++;
            }
        }
        return C;
    }

    ll gao(VI a, ll n) {
        VI c = BM(a);
        c.erase(c.begin());
        for (ll i = 0; i < c.size(); i++)
            c[i] = (mod - c[i]) % mod;
        return solve(n, c, VI(a.begin(), a.begin() + c.size()));
    }
}

void solve() {
    int n;
    while (~scanf("%d", &n)) {
        VI v = VI{1,2,4,7,13,24};
            printf("%d\n", linear_seq::gao(v, n - 1));
    }
}
```

6.24 duSieve

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

using namespace std;
```

```cpp

typedef long long ll;
const int N = 1e6 + 10;

unordered_map<int, ll> smu, sphi;
bool isPrime[N];
int prime[N], num;
ll mu[N], phi[N];

void makeMobiusAndEuler(int siz) {
    mu[1] = phi[1] = 1;
    for (int i = 2; i <= siz; i++) {
        if (!isPrime[i]) prime[++num] = i, mu[i] = -1, phi[i] = i - 1;
        for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
            isPrime[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else {
                phi[i * prime[j]] = phi[prime[j]] * phi[i];
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
    for (int i = 1; i <= siz; i++) mu[i] += mu[i - 1], phi[i] += phi[i - 1];
}

ll getSmu(int n) {
    if (n < N) return mu[n];
    if (smu[n]) return smu[n];
    ll res = 1;
    for (unsigned int l = 2, r = 0; l <= n; l = r + 1) {
        r = n / (n / l);
        res -= 1ll * (r - l + 1) * getSmu(n / l);
    }
    return smu[n] = res;
}

ll getSphi(int n) {
    if (n < N) return phi[n];
    if (sphi[n]) return sphi[n];
    ll res = 1ll * n * (n + 1) / 2;
    for (unsigned int l = 2, r = 0; l <= n; l = r + 1) {
        r = n / (n / l);
        res -= 1ll * (r - l + 1) * getSphi(n / l);
    }
    return sphi[n] = res;
}
```

6.25 反演相关

```cpp
/*
莫比乌斯反演
g[n] = \sum_{d | n} f[d]
f[d] = \sum_{d | n} g[d] * mu[n / d]
二项式反演
```

```
g[n] = \sum{i = 1}^{n} C(n, i) * f[i]
f[n] = \sum{i = 1}^{n} C(n, i) * g[i] * (-1)^{n - i}
子集反演
f(S) = \sum_{T \belong S} g(T)
g(S) = \sum_{T \belong S} f(T) * (-1) ^ {|S| - |T|}
*/
```

6.26 simpson

```
// 求一个函数在一个区间上的数值积分

double f(double x) { // 题目中要求的辛普森积分函数，这里简单写一下f(x)=x*x
    return x * x;
}

double Simpson(double a, double b) {
    double mid = (a + b) / 2.0;
    return (b - a) *(f(a) + f(b) + 4.0 * f(mid)) / 6.0;
}

double DFS(double a, double b, double eps)
{
    double mid = (a + b) / 2.0;
    double SA = Simpson(a, mid), SM = Simpson(a, b), SB = Simpson(mid, b);
    if(fabs(SA + SB - SM) <= 15.0 * eps)
        return SA + SB + (SA + SB - SM) / 15.0;
    return DFS(a, mid, eps / 2.0) + DFS(mid, b, eps / 2.0);
}

// 求一个函数在0~无穷的上的数值积分，若收敛输出答案，若发散输出orz
```

6.27 Bell

```

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 20;

ll S2[N][N];
ll B[N];

void Stirling2() {
    S2[0][0] = 1;

    for(int i = 1;i < N; i++) {
        for(int j = 1;j <= i; j++) {
            S2[i][j] = S2[i - 1][j - 1] + j * S2[i - 1][j];
        }
    }

}

// 根据第二类斯特林数

```

```
void Bell1() {

    for(int i = 0;i < N; i++) {
        for(int j = 0;j <= i; j++) {
            B[i] += S2[i][j];
        }
    }
}

// Bell三角形递推

ll b[N][N];

void Bell2() {
    b[1][1] = 1;
    for(int i = 2;i < N; i++) {
        b[i][1] = b[i - 1][i - 1];

        for(int j = 2;j < N; j++) {
            b[i][j] = b[i][j - 1] + b[i - 1][j - 1];
        }
    }
}

// 自身递推

ll fac[N];

ll C(ll m, ll n) {
    return fac[m] / (fac[n] * fac[m - n]);
}

void Bell3() {

    fac[1] = 1;
    for(int i = 2;i < N; i++)
        fac[i] = fac[i - 1] * i;

    B[0] = 1;

    for(int i = 1;i < N; i++) {
        for(int k = 0;k <= i; k++) {
            B[i] += C(i, k) * B[k];
        }
    }
}
```

6.28 Catalan

```

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int N = 1e5 + 10;

int C[N];
```

```

// 线性递推

void Calc1() {
    C[0] = 1;
    for(int i = 1;i < N; i++) {
        C[i] = C[i - 1] * (4 * i - 2) / (i + 1);
    }
}

// 组合数求解

int f[N];

void fac() {
    f[0] = 1;
    for(int i = 1;i < N; i++) {
        f[i] = f[i - 1] * i;
    }
}

void Calc2(int n) {
    C[n] = f[2 * n] / f[n + 1];
}

// 多项式求解

void Calc3(int n) {
    if(n == 1)
        C[n] = 1;

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

6.29 Lucas

```
// mod一定为质数

namespace Comb {
    ll mod;
    const int N = 1e6 + 10;
    ll F[N], invF[N], inv[N];

    void init() {
        F[0] = F[1] = invF[0] = invF[1] = inv[0] = inv[1] = 1;
        for (int i = 2; i < N; i++) {
            F[i] = F[i - 1] * i % mod;
            inv[i] = (mod - mod / i) * inv[mod % i] % mod;
            invF[i] = invF[i - 1] * inv[i] % mod;
        }
    }

    ll C(ll m, ll n) {
        if (m < 0 || n < 0 || n > m) return 0;
        ll ans = F[m];
        ans = ans * invF[n] % mod;
        ans = ans * invF[m - n] % mod;
        return ans;
    }

    ll Lucas(ll m, ll n) {
        return n ? Lucas(m / mod, n / mod) * C(m % mod, n % mod) % mod : 1;
    }

}

// Comb::Lucas(m, n)
```

6.30 伯努利数

```
namespace BNL {
    const int N = 1e7 + 10, M = 1e6 + 10;
    struct Complex {
        double x, y;
        Complex(double a = 0, double b = 0): x(a), y(b) {}
        Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
        Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
        Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
        Complex conj() { return Complex(x, -y); }
    } w[N];

    int tr[N];
    ll F[N], G[N];

    ll quick_pow(ll a, ll b, ll p) {
        ll ans = 1;
        while(b) {
            if(b & 1) ans = ans * a % p;
            a = a * a % p;
            b >>= 1;
        }
        return ans % p;
    }

    void FFT(Complex *A, int len) {
        for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
        for (int i = 2, lyc = len >> 1; i <= len; i <<= 1, lyc >>= 1)
            for (int j = 0; j < len; j += i) {
                Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
                for (int k = 0; k < i >> 1; k++) {
                    Complex tmp = *r * *p;
                    *r = *l - tmp, *l = *l + tmp;
                    ++l, ++r, p += lyc;
                }
            }
    }

    inline void MTT(ll *x, ll *y, ll *z, int n) {
        int len = 1; while (len <= n) len <<= 1;
        for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
        for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI * i / len));

        for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
        static Complex a[N], b[N];
        static Complex dfta[N], dftb[N], dftc[N], dftd[N];

        for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
        for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
        FFT(a, len), FFT(b, len);
        for (int i = 0; i < len; i++) {
            int j = (len - i) & (len - 1);
            static Complex da, db, dc, dd;
            da = (a[i] + a[j].conj()) * Complex(0.5, 0);
            db = (a[i] - a[j].conj()) * Complex(0, -0.5);
            dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
            dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
            dfta[j] = da * dc;
            dftb[j] = da * dd;
            dftc[j] = db * dc;
            dftd[j] = db * dd;
        }
        for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
        for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
        FFT(a, len), FFT(b, len);
        for (int i = 0; i < len; i++) {
            int da = (ll)(a[i].x / len + 0.5) % mod;
            int db = (ll)(a[i].y / len + 0.5) % mod;
            int dc = (ll)(b[i].x / len + 0.5) % mod;
            int dd = (ll)(b[i].y / len + 0.5) % mod;
            z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
        }
    }

    int getLen(int n) {
        int len = 1; while (len < (n << 1)) len <<= 1;
        for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
        for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI * i / len));
        return len;
    }

    void Get_Inv(ll *f, ll *g, int n) {
        if(n == 1) { g[0] = quick_pow(f[0], mod - 2, mod); return ; }
        Get_Inv(f, g, (n + 1) >> 1);
        int len = getLen(n);
        static ll c[N];
        for(int i = 0;i < len; i++) c[i] = i < n ? f[i] : 0;
        MTT(c, g, c, len); MTT(c, g, c, len);
        for(int i = 0;i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
        for(int i = n;i < len; i++) g[i] = 0;
        for(int i = 0;i < len; i++) c[i] = 0;
    }

    ll ff[N], invff[N], inv[N];
    ll B[N];

    ll C(ll m, ll n) {
        if(m < 0 || n < 0 || n > m)
            return 0;
        ll ans = ff[m];
        ans = ans * invff[n] % mod;
        ans = ans * invff[m - n] % mod;
        return ans;
    }

    void init(int m) {
        ff[0] = ff[1] = inv[0] = inv[1] = invff[0] = invff[1] = 1;
        for(int i = 2;i < M; i++)
        {
            ff[i] = ff[i - 1] * i % mod;
            inv[i] = mod - (mod / i) * inv[mod % i] % mod;
            invff[i] = invff[i - 1] * inv[i] % mod;
        }

        for(int i = 0;i <= m + 10; i++) F[i] = invff[i + 1];
        Get_Inv(F, G, m + 10);
        for(int i = 0;i <= m + 10; i++) B[i] = G[i] * ff[i] % mod;
    }

    ll solve(ll n, int k) {
        init(k);
        ll ans = 0, prod = n % mod;
        for(int i = k; ~i ; i--) {
            ans = (ans + prod * B[i] % mod * C(k + 1, i) % mod) % mod;
            prod = prod * n % mod;
        }
        ans = ans * quick_pow(k + 1, mod - 2, mod) % mod;
        return ans;
    }
}

void solve() {
    ll n; int k; cin >> n >> k;
    cout << BNL::solve(n + 1, k) << endl;
}
```

6.31 步移-组合数前缀和

```
// S(n, m) = \sum_{i=0}^mC(n, i)

int x, y;
ll s;
ll S(int n, int m) {
    while(y < m) (s = s + C(x, ++y)) %= mod;
    while(y > m) (s = s - C(x, y--)) %= mod;
    while(x < n) (s = s * 2 - C(x++, y)) %= mod;
    while(x > n) (s = (s + C(--x, y)) * inv2) %= mod;
    return s;
}
```

6.32 康托展开

```
#include  <iostream>
#include <vector>
```

```cpp
#include <algorithm>

using namespace std;

typedef long long ll;
const int mod = 1e9 + 7;
const int N = 1e5 + 10;

ll fac[N];
int a[N]; // 排列，康托展开求解
int n;
ll x; // 逆康托展开求解

void Get_F() {
    fac[0] = 1;
    for(int i = 1;i < N; i++)
        fac[i] = fac[i - 1] * i % mod;
}

ll CanTor() {
    ll ans = 0;
    for(int i = 1;i <= n; i++) {
        ll smaller = 0;
        for(int j = i + 1;j <= n; j++) {
            if(a[j] < a[i])
                smaller++;
        }
        ans = (ans + fac[n - i] * smaller % mod) % mod;
    }
    return ans + 1;
}

void DeCantor() {
    vector<int> v; // 存放当前可选数
    vector<int> a; // 所求的排列组合序
    for(int i = 1;i <= n; i++) {
        v.push_back(i);
    }
    for(int i = n;i >= 1; i--) {
        int r = x % fac[i - 1];
        int t = x / fac[i - 1];
        x = r;
        sort(v.begin(), v.end());
        a.push_back(v[t]);
        v.erase(v.begin() + t);
    }
    for(int i = 0;i < a.size(); i++)
        cout << a[i] << " ";
    cout << endl;
}

// 线段树优化

const int N = 1000010;

ll fac[N];
int a[N]; // 排列，康托展开求解
int n;

```

```
struct SegmentTree {
    int ls, rs;
    int sum;
}t[N << 2];

int cnt, root;

void push_up(int u) {
    t[u].sum = (t[lc].sum + t[rc].sum) % mod;
}

void build(int &u, int l, int r) {
    if(!u) u = ++cnt;
    if(l == r) {
        t[u].sum = 1;
        return ;
    }
    build(lc, l, m);
    build(rc, m + 1, r);
    push_up(u);
}

void update(int &u, int l, int r, int k) {
    if(!u) u = ++cnt;
    if(l == r) {
        t[u].sum = 0;
        return ;
    }
    if(k <= m) update(lc, l, m, k);
    else update(rc, m + 1, r, k);
    push_up(u);
}

ll query(int u, int l, int r, int ql, int qr) {
    if(ql > qr) return 0;
    if(ql == l && qr == r) {
        return t[u].sum;
    }
    if(qr <= m) return query(lc, l, m, ql, qr) % mod;
    else if(ql > m) return query(rc, m + 1, r, ql, qr) % mod;
    else return (query(lc, l, m, ql, m) + query(rc, m + 1, r, m + 1, qr)) % mod;
}

void Get_F() {
    fac[0] = 1;
    for(int i = 1;i < N; i++)
        fac[i] = fac[i - 1] * i % mod;
}

void solve()
{
    Get_F();
    cin >> n;
    build(root, 1, n);
    ll ans = 0;
    for(int i = 1;i <= n; i++) {
        cin >> a[i];
        update(root, 1, n, a[i]);
        ans = (ans + query(root, 1, n, 1, a[i] - 1) * fac[n - i]) % mod;
    }
    cout << (ans + 1) % mod << endl;
}
```

6.33 模数非质数的组合

```
// 模数非质数情况下的组合问题
// one way, use CRT merge ans
// https://ac.nowcoder.com/discuss/655940?type=101&order=0&pos=2&page=1&channel=-1&source_id=discuss_tag_nctrack
// another way
// https://ac.nowcoder.com/acm/contest/view-submission?submissionId=47754622

#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const int N = 1e6 + 10;

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll res = exgcd(b, a % b, x, y);
    ll t = y;
    y = x - a / b * y;
    x = t;
    return res;
}

ll inv(ll a, ll b) {
    ll x = 0, y = 0;
    exgcd(a, b, x, y);
    return x = (x % b + b) % b;
}

//r[]为余数，m为模数，其中模数互质
//M = pi(mi), Mi = M / mi, invMi = Mi % mi
//ni满足是除了mi之外的倍数，且模mi为ri
//利用逆元性质，即ri * Mi * invMi = ri (mod mi)
//res = (sigma(ri * Mi * invMi)) % M

ll china(ll r[], ll m[], int n) {
    ll M = 1, res = 0;
    for (int i = 1; i <= n; i++) M *= m[i];
    for (int i = 1; i <= n; i++) {
        ll Mi = M / m[i], invMi = inv(Mi, m[i]);
        res = (res + r[i] * Mi % M * invMi % M) % M;
        //res = (res + mul(mul(r[i], Mi, M), invMi, M)) % M;按位乘
    }
    return (res % M + M) % M;
}

int f[N], g[N], F[N], G[N], invF[N];

int calc(int n, int p, int k) {
    ll mod = qpow(p, k, LONG_LONG_MAX);
    F[0] = 1, G[0] = 0;
    for (int i = 1; i <= n; i++) {
        g[i] = 0, f[i] = i;
        while (f[i] % p == 0) f[i] /= p, g[i]++;
        F[i] = 1ll * F[i - 1] * f[i] % mod;
        G[i] = G[i - 1] + g[i];
    }
    invF[n] = inv(F[n], mod);
    for (int i = n; i >= 1; i--) invF[i - 1] = 1ll * invF[i] * f[i] % mod;
    int ans = 0;
    for (int i = 0; i <= n / 2; i++) {
        int t = 1ll * F[n] * invF[n - 2 * i] % mod * invF[i] % mod * invF[i] % mod *
                qpow(p, G[n] - G[n - 2 * i] - 2 * G[i], LONG_LONG_MAX) % mod;
        ans = (ans + 1ll * t) % mod;
    }
    return ans;
}

ll r[20], m[20];

int main() {
#ifdef ACM_LOCAL
    freopen("input.in", "r", stdin);
    freopen("output.out", "w", stdout);
#endif
    int n, p;
    scanf("%d%d", &n, &p);
    int num = 0;
    for (int i = 2; i * i <= p; i++)
        if (p % i == 0) {
            int k = 0;
            m[++num] = 1;
            while (p % i == 0) p /= i, k++, m[num] *= i;
            r[num] = calc(n, i, k);
        }
    if (p > 1) {
        m[++num] = p;
        r[num] = calc(n, p, 1);
    }
    printf("%lld\n", china(r, m, num));

    return 0;
}
```

6.34 k 次最小置换复原

```

void solve() {
    int n; cin >> n;
    vector<int> a(n + 1), vis(n + 1);
    for(int i = 1;i <= n; i++) cin >> a[i];
    ll ans = 1;
    for(int i = 1;i <= n; i++) {
        if(!vis[i]) {
            int cnt = 0;
            int x = i;
            while(!vis[x]) {
                vis[x] = 1;
                cnt++;
                x = a[x];
            }
            ans = lcm(ans, cnt);
        }
    }
    cout << ans << endl;
}
```

6.35 Striling

```
typedef long long ll;
const int N = 20;

// 第一类斯特林数
ll S1[N][N];
void Stirling1() {
    S1[0][0] = 1;
    for(int i = 1;i < N; i++) {
        for(int j = 1;j <= i; j++) {
            S1[i][j] = S1[i - 1][j - 1] + (i - 1) * S1[i - 1][j];
        }
    }
}

// 第二类斯特林数
ll S2[N][N];
void Stirling2() {
    S2[0][0] = 1;
    for(int i = 1;i < N; i++) {
        for(int j = 1;j <= i; j++) {
            S2[i][j] = S2[i - 1][j - 1] + j * S2[i - 1][j];
        }
    }
}
```

6.36 第二类斯特林数-行

```
// {n,m}->n个不同元素划分成m个相同的集合中（不能有空集）的方案数。

// {n,m}={n-1,m-1}+k{n-1,m}

// {n,m}=\sum_{i=0}^n \frac{i^n}{i!} * {(-1)^{m-i}}{(m-i)!}

const int N = 1e6 + 10;
const ll mod = 167772161;

```

```
ll F[N], invF[N];
void init() ;

ll qpow(ll a, ll b, ll mod) ;

const ll G = 3;
const ll invG = qpow(G, mod - 2, mod);
int tr[N];

void NTT(ll *A, int len, int type) ;
void mul(ll *a, ll *b, int n) ;

ll a[N], b[N];

void solve() {
    init();
    int n; cin >> n;
    for(int i = 0;i <= n; i++) {
        a[i] = qpow(i, n, mod) * invF[i] % mod;
        if(i & 1) b[i] = mod - invF[i];
        else b[i] = invF[i];
    }
    mul(a, b, 2 * n);
    for(int i = 0;i <= n; i++) cout << a[i] << (i == n ? endl : " ");
}
```

6.37 第二类斯特林数-列

```
// 把n个不同元素划分成m个相同的集合（不能有空集）的方案数。

// k!\sum_{i=0}\frac{{i,k}x^i}{i!}=(e^x-1)^k

const int N = 6e5 + 10;
const ll mod = 167772161;

ll quick_pow(ll a, ll b) ;

const ll G = 3;
const ll invG = quick_pow(G, mod - 2);

int tr[N];
bool flag;

void NTT(ll *A, int len, int type) ;
void mul(ll *a, ll *b, int len) ;
void Get_Der(ll *f, ll *g, int len) ;
void Get_Int(ll *f, ll *g, int len) ;
void Get_Inv(ll *f, ll *g, int n) ;
void Get_Ln(ll *f, ll *g, int n) ;
void Get_Exp(ll *f, ll *g, int n) ;
void Get_Pow(ll *f, ll *g, int n, ll k1, ll k2);

ll a[N], ans[N];

ll F[N], invF[N], inv[N];
void init() ;
void solve() {
    init();
    int n; ll k; cin >> n >> k; n++;
    if(k >= mod) flag = 1;
    for(int i = 1;i < n; i++) a[i] = invF[i];
    Get_Pow(a, ans, n, k % mod, k % (mod - 1));
    for(int i = 0;i < n; i++) {
        cout << ans[i] * invF[k] % mod * F[i] % mod << (i == n - 1 ? endl : " ");
    }
}
```

6.38 第一类斯特林数-行

```
#include <algorithm>
#include <cstdio>
#include <cstring>

typedef long long LL;
const int N = 550050;
const int mod = 167772161;

LL pow_mod(LL a, LL b) {
  LL ans = 1;
  for (; b; b >>= 1, a = a * a % mod)
    if (b & 1) ans = ans * a % mod;
  return ans;
}

int L, rev[N];
LL w[N], inv[N], fac[N], ifac[N];

void Init(int n) {
  L = 1;
  while (L <= n) L <<= 1;
  for (int i = 1; i < L; ++i)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) * L / 2);
  LL wn = pow_mod(3, (mod - 1) / L);
  w[L >> 1] = 1;
  for (int i = L >> 1; i < L; ++i) w[i + 1] = w[i] * wn % mod;
  for (int i = (L >> 1) - 1; i; --i) w[i] = w[i << 1];
}

void DFT(LL *A, int len) {
  int k = __builtin_ctz(L) - __builtin_ctz(len);
  for (int i = 1; i < len; ++i) {
    int j = rev[i] >> k;
    if (j > i) std::swap(A[i], A[j]);
  }
  for (int h = 1; h < len; h <<= 1)
    for (int i = 0; i < len; i += (h << 1))
      for (int j = 0; j < h; ++j) {
        LL t = A[i + j + h] * w[j + h] % mod;
        A[i + j + h] = A[i + j] - t;
        A[i + j] += t;
      }
  for (int i = 0; i < len; ++i) A[i] %= mod;
}

```

```
void IDFT(LL *A, int len) {
  std::reverse(A + 1, A + len);
  DFT(A, len);
  int v = mod - (mod - 1) / len;
  for (int i = 0; i < len; ++i) A[i] = A[i] * v % mod;
}

void offset(const LL *f, int n, LL c, LL *g) {
  // g(x) = f(x + c)
  // g[i] = 1/i! sum_{j=i}^n j!f[j] c^(j-i)/(j-i)!
  static LL tA[N], tB[N];
  int l = 1; while (l <= n + n) l <<= 1;
  for (int i = 0; i < n; ++i) tA[n - i - 1] = f[i] * fac[i] % mod;
  LL pc = 1;
  for (int i = 0; i < n; ++i, pc = pc * c % mod) tB[i] = pc * ifac[i] % mod;
  for (int i = n; i < l; ++i) tA[i] = tB[i] = 0;
  DFT(tA, l); DFT(tB, l);
  for (int i = 0; i < l; ++i) tA[i] = tA[i] * tB[i] % mod;
  IDFT(tA, l);
  for (int i = 0; i < n; ++i)
    g[i] = tA[n - i - 1] * ifac[i] % mod;
}

void Solve(int n, LL *f) {
  if (n == 0) return void(f[0] = 1);
  static LL tA[N], tB[N];
  int m = n / 2;
  Solve(m, f);
  int l = 1; while (l <= n) l <<= 1;
  offset(f, m + 1, m, tA);
  for (int i = 0; i <= m; ++i) tB[i] = f[i];
  for (int i = m + 1; i < l; ++i) tA[i] = tB[i] = 0;
  DFT(tA, l); DFT(tB, l);
  for (int i = 0; i < l; ++i) tA[i] = tA[i] * tB[i] % mod;
  IDFT(tA, l);
  if (n & 1)
    for (int i = 0; i <= n; ++i)
      f[i] = ((i ? tA[i - 1] : 0) + (n - 1) * tA[i]) % mod;
  else
    for (int i = 0; i <= n; ++i)
      f[i] = tA[i];
}

LL f[N];

int main() {
  int n;
  scanf("%d", &n);
  Init(n * 2);
  inv[1] = 1;
  for (int i = 2; i <= n; ++i) inv[i] = -(mod / i) * inv[mod % i] % mod;
  fac[0] = ifac[0] = 1;
  for (int i = 1; i <= n; ++i) {
    fac[i] = fac[i - 1] * i % mod;
    ifac[i] = ifac[i - 1] * inv[i] % mod;
  }
  Solve(n, f);
  for (int i = 0; i <= n; ++i)
    printf("%lld ", (f[i] + mod) % mod);
  return 0;
}
```

6.39 第一类斯特林数-列

```
#include <bits/stdc++.h>
using namespace std;

#define Int register int
#define mod 167772161
#define MAXN 531072
#define Gi 3

int quick_pow (int a,int b,int c)
{
    int res = 1;
    while (b){
        if (b & 1) res = 1ll * res * a % c;
        a = 1ll * a * a % c;
        b >>= 1;
    }
    return res;
}

int limit = 1,l,r[MAXN];

void NTT (int *a,int type)
{
    for (Int i = 0;i < limit;++ i) if (i < r[i]) swap (a[i],a[r[i]]);
    for (Int mid = 1;mid < limit;mid <<= 1){
        int Wn = quick_pow (Gi,(mod - 1) / (mid << 1),mod);
        if (type == -1) Wn = quick_pow (Wn,mod - 2,mod);
        for (Int R = mid << 1,j = 0;j < limit;j += R){
            for (Int k = 0,w = 1;k < mid;++ k,w = 1ll * w * Wn % mod)
            {
                int x = a[j + k],y = 1ll * w * a[j + k + mid] % mod;
                a[j + k] = (x + y) % mod,a[j + k + mid] = (x + mod - y) % mod;
            }
        }
    }
    if (type == 1) return ;
    int Inv = quick_pow (limit,mod - 2,mod);
    for (Int i = 0;i < limit;++ i) a[i] = 1ll * a[i] * Inv % mod;
}

int c[MAXN];

void Solve (int len,int *a,int *b)
{
    if (len == 1) return b[0] = quick_pow (a[0],mod - 2,mod),void ();
    Solve ((len + 1) >> 1,a,b);
    limit = 1,l = 0;
    while (limit < (len << 1)) limit <<= 1,l ++;
    for (Int i = 0;i < limit;++ i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    for (Int i = 0;i < len;++ i) c[i] = a[i];
    for (Int i = len;i < limit;++ i) c[i] = 0;
    NTT (c,1);NTT (b,1);
    for (Int i = 0;i < limit;++ i) b[i] = 1ll * b[i] * (2 + mod - 1ll * c[i] * b[i] % mod) % mod;
    NTT (b,-1);
    for (Int i = len;i < limit;++ i) b[i] = 0;
}

void deravitive (int *a,int n){
    for (Int i = 1;i <= n;++ i) a[i - 1] = 1ll * a[i] * i % mod;
    a[n] = 0;
}

void inter (int *a,int n){
    for (Int i = n;i >= 1;-- i) a[i] = 1ll * a[i - 1] * quick_pow (i,mod - 2,mod) % mod;
    a[0] = 0;
}

int b[MAXN];

void Ln (int *a,int n){
    memset (b,0,sizeof (b));
    Solve (n,a,b);deravitive (a,n);
    while (limit <= n) limit <<= 1,l ++;
    for (Int i = 0;i < limit;++ i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    NTT (a,1),NTT (b,1);
    for (Int i = 0;i < limit;++ i) a[i] = 1ll * a[i] * b[i] % mod;
    NTT (a,-1);
    inter (a,n);
    for (Int i = n + 1;i < limit;++ i) a[i] = 0;
}

int F0[MAXN];

void Exp (int *a,int *B,int n){
    if (n == 1) return B[0] = 1,void ();
    Exp (a,B,(n + 1) >> 1);
    for (Int i = 0;i < limit;++ i) F0[i] = B[i];
    Ln (F0,n);
    F0[0] = (a[0] + 1 + mod - F0[0]) % mod;
    for (Int i = 1;i < n;++ i) F0[i] = (a[i] + mod - F0[i]) % mod;
    NTT (F0,1);NTT (B,1);
    for (Int i = 0;i < limit;++ i) B[i] = 1ll * F0[i] * B[i] % mod;
    NTT (B,-1);
    for (Int i = n;i < limit;++ i) B[i] = 0;
}

int read ()
{
    int x = 0;char c = getchar();int f = 1;
    while (c < '0' || c > '9'){if (c == '-') f = -f;c = getchar();}
    while (c >= '0' && c <= '9'){x = (int)((int)(x << 3) % mod + (int)(x << 1) % mod + c - '0') % mod;c = getchar();}
    return x * f;
}

void write (int x)
{
    if (x < 0){x = -x;putchar ('-');}
    if (x > 9) write (x / 10);
    putchar (x % 10 + '0');
}

int n,k;
int fac[MAXN],A[MAXN],B[MAXN];

signed main()
{
    n = read (),k = read ();
    for (Int i = 0;i < n;++ i) A[i] = quick_pow (i + 1,mod - 2,mod);
    Ln (A,n);
    for (Int i = 0;i < n;++ i) A[i] = 1ll * A[i] * k % mod;
    Exp (A,B,n);fac[0] = 1;
    for (Int i = 1;i <= max (n,k);++ i) fac[i] = 1ll * fac[i - 1] * i % mod;
    for (Int i = n;i >= k;-- i) B[i] = B[i - k];
    for (Int i = 0;i < k;++ i) B[i] = 0;int Inv = quick_pow (fac[k],mod - 2,mod);
    for (Int i = 0;i <= n;++ i) write (1ll * B[i] * fac[i] % mod * Inv % mod),putchar (
    ' ');
    putchar ('\n');
    return 0;
}
```

6.40 整数拆分多项式求逆

```

// NTT求法，任意模数复杂度较高
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const int N = 1e5 + 10;

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
     rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

ll mod, g;
int tr[N];
ll F[N], G[N];

ll quick_pow(ll a, ll b) ;

int getLen(int n) ;

void NTT(ll *a, int len, int opt) ;

void mul(ll *a, ll *b, ll *z, int n) ;

void Get_Inv(ll *f, ll *g, int n) ;

void ChaiFen(ll *f, ll *g, int n) {
    int len = getLen(n);
    for(int i = 0;i < len; i++) {
        ll a = 1ll * i * (3 * i - 1) / 2; ll b = 1ll * i * (3 * i + 1) / 2;
        if(a > len && b > len) break;
        ll tmp;
        if(i & 1) tmp = mod - 1;
        else tmp = 1;
        if(a < len) f[a] = tmp;
        if(b < len) f[b] = tmp;
    }

    Get_Inv(f, g, n);
}

int main() {
    int n;
    cin >> n;
    ChaiFen(F, G, n);
    for(int i = 1;i <= n; i++) cout << G[i] << endl;
}
```

6.41 普通型母函数

```
// 普通型母函数：（1+x^1+x^2+...）(1+x^2+x^4)(1+x^3+x^6..)(...)(...)... 类似整数拆分

// a_n=1,1,1,1... = \frac{1}{1-x}
// a_n=1,0,1,0... = \frac{1}{1-x^2}
// a_n=1,2,3,4... = \frac{1}{(1-x)^2}
// a_n=C(m,n)     = (1+x)^m
// a_n=C(m+n,n)   = \frac{1}{(1-x)^{m+1}}

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

// 求解硬币等普通问题

const int N = 1e5 + 10;

int a[N]; // 权重为i的组合数，a[P]为答案
int b[N]; // 辅助数组
int P; // 需要被分解的数
int k; // 物品个数
int v[N]; // 每个物品的权重
int n1[N]; // 对于每种物品起始的因子（所需要的每个物品最小个数），最小为0
int n2[N]; // 对于每种物品最终的因子（所需要的每个物品最大个数），最大为INF

// 模板一（标准）

void Calc1() {
    memset(a, 0, sizeof(a));
    a[0] = 1;

    for(int i = 1;i <= k; i++) { // 枚举每个物品因子
        memset(b, 0, sizeof(b));
        for(int j = n1[i];j <= n2[i] && j * v[i] <= P; j++) { // 每个物品从最小因子到最大因子循环,如果n2是无穷的，则j<=n2[i]可以删去
            for(int m = 0;m + j * v[i] <= P; m++) { // 循环a的每个项
```

```
                b[m + j * v[i]] += a[m]; // 把结果加到对应项里，有点dp的味道
            }
        }
        memcpy(a, b, sizeof(b));
    }
}

// 模板二（数据量大的时候可以用，快速）

void Calc2() {
    memset(a, 0, sizeof(a));
    a[0] = 1;
    int last = 0;
    for(int i = 1; i <= k; i++) {
        int last2 = min(last + n2[i] * v[i], P);//计算下一次的last
        memset(b, 0, sizeof(int) * (last2 + 1));//只清空b[0..last2]
        for(int j = n1[i]; j <= n2[i] && j * v[i] <= last2; j++) //last2
            for(int m = 0; m <= last && m + j * v[i] <= last2; m++) //一个是last，一个是last2
                b[m + j * v[i]] += a[m];
        memcpy(a, b, sizeof(int) * (last2 + 1));//b赋值给a，只赋值0..last2
        last = last2;//更新last
    }
}
```

6.42 指数型母函数

```

// 需要借助e^x的泰勒展开，一般求解多重排列数，即有 种物品，已知每种物品的数量为 k1,k2,...,kn 个，求从中选出m件物品的排列数。

// 对n个元素全排列，方案数为n!/(n1!n2!...nk!)，对n个中的r个元素进行全排列，这里就用到了指数型母函数，即G(x)=(1+x/1!+x^2/2!+...+x^k1/k1!)(1+x/1!+x^2/2!+...+x^k2/k2!)...(1+x/1!+x^2/2!+...+x^kn/kn!)

// 化简得G(x)=a0 + a1*x+a2*x^2/2!+...+ap*x^p/p!   (p = k1+k2+k3+...) ai为选出i个物品的排列方案数

//  若题目有规定条件，比如需要物品i出现非0的偶数次，即原式为(x^2/2!+x^4/4!+...+x^ki/ki!)

#include <bits/stdc++.h>
using namespace std;

typedef long double ld;

double num[15]; // 每种物品的数量，第i个物品有num[i]个

double a[15], b[15];

double f[120]; // 阶乘

void fac()
{
    f[0] = 1;
    for(int i = 1;i <= 105; i++)
        f[i] = f[i - 1] * i;
}

```

```
void Calc() {
    int n, m;
    cin >> n >> m;
    for(int i = 1;i <= n; i++)
        cin >> num[i];

    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));

    for(int i = 0;i <= num[1]; i++) {
        a[i] = 1.0 / f[i];
    }

    for(int i = 2;i <= n; i++) {
        for(int j = 0;j <= m; j++) {
            for(int k = 0;k <= num[i] && j + k <= m; k++) {
                b[j + k] += a[j] / f[k];
            }
        }

        for(int j = 0;j <= m; j++) {
            a[j] = b[j];
            b[j] = 0;
        }
    }

    cout << a[m] * f[m] << endl;
}
```

6.43 伯努利数求和

```
namespace BNL {
    const int N = 1e7 + 10, M = 1e6 + 10;
    struct Complex {
        double x, y;
        Complex(double a = 0, double b = 0): x(a), y(b) {}
        Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
        Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
        Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
        Complex conj() { return Complex(x, -y); }
    } w[N];

    int tr[N];
    ll F[N], G[N];

    ll quick_pow(ll a, ll b, ll p) {
        ll ans = 1;
        while(b) {
            if(b & 1) ans = ans * a % p;
            a = a * a % p;
            b >>= 1;
        }
        return ans % p;
    }

    void FFT(Complex *A, int len) {
        for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
        for (int i = 2, lyc = len >> 1; i <= len; i <<= 1, lyc >>= 1)
            for (int j = 0; j < len; j += i) {
                Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
                for (int k = 0; k < i >> 1; k++) {
                    Complex tmp = *r * *p;
                    *r = *l - tmp, *l = *l + tmp;
                    ++l, ++r, p += lyc;
                }
            }
    }

    inline void MTT(ll *x, ll *y, ll *z, int n) {
        int len = 1; while (len <= n) len <<= 1;
        for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
        for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI * i / len));

        for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
        static Complex a[N], b[N];
        static Complex dfta[N], dftb[N], dftc[N], dftd[N];

        for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
        for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
        FFT(a, len), FFT(b, len);
        for (int i = 0; i < len; i++) {
            int j = (len - i) & (len - 1);
            static Complex da, db, dc, dd;
            da = (a[i] + a[j].conj()) * Complex(0.5, 0);
            db = (a[i] - a[j].conj()) * Complex(0, -0.5);
            dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
            dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
            dfta[j] = da * dc;
            dftb[j] = da * dd;
            dftc[j] = db * dc;
            dftd[j] = db * dd;
        }
        for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
        for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
        FFT(a, len), FFT(b, len);
        for (int i = 0; i < len; i++) {
            int da = (ll)(a[i].x / len + 0.5) % mod;
            int db = (ll)(a[i].y / len + 0.5) % mod;
            int dc = (ll)(b[i].x / len + 0.5) % mod;
            int dd = (ll)(b[i].y / len + 0.5) % mod;
            z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
        }
    }

    int getLen(int n) {
        int len = 1; while (len < (n << 1)) len <<= 1;
        for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
        for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI * i / len));
        return len;
    }

    void Get_Inv(ll *f, ll *g, int n) {
        if(n == 1) { g[0] = quick_pow(f[0], mod - 2, mod); return ; }
        Get_Inv(f, g, (n + 1) >> 1);
        int len = getLen(n);
        static ll c[N];
        for(int i = 0;i < len; i++) c[i] = i < n ? f[i] : 0;
        MTT(c, g, c, len); MTT(c, g, c, len);
        for(int i = 0;i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
        for(int i = n;i < len; i++) g[i] = 0;
        for(int i = 0;i < len; i++) c[i] = 0;
    }

    ll ff[N], invff[N], inv[N];
    ll B[N];

    ll C(ll m, ll n) {
        if(m < 0 || n < 0 || n > m)
            return 0;
        ll ans = ff[m];
        ans = ans * invff[n] % mod;
        ans = ans * invff[m - n] % mod;
        return ans;
    }

    void init(int m) {
        ff[0] = ff[1] = inv[0] = inv[1] = invff[0] = invff[1] = 1;
        for(int i = 2;i < M; i++)
        {
            ff[i] = ff[i - 1] * i % mod;
            inv[i] = mod - (mod / i) * inv[mod % i] % mod;
            invff[i] = invff[i - 1] * inv[i] % mod;
        }

        for(int i = 0;i <= m + 10; i++) F[i] = invff[i + 1];
        Get_Inv(F, G, m + 10);
        for(int i = 0;i <= m + 10; i++) B[i] = G[i] * ff[i] % mod;
    }

    ll solve(ll n, int k) {
        init(k);
        ll ans = 0, prod = n % mod;
        for(int i = k; ~i ; i--) {
            ans = (ans + prod * B[i] % mod * C(k + 1, i) % mod) % mod;
            prod = prod * n % mod;
        }
        ans = ans * quick_pow(k + 1, mod - 2, mod) % mod;
        return ans;
    }
}

void solve() {
    ll n; int k; cin >> n >> k;
    cout << BNL::solve(n + 1, k) << endl;
}
```

6.44 fibonacciBigInteger

```
class Fib {
    int n;
    std::vector<std::vector<int>> f;
    Fib(int _n) : n(_n), f(_n, std::vector<int>(_n)) {
        f[1][1] = 1;
        f[2][1] = 1;
        int s, add = 0;
        for (int i = 3; i < _n; i++) {
            for (int j = 1; j < _n; j++) {
                s = f[i - 2][j] + f[i - 1][j] + add;
                f[i][j] = s % 10;
                add = s / 10;
            }
        }
    }

    std::string get(const int m) {
        int k = n - 1;
        while (!f[m][k]) k--;
        std::string s = "";
        s.push_back(f[n][k] + '0');
        return s;
    }
};
```

6.45 fibonacciCircular

```
// 斐波那契循环节

typedef long long ll;
typedef long double ld;
typedef pair<int, int> pdd;

#define INF 0x7f7f7f
#define mem(a,b) memset(a , b , sizeof(a))
#define FOR(i, x, n) for(int i = x;i <= n; i++)

const ll mod = 1e9 + 7;
const int maxn = 5e6 + 10;

bool is_prime[maxn];
ll prime[maxn];
int p;

void sieve() // 素数筛
{
    p = 0;
    mem(is_prime, true);
    is_prime[0] = is_prime[1] = false;
    for(int i = 2;i < maxn; i++)
    {
        if(is_prime[i])
        {
            prime[++p] = i;
            for(int j = i + i;j < maxn; j += i)
            {
                is_prime[j] = false;
            }
        }
    }
}

ll gcd(ll a, ll b)
{
    return b ? gcd(b, a % b) : a;
}

ll quick_pow(ll a, ll b)
{
    a %= mod;
    ll ans = 1;
    ll base = a;
    while(b)
    {
        if(b&1)
        {
            ans = ans * base % mod;
        }
        base = base * base % mod;
        b >>= 1;
    }
    return ans % mod;
}

ll num[maxn]; // 所有质数的循环节
ll f[maxn]; // 斐波那契数列

void Fib_Cyclic_node()
{
    num[1] = 3;
    for(int i = 2;i <= p; i++) // 找每个素数的循环节num[1~p]
    {
        f[1] = 1;
        f[2] = 2;
        int x = 3;
        while(true)
        {
            f[x] = f[x - 1] + f[x - 2];
            f[x] %= prime[i];
            if(f[x] == 1 && f[x - 1] == 0) // f[x] % prime[i] == f[1]
                break;
            x++;
        }
        num[i] = x;
    }

    ll n;
    cin >> n; // 如果是质数，那循环节就是num[n] ;   如果是合数，那循环节就是n的素因子的最小公倍数
    ll ans = 1;
    ll x;
    for(int i = 1;i <= p; i++)
    {
        if(n % prime[i] == 0)
        {
            x = 0;
            while(n % prime[i] == 0)
            {
                n /= prime[i];
                x++;
            }
        }
        ll k = num[i] * quick_pow(prime[i], x - 1);
        ans = ans * k / gcd(ans, k); // 最小公倍数
    }
    cout << ans << endl; // 最小循环节
}

// 广义斐波那契循环节

// fib(n) = a * fib(n - 1) + b * fib(n - 2)
// fib(1) = c   fib(2) = d
// 求f(n) mod p的循环节
//c = a * a - 4b是模p的二次剩余时枚举p-1的因子，否则枚举(p+1)(p-1)的因子

typedef long long ll;
typedef long double ld;
typedef pair<int, int> pdd;

#define INF 0x7f7f7f
#define mem(a,b) memset(a , b , sizeof(a))
#define FOR(i, x, n) for(int i = x;i <= n; i++)

ll fac[2][505];
ll cnt, ct;

ll pri[6] = {2, 3, 7, 109, 167, 500000003};
ll num[6] = {4, 2, 1, 2, 1, 1};

const ll mod = 1e9 + 7;
const int maxn = 5e6 + 10;

struct Matrix{
    ll m[2][2];
};

Matrix A;

Matrix I = {1, 0, 0 , 1}; // 单位矩阵

Matrix multi(Matrix a, Matrix b) // 矩阵乘法
{
    Matrix C;
    for(int i = 0;i < 2; i++)
    {
        for(int j = 0;j < 2; j++)
        {
            C.m[i][j] = 0;
            for(int k = 0;k < 2; k++)
            {
                C.m[i][j] = (C.m[i][j] + a.m[i][k] * b.m[k][j] % mod) % mod;
            }
            C.m[i][j] %= mod;
        }
    }
    return C;
}
```

```

Matrix quick_Matrix(Matrix a, ll b) // 矩阵快速幂
{
    Matrix ans = I, base = a;
    while(b)
    {
        if(b & 1)
        {
            ans = multi(a, base);
        }
        base = multi(base, base);
        b >>= 1;
    }
    return ans;
}

ll quick_pow(ll a, ll b) ;

ll legendre(ll a, ll p) // 勒让德符号 = {1, -1, 0}
{
    ll k = quick_pow(a, (p - 1) >> 1);
    if(k == 1)
        return 1;
    else
        return -1;
}

void DFS(int dept,ll product = 1)
{
    if(dept == cnt)
    {
        fac[1][ct++] = product;
        return;
    }
    for(int i=0; i<=num[dept]; i++)
    {
        DFS(dept+1,product);
        product *= pri[dept];
    }
}

bool Fib_node(Matrix a, ll n) // n是否为循环节
{
    Matrix ans = quick_Matrix(a, n);
    return (ans.m[0][0] == 1 && ans.m[0][1] == 0 && ans.m[1][0] == 0 && ans.m[1][1] == 1); // 是否为单位矩阵I
}

ll Fib_Cyclic_node(ll a, ll b, ll c, ll d) // 广义斐波那契循环节斐波那契循环节
{
    fac[0][0] = 1;
    fac[0][1] = 2;
    fac[0][3] = 500000003;
    fac[0][3] = 1000000006;
    ll c = a * a - 4 * b;
    A.m[0][0] = a;
    A.m[0][1] = b;
    A.m[1][0] = 1;
    A.m[1][1] = 0;
    if(legendre(c, mod) == 1) // c是否为1e9+7的二次剩余
    {
        for(int i = 0;i < 4; i++)
        {
            if(Fib_node(A, fac[0][i]))
                return fac[0][i];
        }
    }
    else
    {
        ct = 0;
        cnt = 6;
        DFS(0, 1);
        sort(fac[1], fac[1] + ct);
        for(int i = 0;i < ct; i++)
        {
            if(Fib_node(A, fac[1][i]))
                return fac[1][i];
        }
    }

}

int main()
{
    ll a, b, c, d;
    cin >> a >> b >> c >> d;
    ll n = Fib_Cyclic_node(a, b, c, d); // 广义斐波那契循环节循环节长度
    cout << n << endl;
}
```

6.46 Matrix

```
template <typename T>
struct Matrix {
    const int n;
    std::vector<std::vector<T>> mat;

    Matrix(int n = 0) : n(n), mat(n, std::vector<T>(n)) {}
    Matrix(const std::vector<std::vector<T>> &rhs) : n(n) {
        mat = std::move(rhs);
    }
    Matrix(const Matrix<T> &rhs) : n(n) {
        mat = std::move(rhs.mat);
    }
    void identify() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                mat[i][j] = (i == j);
            }
        }
    }
    T getVal(const int& i, const int& j) { return mat[i][j]; }
    T size() { return n; }
    const Matrix operator+(const Matrix& rhs) {
        Matrix ret(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                ret.mat[i][j] = mat[i][j] + rhs.mat[i][j];
            }
        }
        return ret;
    }
    const Matrix operator-(const Matrix& rhs) {
        Matrix ret(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                ret[i][j] = mat[i][j] - rhs[i][j];
            }
        }
        return ret;
    }
    const Matrix operator*(const Matrix &rhs) {
        Matrix ret(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    ret.mat[i][j] += mat[i][k] * rhs.mat[k][j];
                }
            }
        }
        return ret;
    }
    const Matrix operator+() { return *this; }
    const Matrix operator-() {
        Matrix ret(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                ret.mat[i][j] = -mat[i][j];
            }
        }
        return ret;
    }
    Matrix &operator+=(const Matrix &rhs) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                mat[i][j] += rhs.mat[i][j];
            }
        }
        return *this;
    }
    Matrix &operator-=(const Matrix &rhs) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                mat[i][j] -= rhs.mat[i][j];
            }
        }
        return *this;
    }
    const Matrix operator*=(const Matrix &rhs) {
        Matrix ret(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    ret.mat[i][j] += mat[i][k] * rhs.mat[k][j];
                }
            }
        }
        return ret;
    }
};

template <typename T>
Martix power(Matrix<T> a, T b) {
    Matrix<T> res(2);
    res.mat[0][0] = res.mat[1][1] = 1;
    for (; b /= 2; a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
```

6.47 拉格朗日插值求和

```
namespace polysum {
#define rep(i, a, n) for (int i=a;i<n;i++)
#define per(i, a, n) for (int i=n-1;i>=a;i--)
    const int D = 1010000;///可能需要用到的最高次
    ll a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D], num[D];

    ll powmod(ll a, ll b) {
        ll res = 1;
        a %= mod;
        assert(b >= 0);
        for (; b; b >>= 1) {
            if (b & 1)res = res * a % mod;
            a = a * a % mod;
        }
        return res;
    }

    ///函数用途：给出数列的（d+1）项，其中d为最高次方项
    ///求出数列的第n项，数组下标从0开始
    ll calcn(int d, ll *a, ll n) { /// a[0].. a[d]  a[n]
        if (n <= d) return a[n];
        p1[0] = p2[0] = 1;
        rep(i, 0, d + 1) {
            ll t = (n - i + mod) % mod;
            p1[i + 1] = p1[i] * t % mod;
        }
        rep(i, 0, d + 1) {
            ll t = (n - d + i + mod) % mod;
            p2[i + 1] = p2[i] * t % mod;
        }
        ll ans = 0;
        rep(i, 0, d + 1) {
            ll t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
            if ((d - i) & 1) ans = (ans - t + mod) % mod;
            else ans = (ans + t) % mod;
        }
        return ans;
    }

    void init(int M) {///用到的最高次
        f[0] = f[1] = g[0] = g[1] = 1;
        rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
        g[M + 4] = powmod(f[M + 4], mod - 2);
        per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod;///费马小定理筛逆元
    }

    ///函数用途：给出数列的（m+1）项，其中m为最高次方
    ///求出数列的前（n-1）项的和（从第0项开始）
    ll polysum(ll m, ll *a, ll n) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]
        for (int i = 0; i <= m; i++) b[i] = a[i];

        ///前n项和，其最高次幂加1
        b[m + 1] = calcn(m, b, m + 1);
        rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
        return calcn(m + 1, b, n - 1);
    }

    ll qpolysum(ll R, ll n, ll *a, ll m) { /// a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
        if (R == 1) return polysum(n, a, m);
        a[m + 1] = calcn(m, a, m + 1);
        ll r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;
        h[0][0] = 0;
        h[0][1] = 1;
        rep(i, 1, m + 2) {
            h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
            h[i][1] = h[i - 1][1] * r % mod;
        }
        rep(i, 0, m + 2) {
            ll t = g[i] * g[m + 1 - i] % mod;
            if (i & 1) p3 = ((p3 - h[i][0] * t) % mod + mod) % mod, p4 = ((p4 - h[i][1] * t) % mod + mod) % mod;
            else p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
        }
        c = powmod(p4, mod - 2) * (mod - p3) % mod;
        rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
        rep(i, 0, m + 2) C[i] = h[i][0];
        ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
        if (ans < 0) ans += mod;
        return ans;
    }

    ll solve(ll n, int k) {
        init(k + 10);
        for (int i = 0; i <= k + 1; i++) num[i] = powmod((ll) i + 1, k);
        ll ans = polysum(k + 1, num, n) % mod;
        return ans;
    }
}

void solve() {
    ll n;
    int k;
    cin >> n >> k;
    cout << polysum::solve(n, k) << endl;
}
```

6.48 SolveLinearSystem

```cpp
bool IsZero(double v) {
  return abs(v) < 1e-9;
}

enum GAUSS_MODE {
  DEGREE, ABS
};

template <typename T>
void GaussianElimination(std::vector<std::vector<T>>& a, int limit, GAUSS_MODE mode = ABS) {
    if (a.empty() || a[0].empty()) {
        return;
    }
    int h = static_cast<int>(a.size());
    int w = static_cast<int>(a[0].size());
    for (int i = 0; i < h; i++) {
        assert(w == static_cast<int>(a[i].size()));
    }
    assert(limit <= w);
    std::vector<int> deg(h);
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            deg[i] += !IsZero(a[i][j]);
        }
    }
    int r = 0;
    for (int c = 0; c < limit; c++) {
        int id = -1;
        for (int i = r; i < h; i++) {
            if (!IsZero(a[i][c]) && (id == -1 || (mode == DEGREE && deg[i] < deg[id]) || (mode == ABS && abs(a[id][c]) < abs(a[i][c])))) {
                id = i;
            }
        }
        if (id == -1) {
        continue;
        }
        if (id > r) {
            std::swap(a[r], a[id]);
            std::swap(deg[r], deg[id]);
            for (int j = c; j < w; j++) {
                a[id][j] = -a[id][j];
            }
        }
        std::vector<int> nonzero;
        for (int j = c; j < w; j++) {
            if (!IsZero(a[r][j])) {
                nonzero.push_back(j);
            }
        }
        T inv_a = 1 / a[r][c];
        for (int i = r + 1; i < h; i++) {
            if (IsZero(a[i][c])) {
                continue;
            }
            T coeff = -a[i][c] * inv_a;
            for (int j : nonzero) {
                if (!IsZero(a[i][j])) deg[i]--;
                a[i][j] += coeff * a[r][j];
                if (!IsZero(a[i][j])) deg[i]++;
            }
        }
        ++r;
    }
    for (r = h - 1; r >= 0; r--) {
        for (int c = 0; c < limit; c++) {
            if (!IsZero(a[r][c])) {
                T inv_a = 1 / a[r][c];
                for (int i = r - 1; i >= 0; i--) {
                    if (IsZero(a[i][c])) {
                        continue;
                    }
                    T coeff = -a[i][c] * inv_a;
                    for (int j = c; j < w; j++) {
                        a[i][j] += coeff * a[r][j];
                    }
                }
                break;
            }
        }
    }
}

template <typename T>
std::vector<T> SolveLinearSystem(std::vector<std::vector<T>> a, const std::vector<T>& b, int w) {
    int h = static_cast<int>(a.size());
    assert(h == static_cast<int>(b.size()));
    if (h > 0) {
        assert(w == static_cast<int>(a[0].size()));
    }
    for (int i = 0; i < h; i++) {
        a[i].push_back(b[i]);
    }
    GaussianElimination(a, w);
    std::vector<T> x(w, 0);
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            // when IsZero(a[i][j]) is no solution
            if (!IsZero(a[i][j])) {
                x[j] = a[i][w] / a[i][j];
                break;
            }
        }
    }
    return x;
}
```

6.49 矩阵求逆

```
//原始矩阵A[0, n - 1][0, n - 1]
//右边一个单位阵I，在a[0, n - 1][n, (n << 1) - 1]
//将左边A变成单位阵时，右边的I变为A^-1

```

```
ll a[MAX][MAX];
bool Gauss(int n) {
    for (int i = 0, r; i < n; i++) {
        r = i;
        for (int j = i + 1; j < n; j++)
            if (a[j][i] > a[r][i]) r = j;
        if (r != i) swap(a[i], a[r]);
        if (!a[i][i]) return false;//无解

        ll inv = qpow(a[i][i], mod - 2);
        for (int k = 0; k < n; k++) {
            if (k == i) continue;
            ll t = a[k][i] * inv % mod;
            for (int j = i; j < (n << 1); j++)
                a[k][j] = (a[k][j] - t * a[i][j] % mod + mod) % mod;
        }
        for (int j = 0; j < (n << 1); j++) a[i][j] = a[i][j] * inv % mod;
    }
    return true;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        a[i][i + n] = 1;
        for (int j = 0; j < n; j++)
            scanf("%lld", &a[i][j]);
    }
    if(Gauss(n)) {
        for(int i = 0;i < n; i++) {
            for(int j = n;j < n * 2; j++) {
                cout << a[i][j] << " ";
            }
            cout << endl;
        }
    }
    else cout << "No Solution" << endl;
}
```

6.50 eraseLinearBasis

```
// 离线删除操作，维护线性基中每个元素的最晚删除时间。

#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const int maxl = 60;

struct LinearBasis {
    ll a[maxl + 10], tim[maxl + 10];
    int n, size; // 每个相同异或值有2^{n-size}个
    vector<ll> v;

    LinearBasis() {
        memset(a, 0, sizeof(a));
        size = n = 0;
        v.clear();
    }

    void insert(ll x, ll t) {
        n++;
        for(int i = maxl;i >= 0; i--) {
            if(!(x >> i & 1)) continue ;
            if(a[i]) {
                if(t > tim[i]) swap(t, tim[i]), swap(x, a[i]);
                x ^= a[i];
            }
            else {
                ++size;
                a[i] = x; tim[i] = t;
                return ;
            }
        }
    }

    void erase(ll t) {
        for(int i = maxl;i >= 0; i--) {
            if(tim[i] == t) {
                a[i] = tim[i] = 0; --size;
                return ;
            }
        }
    }
};

int main() {
    LinearBasis lb;
    int n, m; cin >> n >> m;
    vector<ll> opt(n + 10), a(n + 10), del(n + 10), pre(n + 10);
    for(int i = 1;i <= m; i++) {
        cin >> opt[i] >> a[i];
        if(opt[i] == 1) pre[a[i]] = i, del[i] = m + 1;
        else del[pre[a[i]]] = i;
    }
    ll ans = 0;
    for(int i = 1;i <= m; i++) {
        if(opt[i] == 1) lb.insert(a[i], del[i]);
        else lb.erase(i);
        ans ^= 1ll << (lb.n - lb.size);
    }
    cout << ans << endl;
}
```

6.51 simpleLinearBasis

```
template<class Info>
struct LinearBasis {
    const int n;
    int size;
    long long num;
    // 每个异或值都相同的个数都为2^n-r,所以不同的异或值有2^r个.
    const int maxl = 61;
    std::vector<long long> a, v;
    LinearBasis(int n) : n(n), size(0), a(maxl) {}
    LinearBasis(std::vector<Info> init) : LinearBasis(init.size()) {
        auto insert = [&](long long t) {
            for (int i = maxl - 1; i >= 0; --i) {
                if (!(t >> i & 1)) continue;
                if (a[i]) t ^= a[i];
                else {
                    ++size;
                    // Rebuild
                    for (int j = i - 1; j >= 0; --j) if (t >> j & 1) t ^= a[j];
                    for (int j = i + 1; j < maxl; ++j) if (a[j] >> i & 1) a[j] ^= t;
                    //
                    a[i] = t;
                    return;
                }
            }
        };
        for(int i = 0;i < n; i++) insert(init[i]);
        auto basis = [&]() {
            for (int i = 0; i < maxl; ++i) if (a[i]) v.push_back(i);
        };
        basis();
        num = 1LL << size;
    }

    // 查询能否xor出x这个数
    bool find(long long x) {
        for(int i = maxl - 1;i >= 0; i--) {
            if(x >> i & 1) {
                if(!a[i]) return false;
                x ^= a[i];
            }
        }
        return true;
    }

    // 查询异或最大值
    long long askmax() {
        long long ans = 0;
        for(int i = maxl - 1;i >= 0; i--) ans = max(ans, ans ^ a[i]);
        return ans;
    }

    // 查询异或最小值
    long long askmin() {
        if((int) v.size() < n) return 0;
        for(int i = 0;i < maxl; i++) if(a[i]) return a[i];
        return 0;
    }

    // 查询异或第k小
    long long askmink(long long x) {
        if(v.size() != n) x--;
        if(!x) return 0;
        if(x >= (1LL << v.size())) return -1;
        long long ans = 0;
        for(int i = 0;i < (int) v.size(); i++) {
            if(x >> i & 1) ans ^= a[v[i]];
        }
        return ans;
    }

    long long rank(long long x) {
        long long ret = 0;
        for (int i = 0; i < (int) v.size(); ++i) if (x >> v[i] & 1) ret += 1LL << i;
        return ret;
    }
};
```

6.52 intervalModifyLinearBasis

```
#include<bits/stdc++.h>
#define maxn 200005
using namespace std;
struct Base{
    int a[31],cnt;
    void clear(){memset(a,0,sizeof a),cnt=0;}
    void ins(int x){
        if(cnt==30) return;
        for(int i=29;i>=0&&x;i--) if(x>>i&1){
            if(a[i]) x^=a[i];
            else {a[i]=x,cnt++;return;}
        }
    }
    void merge(const Base &B){
        for(int i=29;i>=0;i--) if(B.a[i]) ins(B.a[i]);
    }
}t[maxn<<2];
int n,m,a[maxn],b[maxn];
int arr[maxn];
void upd(int i,int v){for(;i<=n;i+=i&-i) arr[i]^=v;}
int qxor(int i){int s=0;for(;i;i-=i&-i) s^=arr[i];return s;}
void upd(int i){t[i]=t[i<<1],t[i].merge(t[i<<1|1]);}
void build(int i,int l,int r){
    if(l==r) return t[i].ins(b[l]);
    int mid=(l+r)>>1;
    build(i<<1,l,mid),build(i<<1|1,mid+1,r);
    upd(i);
}
void mdf(int i,int l,int r,int x){
    if(l==r) {t[i].clear(),t[i].ins(b[x]);return;}
    int mid=(l+r)>>1;
    x<=mid?mdf(i<<1,l,mid,x):mdf(i<<1|1,mid+1,r,x);
    upd(i);
}
void qry(int i,int l,int r,int x,int y){
    if(x<=l&&r<=y) return t[0].merge(t[i]);
    int mid=(l+r)>>1;
    if(x<=mid) qry(i<<1,l,mid,x,y);
    if(y>mid) qry(i<<1|1,mid+1,r,x,y);
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]),b[i]=a[i]^a[i-1],upd(i,b[i]);
    build(1,1,n);
    for(int op,l,r,x;m--;){
        scanf("%d%d%d",&op,&l,&r);
        if(op==1){
            scanf("%d",&x);
            upd(l,x),upd(r+1,x),b[l]^=x,b[r+1]^=x;
            mdf(1,1,n,l); if(r<n) mdf(1,1,n,r+1);
        }
        else{
            t[0].clear(),t[0].ins(qxor(l)); if(l<r) qry(1,1,n,l+1,r);
            printf("%d\n",1<<t[0].cnt);
        }
    }
}
```

6.53 noIntervalModifyLinearBasis

```
// 扫描r，维护线性基中每个元素的最大左端点l。与删除操作类似。
// 这个可以强制在线，把每个r的线性基存下来即可。
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 10;
struct node {
    int l, r, id;
    bool operator < (const node &p) const {
        return r < p.r;
    }
}q[N];

const int maxl = 60;

struct LinearBasis {
    ll a[maxl + 10], pos[maxl + 10];
    int n, size; // 每个相同异或值有2^{n-size}个
    vector<ll> v;

    LinearBasis() {
        memset(a, 0, sizeof(a));
        size = n = 0;
        v.clear();
    }

    void insert(ll t, ll id) {
        n++;
        for(int i = maxl;i >= 0; i--) {
            if(!(t >> i & 1)) continue ;
            if(a[i]) {
                if(id > pos[i]) swap(id, pos[i]), swap(t, a[i]);
                t ^= a[i];
            }
            else {
                a[i] = t;
                pos[i] = id;
                return ;
            }
        }
    }

    int askmax(ll x) {
        ll ans = 0;
        for(int i = maxl;i >= 0; i--) {
```

```
//            if(pos[i] >= x && !(ans >> i & 1)) ans ^= a[i];
            if(pos[i] >= x) ans = max(ans, ans ^ a[i]);
        }
        return ans;
    }
};

// 给你n个数，每次查询［公式］这个区间，问着个区间的最大异或值。

void solve() {
    LinearBasis lb;
    int n; cin >> n;
    VI a(n + 1);
    for(int i = 1;i <= n; i++) cin >> a[i];
    int m; cin >> m;
    VI ans(m + 1);
    for(int i = 1;i <= m; i++) cin >> q[i].l >> q[i].r, q[i].id = i;
    sort(q + 1, q + m + 1);
    for(int i = 1, j = 1;i <= n; i++) {
        lb.insert(a[i], i);
        for(; j <= m && q[j].r <= i; j++) ans[q[j].id] = lb.askmax(q[j].l);
    }
    for(int i = 1;i <= m; i++) cout << ans[i] << endl;
}
```

6.54 FWT

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod = 998244353;

const int N = 1e5 + 10;
int a[N], b[N];

inline void FWT_OR(int *f, int n, int opt) {
    for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
        for (int i = 0; i < n; i += o)
            for (int j = 0; j < k; j++)
                f[i + j + k] = (f[i + j + k] + 1ll * f[i + j] * opt % mod + mod) % mod;
}

inline void FWT_AND(int *f, int n, int opt) {
    for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
        for (int i = 0; i < n; i += o)
            for (int j = 0; j < k; j++)
                f[i + j] = (f[i + j] + 1ll * f[i + j + k] * opt % mod + mod) % mod;
}

inline void FWT_XOR(int *f, int n, int opt) {
    for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
        for (int i = 0; i < n; i += o)
            for (int j = 0; j < k; j++) {
                ll a0 = f[i + j], a1 = f[i + j + k];
                f[i + j] = (a0 + a1) % mod * opt % mod;
                f[i + j + k] = (a0 - a1 + mod) % mod * opt % mod;
            }
}
```

```cpp

inline void mul_OR(int *a, int *b, int n) {
    FWT_OR(a, n, 1); FWT_OR(b, n, 1);
    for(int i = 0;i < n; i++) a[i] = a[i] * b[i] % mod;
    FWT_OR(a, n, -1);
}

inline void mul_AND(int *a, int *b, int n) {
    FWT_AND(a, n, 1); FWT_AND(b, n, 1);
    for(int i = 0;i < n; i++) a[i] = a[i] * b[i] % mod;
    FWT_AND(a, n, -1);
}

ll quick_pow(ll a, ll b) ;

inline void mul_XOR(int *a, int *b, int n) {
    ll inv2 = quick_pow(mod, mod - 2);
    FWT_XOR(a, n, 1); FWT_XOR(b, n, 1);
    for(int i = 0;i < n; i++) a[i] = a[i] * b[i] % mod;
    FWT_XOR(a, n, inv2);
}

int main() {
    int n;
    cin >> n;
    n = 1 << n;
    for(int i = 0;i < n; i++) cin >> a[i];
    for(int i = 0;i < n; i++) cin >> b[i];

    mul_OR(a, b, n);
    mul_AND(a, b, n);
    mul_XOR(a, b, n);


}
```

6.55 cdq 分治 FFT

```cpp
// hdu 7054
// 求解(1+x^{a1})*(1+x^{a2})*...*(1+x^{an})
// \sum_{i=1}^n a_i <= 1e6.

// 可以f[i][j]，前i个数的和为j的方案数，可以用生成函数转换，并用多项式求解，同时分治FFT优化。

const int N = 1e5 + 10;
int tr[N];
int getLen(int n) ;

void FFT(Complex *A, int len) ;

inline void MTT(ll *x, ll *y, ll *z, int len) ;

struct Poly {
    ll *p;
    int len;
    void init(int len) {
        p = a + cnt;
        this -> len = len;
        for(int i = 0;i <= len; i++) p[i] = read();
        cnt += len + 1;
    }

    void mul(const Poly b) {
        static ll x[N], y[N];
        int LEN = getLen(len + b.len);
        for(int i = 0;i <= len; i++) x[i] = p[i];
        for(int i = 0;i <= b.len; i++) y[i] = b.p[i];
        for(int i = len + 1;i <= LEN; i++) x[i] = 0;
        for(int i = b.len + 1;i <= LEN; i++) y[i] = 0;
        MTT(x, y, p, LEN);
        this -> len += b.len;
        // 不知道为啥要两倍，可能会有不为0的情况，管他呢
        for(int i = len + 1;i <= 2 * LEN; i++) p[i] = 0;
        for(int i = 0;i <= LEN; i++) x[i] = y[i] = 0;
    }
};

Poly cdq(int l, int r) {
    Poly res;
    if(l == r) res.init(len); // 长度
    else {
        int mid = (l + r) / 2;
        res = cdq(l, mid);
        res.mul(cdq(mid + 1, r));
    }
    return res;
}

void solve() {
    mem(a, 0);
    int n = read();
    cnt = 0;
    ll ans = 1;
    Poly res = cdq(1, n);
    for(int i = 0;i < n; i++) cout << res.p[i] << " ";
}
```

6.56 求逆分治 FFT

```
// f[i] = \sum_{j=1}^i f[i-j] * g[j]
// g相同，可以用多项式求逆
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const int N = 1e5 + 10;

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];
```

```

ll mod;
int tr[N];
ll F[N], G[N];

ll quick_pow(ll a, ll b) ;

int getLen(int n) ;

void FFT(Complex *A, int len) ;

inline void MTT(ll *x, ll *y, ll *z, int len) ;

void Get_Inv(ll *f, ll *g, int n) ;

void fenzhiFFT(ll *f, ll *g, int n) {
    static ll a[N];
    for(int i = 1;i < n; i++) a[i] = (mod - f[i]) % mod;
    a[0] = 1;
    Get_Inv(a, g, n);

    for(int i = 0;i < n; i++) {
        a[i] = 0;
    }
}

int main() {
    int n;
    cin >> n;
    for(int i = 1;i < n; i++) cin >> G[i];
    fenzhiFFT(G, F, n);

    for(int i = 0;i < n; i++) cout << F[i] << " ";
}
```

6.57 多项式求逆

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const int N = 1e5 + 10;

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

ll mod;
int tr[N];
ll F[N], G[N];

ll quick_pow(ll a, ll b) ;
```

```

int getLen(int n) {
    int len = 1; while (len < (n << 1)) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 *
    PI * i / len));
    return len;
}

void FFT(Complex *A, int len) ;

inline void MTT(ll *x, ll *y, ll *z, int len) ;

void Get_Inv(ll *f, ll *g, int n) {
    if(n == 1) { g[0] = quick_pow(f[0], mod - 2); return ; }
    Get_Inv(f, g, (n + 1) >> 1);

    int len = getLen(n);
    static ll c[N];
    for(int i = 0;i < len; i++) c[i] = i < n ? f[i] : 0;
    MTT(c, g, c, len); MTT(c, g, c, len);
    for(int i = 0;i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
    for(int i = n;i < len; i++) g[i] = 0;
    for(int i = 0;i < len; i++) c[i] = 0;
}

int main() {
    int n;
    cin >> n;
    for(int i = 0;i < n; i++) cin >> F[i];
    Get_Inv(F, G, n);
    for(int i = 0;i < n; i++) cout << G[i] << " ";
}
```

6.58 多项式快速幂

```
// f(x)^k，k较小时，可取，每次FFT之后长度*2
#define maxfft 1048576+5

struct cp {
    double a, b;
    cp operator+(const cp &o) const { return (cp) {a + o.a, b + o.b}; }
    cp operator-(const cp &o) const { return (cp) {a - o.a, b - o.b}; }
    cp operator*(const cp &o) const { return (cp) {a * o.a - b * o.b, b * o.a + a * o.b
    }; }
    cp operator*(const double &o) const { return (cp) {a * o, b * o}; }
    cp operator!() const { return (cp) {a, -b}; }
} w[maxfft];

int pos[maxfft];

void fft_init(int len) {
    int j = 0;
    while ((1 << j) < len)j++;
    j--;
    for (int i = 0; i < len; i++)
        pos[i] = pos[i >> 1] >> 1 | ((i & 1) << j);
}
```

```cpp

void fft(cp *x, int len, int sta) {
    for (int i = 0; i < len; i++)
        if (i < pos[i])swap(x[i], x[pos[i]]);
    w[0] = (cp) {1, 0};
    for (unsigned i = 2; i <= len; i <<= 1) {
        cp g = (cp) {cos(2 * PI / i), sin(2 * PI / i) * sta};
        for (int j = i >> 1; j >= 0; j -= 2)w[j] = w[j >> 1];
        for (int j = 1; j < i >> 1; j += 2)w[j] = w[j - 1] * g;
        for (int j = 0; j < len; j += i) {
            cp *a = x + j, *b = a + (i >> 1);
            for (int l = 0; l < i >> 1; l++) {
                cp o = b[l] * w[l];
                b[l] = a[l] - o;
                a[l] = a[l] + o;
            }
        }
    }
    if (sta == -1)for (int i = 0; i < len; i++)x[i].a /= len, x[i].b /= len;
}

cp x[maxfft], y[maxfft], z[maxfft];

void FFT(int *a, int *b, int n, int m, int *c) {
    int len = 1;
    while (len < (n + m) >> 1)len <<= 1;
    fft_init(len);
    for (int i = n / 2; i < len; i++)x[i].a = x[i].b = 0;
    for (int i = m / 2; i < len; i++)y[i].a = y[i].b = 0;
    for (int i = 0; i < n; i++)(i & 1 ? x[i >> 1].b : x[i >> 1].a) = a[i];
    for (int i = 0; i < m; i++)(i & 1 ? y[i >> 1].b : y[i >> 1].a) = b[i];
    fft(x, len, 1), fft(y, len, 1);
    for (int i = 0; i < len / 2; i++) {
        int j = len - 1 & len - i;
        z[i] = x[i] * y[i] - (x[i] - !x[j]) * (y[i] - !y[j]) * (w[i] + (cp) {1, 0}) * 0.25;
    }
    for (int i = len / 2; i < len; i++) {
        int j = len - 1 & len - i;
        z[i] = x[i] * y[i] - (x[i] - !x[j]) * (y[i] - !y[j]) * ((cp) {1, 0} - w[i ^ len >> 1]) * 0.25;
    }
    fft(z, len, -1);
    for (int i = 0; i < n + m; i++)
        if (i & 1)c[i] = (int) (z[i >> 1].b + 0.5) ? 1 : 0;
        else c[i] = (int) (z[i >> 1].a + 0.5) ? 1 : 0;
}

int n, k, f[maxfft], g[maxfft];

void Pow(int *f, int len, int k, int *g) {
    g[0] = 1;
    while (k) {
        if (k & 1)FFT(g, f, len, len, g);
        FFT(f, f, len, len, f);
        k >>= 1;
        len <<= 1;
    }
}
```

6.59 多项式除法、取模

```cpp

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const int N = 3e5 + 10;
ll mod;

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

int tr[N];
ll F[N], G[N], D[N], R[N];

ll quick_pow(ll a, ll b) ;

int getLen(int n) ;

void FFT(Complex *A, int len) ;

inline void MTT(ll *x, ll *y, ll *z, int len) ;

void Get_Inv(ll *f, ll *g, int n) ;

void rever(ll *f, int n) { for(int i = 0, j = n - 1;i < j; i++, j--) swap(f[i], f[j]); }

void Get_Div(ll *f, ll *g, ll *d, ll *r, int n, int m) {
    static ll a[N], b[N], invb[N];
    for(int i = 0;i < n; i++) a[i] = f[i];
    for(int i = 0;i < m; i++) b[i] = g[i];
    rever(a, n); rever(b, m);
    //for(int i = 0;i < n - m + 1; i++) b[i] = i < m ? b[i] : 0;
    Get_Inv(b, invb, n - m + 1);

    int len = getLen(n);
    MTT(a, invb, a, len);
    rever(a, n - m + 1);
    for(int i = 0;i < len; i++) d[i] = i < n - m + 1 ? a[i] : 0;
    MTT(g, d, b, len);
    for(int i = 0;i < m; i++) { r[i] = (f[i] - b[i] + mod) % mod; }

    for(int i = m;i < len; i++) r[i] = 0;
    for(int i = 0;i < len; i++) a[i] = b[i] = invb[i] = 0;
}

int main() {
    int n, m;
    cin >> n >> m;
    for(int i = 0;i < n; i++) { cin >> F[i]; }
    for(int i = 0;i < m; i++) { cin >> G[i]; }
    Get_Div(F, G, D, R, n, m);

    for(int i = 0;i < n - m + 1; i++) cout << D[i] << " ";
    cout << endl;
    for(int i = 0;i < m - 1; i++) cout << R[i] << " ";
    cout << endl;
}
```

6.60 多项式 ln-Exp-Pow

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);

const int N = 1e6 + 10;

ll quick_pow(ll a, ll b) {
    ll ans = 1;
    while(b) {
        if(b & 1) ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans % mod;
}

const ll G = 3;
const ll invG = quick_pow(G, mod - 2);

int tr[N];
bool flag;

void NTT(ll *A, int len, int type) {
    for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int mid = i / 2;
        ll Wn = quick_pow(type == 1 ? G : invG, (mod - 1) / i);
        for (int k = 0; k < len; k += i) {
            ll w = 1;
            for (int l = k; l < k + mid; l++) {
                ll t = w * A[l + mid] % mod;
                A[l + mid] = (A[l] - t + mod) % mod;
                A[l] = (A[l] + t) % mod;
                w = w * Wn % mod;
            }
        }
    }
    if (type == -1) {
        ll invn = quick_pow(len, mod - 2);
        for (int i = 0; i < len; i++)
            A[i] = A[i] * invn % mod;
    }
}

void mul(ll *a, ll *b, int len) {
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    NTT(a, len, 1), NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
    NTT(a, len, -1); NTT(b, len, -1);
}

int getLen(int n) {
    int len = 1; while (len <= (n << 1)) len <<= 1;
    return len;
}

void Get_Der(ll *f, ll *g, int len) { for(int i = 1;i < len; i++) g[i - 1] = f[i] * i % mod; g[len - 1] = 0; }

void Get_Int(ll *f, ll *g, int len) { for(int i = 1;i < len; i++) g[i] = f[i - 1] * quick_pow(i, mod - 2) % mod; g[0] = 0; }

void Get_Inv(ll *f, ll *g, int n) {
    if(n == 1) { g[0] = quick_pow(f[0], mod - 2); return ; }
    Get_Inv(f, g, (n + 1) >> 1);

    int len = getLen(n);
    static ll c[N];
    for(int i = 0;i < len; i++) c[i] = i < n ? f[i] : 0;
    mul(c, g, len);
    mul(c, g, len);
    for(int i = 0;i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
    for(int i = n;i < len; i++) g[i] = 0;
    for(int i = 0;i < len; i++) c[i] = 0;
}

void Get_Ln(ll *f, ll *g, int n) {
    static ll a[N], b[N];
    Get_Der(f, a, n);
    Get_Inv(f, b, n);
    int len = getLen(n);
    mul(a, b, len);
    Get_Int(a, g, len);
    for(int i = n;i < len; i++) g[i] = 0;
    for(int i = 0;i < len; i++) a[i] = b[i] = 0;
}

void Get_Exp(ll *f, ll *g, int n) {
    if(n == 1) return (void)(g[0] = 1);
    Get_Exp(f, g, (n + 1) >> 1);

    static ll a[N];
    Get_Ln(g, a, n);
    a[0] = (f[0] + 1 - a[0] + mod) % mod;
    for(int i = 1;i < n; i++) a[i] = (f[i] - a[i] + mod) % mod;
    int len = getLen(n);
    mul(g, a, len);
    for(int i = n;i < len; i++) g[i] = 0;
    for(int i = 0;i < len; i++) a[i] = 0;
}

void Get_Pow(ll *f, ll *g, int n, ll k1, ll k2) {
    static ll a[N], b[N], c[N];
    ll deg = 0; for(int i = 0;i < n && f[i] == 0; i++) ++ deg;
    if(deg * k1 > n || (flag && deg)) return ;
    ll f0 = f[deg], f0k = quick_pow(f0, k2), inv0 = quick_pow(f0, mod - 2);
    for(int i = deg;i < n; i++) a[i - deg] = f[i] * inv0 % mod;
    Get_Ln(a, b, n);
    for(int i = 0;i < n - deg * k1; i++) b[i] = b[i] * k1 % mod;
    Get_Exp(b, c, n);
    deg *= k1;
    for(int i = deg;i < n; i++) g[i] = (c[i - deg] * f0k % mod + mod) % mod;
    for(int i = 0;i < deg; i++) g[i] = 0;
    int len = getLen(n);
    for(int i = n;i < len; i++) g[i] = 0;
    for(int i = 0;i < len; i++) a[i] = b[i] = c[i] = 0;
}


ll a[N], ans[N];

void solve() {
    int n; string s; cin >> n >> s;
    ll k1 = 0, k2 = 0;
    for(int i = 0; i < s.length(); i++) {
        k1 = (k1 * 10 + s[i] - '0');
        flag |= (k1 >= mod);
        k1 %= mod;
        k2 = (k2 * 10 + s[i] - '0') % (mod - 1);
    }
    for(int i = 0; i < n; i++) cin >> a[i];
    Get_Pow(a, ans, n, k1, k2); // k1是底 % mod，k2是指数 % mod-1
    for(int i = 0;i < n; i++) cout << ans[i] << (i == n - 1 ? endl : " ");
}
```

6.61 任意模数 MTT-拆系数法

```
//将多项式拆成(a1 * mod + a2) * (b1 * mod + b2)的形式
//=>a1 * b1 * mod ^ 2 + (a2 * b1 + a1 * b2) * mod + a2 * b2
//在利用DFT合并、IDFT合并，最终只需要4次DFT即可
//精度10^14
//4倍空间

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const int N = 1e5 + 10;

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

int tr[N];
ll a[N], b[N], ans[N];

int getLen(int n) {
    int len = 1; while (len <= n) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI * i / len));
    return len;
}

void FFT(Complex *A, int len) {
    for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2, lyc = len >> 1; i <= len; i <<= 1, lyc >>= 1)
        for (int j = 0; j < len; j += i) {
            Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
            for (int k = 0; k < i >> 1; k++) {
                Complex tmp = *r * *p;
                *r = *l - tmp, *l = *l + tmp;
                ++l, ++r, p += lyc;
            }
        }
}

inline void MTT(ll *x, ll *y, ll *z, int len) {
    for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
    static Complex a[N], b[N];
    static Complex dfta[N], dftb[N], dftc[N], dftd[N];

    for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
    for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
    FFT(a, len), FFT(b, len);
    for (int i = 0; i < len; i++) {
        int j = (len - i) & (len - 1);
        static Complex da, db, dc, dd;
        da = (a[i] + a[j].conj()) * Complex(0.5, 0);
        db = (a[i] - a[j].conj()) * Complex(0, -0.5);
        dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
        dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
        dfta[j] = da * dc;
        dftb[j] = da * dd;
        dftc[j] = db * dc;
        dftd[j] = db * dd;
    }
    for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
    for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
    FFT(a, len), FFT(b, len);
    for (int i = 0; i < len; i++) {
        ll da = (ll)(a[i].x / len + 0.5) % mod;
        ll db = (ll)(a[i].y / len + 0.5) % mod;
        ll dc = (ll)(b[i].x / len + 0.5) % mod;
        ll dd = (ll)(b[i].y / len + 0.5) % mod;
        z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
    }
}

int main() {

    int n, m;
    scanf("%d%d%lld", &n, &m, &mod);
    for (int i = 0; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 0; i <= m; i++) scanf("%d", &b[i]);

    MTT(a, b, ans, n + m);
    for (int i = 0; i <= n + m; i++)
        printf("%s%d", i == 0 ? "" : " ", (ans[i] + mod) % mod);

    return 0;
}
```

6.62 任意模数 NTT-三模数法

```
//要求选取的三个模数mod1 * mod2 * mod3 >= p^2*n
//优点是精度高，可达10^26
//缺点是常数大(9次NTT)，并且还使用了龟速乘
//4倍空间

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAX = 4e5 + 10;

ll qmul(ll a, ll b, ll mod) {
    ll res = 0;
    while (b) {
        if (b & 1)
            res = (res + a) % mod;
        a = (a << 1) % mod;
        b >>= 1;
    }
    return res;
}

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = qmul(res, a, mod);
        a = qmul(a, a, mod);
        b >>= 1;
    }
    return res;
}

const ll mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, mod4 = mod1 * mod2;
const ll G = 3;
ll a[3][MAX], b[3][MAX], ans[MAX], p;
int tr[MAX];

void NTT(ll *A, int len, int type, ll mod) {
    for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int mid = i / 2;
        ll Wn = qpow(type == 1 ? G : qpow(G, mod - 2, mod), (mod - 1) / i, mod);
        for (int k = 0; k < len; k += i) {
            ll w = 1;
            for (int l = k; l < k + mid; l++) {
                ll t = w * A[l + mid] % mod;
                A[l + mid] = (A[l] - t + mod) % mod;
                A[l] = (A[l] + t) % mod;
                w = w * Wn % mod;
            }
        }
    }
    if (type != 1) {
        ll invn = qpow(len, mod - 2, mod);
        for (int i = 0; i < len; i++) A[i] = A[i] * invn % mod;
    }
}

void mul(int i, int len, ll mod) {
    NTT(a[i], len, 1, mod), NTT(b[i], len, 1, mod);
    for (int j = 0; j < len; j++) a[i][j] = a[i][j] * b[i][j] % mod;
    NTT(a[i], len, -1, mod);
}

void CRT(int len) {
    ll inv1 = qpow(mod2, mod1 - 2, mod1);
    ll inv2 = qpow(mod1, mod2 - 2, mod2);
    ll inv3 = qpow(mod4 % mod3, mod3 - 2, mod3);
    for (int i = 0; i < len; i++) {
        ll t = 0;
        t = (t + qmul(a[0][i] * mod2 % mod4, inv1, mod4)) % mod4;
        t = (t + qmul(a[1][i] * mod1 % mod4, inv2, mod4)) % mod4;
        a[1][i] = t;
        t = (a[2][i] - a[1][i] % mod3 + mod3) % mod3 * inv3 % mod3;
        ans[i] = (mod4 % p * t % p + a[1][i] % p) % p;
    }
}

void doNTT(int n) {
    int len = 1; while (len <= n) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    mul(0, len, mod1), mul(1, len, mod2), mul(2, len, mod3);
    CRT(len);
}

int main() {

    int n, m;
    scanf("%d%d%lld", &n, &m, &p);
    for (int i = 0; i <= n; i++) {
        ll x; scanf("%lld", &x);
        a[0][i] = a[1][i] = a[2][i] = x % p;
    }
    for (int i = 0; i <= m; i++) {
        ll x; scanf("%lld", &x);
        b[0][i] = b[1][i] = b[2][i] = x % p;
    }
    doNTT(n + m);
    for (int i = 0; i <= n + m; i++) printf("%lld ", ans[i]);

    return 0;
}
```

6.63 多项式优化常系数齐次线性递推

```

#include <bits/stdc++.h>
using namespace std;
```

```cpp
typedef long long ll;
const double PI = acos(-1);
const int N = 3e5 + 10;

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

ll mod;
int n, k, len, tr[N];
ll a[N], h[N], ans[N], s[N], invG[N], G[N];

ll quick_pow(ll a, ll b) ;

int getLen(int n) ;

void rever(ll *f, int n) ;

void FFT(Complex *A, int len) ;

inline void MTT(ll *x, ll *y, ll *z, int len) ;

void Get_Inv(ll *f, ll *g, int n) ;


void Mod(ll *f,ll *g) {
    static ll tmp[N];
    rever(f, k + k - 1);
    for(int i = 0;i < k; i++) tmp[i] = f[i];
    MTT(tmp, invG, tmp, len);
    for(int i = k - 1; i < len; i++) tmp[i] = 0;
    rever(f, k + k - 1); rever(tmp, k - 1);
    MTT(tmp, G, tmp, len);
    for(int i = 0;i < k; i++) g[i] = (f[i] + mod - tmp[i]) % mod;
    for(int i = k;i < len; i++) g[i] = 0;
    for(int i = 0;i < len; i++) tmp[i] = 0;
}
void fpow(int b) {
    s[1] = 1; ans[0] = 1;
    while(b) {
        if(b & 1) { MTT(ans, s, ans, len); Mod(ans, ans); }
        MTT(s, s, s, len); Mod(s, s);
        b >>= 1;
    }
}

ll DITI(ll *a, ll *h, ll *ans, int n, int k) {
    G[k] = 1; for(int i = 1;i <= k; i++) G[k - i] = (mod - a[i]) % mod;
    rever(G, k + 1);
    len = getLen(k + 1);
    Get_Inv(G, invG, k + 1);
    for(int i = k + 1;i < len; i++) invG[i] = 0;
    rever(G, k + 1);
    fpow(n);
    ll Ans = 0;
    for(int i = 0;i < k; i++) Ans = (Ans + 1ll * h[i] * ans[i] % mod) % mod;
    return Ans;
}

int main() {
    int n, k;
    cin >> n >> k;
    for(int i = 1;i <= k; i++){ cin >> a[i]; a[i] = a[i] < 0 ? a[i] + mod : a[i]; }
    for(int i = 0;i < k; i++) { cin >> h[i]; h[i] = h[i] < 0 ? h[i] + mod : h[i]; }

    ll Ans = DITI(a, h, ans, n, k);
    cout << Ans << endl;
}
```

6.64 FFT 加速带有通配符字符串匹配

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

// p[x] = \sum_{i=0}^{m-1} A[i]^3 * B[x-m+i+1] + \sum_{i=0}^{m-1} A[i] * B[x-m+i+1]^3 - 2 * \sum_{i=0}^{m-1} A[i]^2 * B[x-m+i+1]^2

const int N = 1e6 + 1e5;

ll qpow(ll a, ll b, ll mod) {
    ll ans = 1;
    while(b) {
        if(b & 1) ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans % mod;
}

const ll G = 3;
const ll invG = qpow(G, mod - 2, mod);
int tr[N];

void NTT(ll *A, int len, int type) {
    for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int mid = i / 2;
        ll Wn = qpow(type == 1 ? G : invG, (mod - 1) / i, mod);
        for (int k = 0; k < len; k += i) {
            ll w = 1;
            for (int l = k; l < k + mid; l++) {
                ll t = w * A[l + mid] % mod;
                A[l + mid] = (A[l] - t + mod) % mod;
                A[l] = (A[l] + t) % mod;
                w = w * Wn % mod;
            }
        }
    }
    if (type == -1) {
        ll invn = qpow(len, mod - 2, mod);
        for (int i = 0; i < len; i++)
            A[i] = A[i] * invn % mod;
    }
}

void mul(ll *a, ll *b, int n) {
    int len = 1; while (len <= n) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    NTT(a, len, 1), NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
    NTT(a, len, -1);
}

ll a1[N], a2[N], a3[N], b1[N], b2[N], b3[N];

void solve() {
    int m, n; cin >> m >> n;
    string s, t; cin >> t >> s;
    for(int i = 0;i < m; i++) {
        if(t[i] == '*') continue ;
        int temp = t[i] - 'a' + 1;
        a1[i] = temp;
        a2[i] = temp * temp;
        a3[i] = temp * temp * temp;
    }
    for(int i = 0;i < n; i++) {
        if(s[i] == '*') continue;
        int temp = s[i] - 'a' + 1;
        b1[i] = temp;
        b2[i] = temp * temp;
        b3[i] = temp * temp * temp;
    }
    reverse(a1, a1 + m);
    reverse(a2, a2 + m);
    reverse(a3, a3 + m);
    mul(a1, b3, n + m);
    mul(a2, b2, n + m);
    mul(a3, b1, n + m);
    vector<int> ans;
    for(int x = m - 1;x < n; x++) {
        ll res = a1[x] + a3[x] - a2[x] * 2;
        if(!res) ans.push_back(x - m + 2);
    }
    cout << ans.size() << endl;
    for(int i = 0;i < ans.size(); i++) cout << ans[i] << (i == ans.size() - 1 ? endl : " ");
}
```

6.65 FFT 加速朴素字符串匹配

```
#include <bits/stdc++.h>
using namespace std;
const int N = 4e5 + 10;

// P[x] = \sum_{i=0}^{m-1} A[i] + \sum_{i=0}^{m-1} B[x - m + i + 1] - 2 * \sum_{i=0}^{m-1}A[i] * B[x - m + i + 1]

// reverse(a)
```

```cpp

// 当串中的字符集较少时，可以针对每个字符进行FFT，计算每个字符对整个串的贡献

ll qpow(ll a, ll b, ll mod) ;

const ll mod = 998244353;
const ll G = 3;
const ll invG = qpow(G, mod - 2, mod);
int tr[N];

void NTT(ll *A, int len, int type) {
    for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int mid = i / 2;
        ll Wn = qpow(type == 1 ? G : invG, (mod - 1) / i, mod);
        for (int k = 0; k < len; k += i) {
            ll w = 1;
            for (int l = k; l < k + mid; l++) {
                ll t = w * A[l + mid] % mod;
                A[l + mid] = (A[l] - t + mod) % mod;
                A[l] = (A[l] + t) % mod;
                w = w * Wn % mod;
            }
        }
    }
    if (type == -1) {
        ll invn = qpow(len, mod - 2, mod);
        for (int i = 0; i < len; i++)
            A[i] = A[i] * invn % mod;
    }
}

void mul(ll *a, ll *b, int n) {
    int len = 1; while (len <= n) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    NTT(a, len, 1), NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
    NTT(a, len, -1);
}

ll a[N], b[N];

void solve() {
    string s, t; cin >> s >> t;
    int n = s.length(), m = t.length();
    for(int i = 0;i < n; i++) a[i] = s[i] - 'a' + 1;
    for(int i = 0;i < m; i++) b[i] = t[i] - 'a' + 1;
    reverse(b, b + m);
    mul(a, b, n + m - 2);
    double P = 0;
    for(int i = 0;i < m; i++) {
        P += (t[i] - 'a' + 1) * (t[i] - 'a' + 1);
    }
    vector<int> f(n + 1);
    for(int i = 1;i < n; i++) {
        f[i] = f[i - 1] + (s[i] - 'a' + 1) * (s[i] - 'a' + 1);
    }
    for(int x = m - 1;x < n; x++) {
        double res;
        if(x == m - 1) res = P + f[x] - a[x] * 2;
        else res = P + f[x] - f[x - m] - a[x] * 2;
        if(!res) cout << x - m + 2 << endl;
    }
}
```

6.66 x 不连续、暴力插值

```

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const int N = 3e5 + 10;

ll mod;
ll X[N], Y[N];

ll quick_pow(ll a, ll b) ;

ll Lagrange(ll *x, ll *y, int n, int k) {
    ll ans = 0;
    for(int i = 0;i < n; i++) {
        ll s1 = 1, s2 = 1;
        for(int j = 0;j < n; j++) {
            if(i == j) continue;
            s1 = s1 * (k - x[j] + mod) % mod;
            s2 = s2 * (x[i] - x[j] + mod) % mod;
        }
        ans = (ans + 1ll * y[i] * s1 % mod * quick_pow(s2, mod - 2) % mod) % mod;
    }
    return ans;
}

int main() {
    int n, k;
    cin >> n >> k;
    for(int i = 0;i < n; i++) cin >> X[i] >> Y[i];
    cout << Lagrange(X, Y, n, k) << endl;
}
```

6.67 x 连续、前缀优化

```

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 10;

ll mod;
ll F[N];
ll pre[N], suf[N];
ll fac[N], invf[N];


ll quick_pow(ll a, ll b) ;

```

```
void init() {
    fac[0] = 1;
    for(int i = 1;i < N; i++) fac[i] = fac[i - 1] * i % mod;
    invf[N - 1] = quick_pow(fac[N - 1], mod - 2);
    for(int i = N - 1;i >= 1; i--) invf[i - 1] = invf[i] * i % mod;
}

ll Lagrange(ll *f, int k, int n) {
    if(k <= n) return f[k];
    pre[0] = suf[n] = 1;
    for(int i = 1;i <= n; i++) pre[i] = pre[i - 1] * (k - i + 1) % mod;
    for(int i = n;i >= 1; i--) suf[i - 1] = suf[i] * (k - i) % mod;
    ll ans = 0;
    for(int i = 0;i <= n; i++) {
        int opt = (n - i) & 1 ? -1 : 1;
        ans = (ans + 1ll * opt * pre[i] % mod * suf[i] % mod * invf[i] % mod * invf[n - i] % mod * f[i] % mod + mod) % mod;
    }
    return f[k] = ans;
}

int main() {
    init();
    int n, k;
    cin >> n >> k;
    for(int i = 0;i <= n; i++) cin >> F[i];
    cout << Lagrange(F, k, n) << endl;

}
```

6.68 多项式 ln-exp-pow 处理边界为 1

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const int N = 1e5 + 10;

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

ll mod, inv2;
int tr[N];
ll F[N], G[N];

ll quick_pow(ll a, ll b) ;

int getLen(int n) ;

void FFT(Complex *A, int len) ;

```

```
inline void MTT(ll *x, ll *y, ll *z, int len) ;

void Get_Inv(ll *f, ll *g, int n) ;

void Get_Der(ll *f, ll *g, int len) { for(int i = 1;i < len; i++) g[i - 1] = f[i] * i %
     mod; g[len - 1] = 0; }

void Get_Int(ll *f, ll *g, int len) { for(int i = 1;i < len; i++) g[i] = f[i - 1] *
    quick_pow(i, mod - 2) % mod; g[0] = 0; }

void Get_Ln(ll *f, ll *g, int n) ;

void Get_Exp(ll *f, ll *g, int n) ;

void Get_Pow(ll *f, ll *g, int n, ll k) ;

void Get_Sqrt(ll *f, ll *g, int n) {
    static ll a[N];
    Get_Ln(f, a, n);
    for(int i = 0;i < n; i++) a[i] = a[i] * inv2 % mod;
    Get_Exp(a, g, n);
    int len = getLen(n);
    for(int i = n;i < len; i++) g[i] = 0;
    for(int i = 0;i < len; i++) a[i] = 0;
}
```

6.69 二次剩余处理边界不为 1

```

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const int N = 1e5 + 10;


struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x *
     rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

ll mod, inv2;
int tr[N];
ll F[N], G[N];

ll quick_pow(ll a, ll b) ;

typedef struct{
    ll x, y; // 把求出来的w作为虚部，则为a + bw
}num;

num num_mul(num a, num b, ll w, ll p) {// 复数乘法
    num ans = {0, 0};
    ans.x = (a.x * b.x % p + a.y * b.y % p * w % p + p) % p;
    ans.y = (a.x * b.y % p + a.y * b.x % p + p) % p;
    return ans;
}

ll num_pow(num a, ll b, ll w, ll p) { // 复数快速幂
    num ans = {1, 0};
    while(b) {
        if(b & 1)
            ans = num_mul(ans, a, w, p);
        a = num_mul(a, a, w, p);
        b >>= 1;
    }
    return ans.x % p;
}

ll legendre(ll a, ll p) { // 勒让德符号 = {1, -1, 0}
    return quick_pow(a, (p - 1) >> 1);
}

ll Cipolla(ll n, ll p) {// 输入a和p，是否存在x使得x^2 = a (mod p)，存在二次剩余返回x，存在二次非剩余返回-1      注意：p是奇质数
    n %= p;
    if(n == 0)
        return 0;
    if(p == 2)
        return 1;
    ll a, w;

    while(true) {// 找出a，求出w，随机成功的概率是50%，所以数学期望是2
        a = rand() % p;
        w = ((a * a - n) % p + p) % p;
        if(legendre(w, p) + 1 == p) // 找到w，非二次剩余条件
            break;
    }
    num x = {a, 1};
    return num_pow(x, (p + 1) >> 1, w, p) % p; // 计算x,一个解是x，另一个解是p-x，这里的w其实要开方，但是由拉格朗日定理可知虚部为0，所以最终答案就是对x的实部用快速幂求解
}

int getLen(int n) ;

void FFT(Complex *A, int len) ;

inline void MTT(ll *x, ll *y, ll *z, int len) ;

void Get_Inv(ll *f, ll *g, int n) ;

void Get_Sqrt(ll *f, ll *g, int n) {
    if(n == 1) { ll t = Cipolla(f[0], mod); g[0] = min(mod - t, t); return ; }
    Get_Sqrt(f, g, (n + 1) >> 1);

    int len = getLen(n);
    static ll c[N], invg[N];
    for(int i = 0;i < len; i++) c[i] = i < n ? f[i] : 0;
    Get_Inv(g, invg, n);
    MTT(c, invg, c, len);
    for(int i = 0;i < n; i++) g[i] = inv2 * (c[i] + g[i]) % mod;
    for(int i = n;i < len; i++) g[i] = 0;
    for(int i = 0;i < len; i++) c[i] = invg[i] = 0;
}

int main() {
    inv2 = quick_pow(2, mod - 2);
    int n;
    cin >> n;
    for(int i = 0;i < n; i++) cin >> F[i];
    Get_Sqrt(F, G, n);
    for(int i = 0;i < n; i++) cout << G[i] << " ";
}
```

6.70 二维几何

```
#include <iostream>
#include <cmath>

using namespace std;

const double eps = 1e-6;
const double pi = acos(-1);

#define zero(x) (((x) > 0 ? (x) : -(x)) < eps)

int sgn(double d) {
    if(fabs(d) < eps)
        return 0;
    if(d > 0)
        return 1;
    else
        return -1;
}

int dcmp(double x, double y) {
    if(fabs(x - y) < eps)
        return 0;
    if(x > y)
        return 1;
    else
        return -1;
}

struct Point{ // 点
    double x, y;
    Point(double x = 0, double y = 0) : x(x), y(y) {}
};

struct line{
    Point a, b;
};

typedef Point Vector; // 向量

// 运算(向量之间)

Vector operator + (Vector A, Vector B) { // AB
    return Vector(A.x + B.x, A.y + B.y);
}
```

```cpp

Vector operator - (Point A, Point B) { // BA
    return Vector(A.x - B.x, A.y - B.y);
}

Vector operator * (Vector A, double p) { // A * p
    return Vector(A.x * p, A.y * p);
}

Vector operator / (Vector A, double p) { // A / p
    return  Vector(A.x / p, A.y / p);
}

bool operator < (const Point& a, const Point& b) { // 将点升序排列
    if(a.x == b.x)
        return a.y < b.y;
    return a.x < b.x;
}

bool operator == (const Point& a, const Point& b) { // 判断是否为同一点
    if(dcmp(a.x, b.x) == 0 && dcmp(a.y, b.y) == 0)
        return true;
    else
        return false;
}

/*
    -------------------------------------------------------------------------------------------------
    */
// 向量

double Dot(Vector A, Vector B) { // 内积
    return A.x * B.x + A.y * B.y;
}

double Cross(Vector A, Vector B) { // 外积
    return A.x * B.y - A.y * B.x;
}

double Length(Vector A) { // 向量取模
    return sqrt(Dot(A, A));
}

double Angle(Vector A, Vector B) { // 向量夹角
    return acos(Dot(A, B) / Length(A) / Length(B));
}

double Area(Point A, Point B, Point C) { // 计算两向量构成的平行四边形有向面积
    return Cross(B - A, C - A);
}

Vector Rotate(Vector A, double rad) { // 计算向量逆时针旋转后的向量
    return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
}

Vector Normal(Vector A) { // 计算向量逆时针转90度后的单位法向量
    double L = Length(A);
    return Vector(-A.y / L, A.x / L);
}
```

```

bool ToLeftTest(Point a, Point b, Point c) { // 判断bc是不是向ab的逆时针方向转向
    return Cross(b - a, c - b) > 0;
}

/* -------------------------------------------------------------------------------------------------- */
// 直线与线段

double Pow(double x) {
    return x * x;
}

double distance (Point p1, Point p2) {// 两点距离
    return sqrt(Pow(p1.x - p2.x) + Pow(p1.y - p2.y));
}

int dots_inline(Point p1, Point p2, Point p3) { // 判断三点共线
    return Cross(p2 - p1, p3 - p1);
}

int dot_online_in(Point p, line l) { // 判断点在线段上（包含端点）
    return zero(Cross(l.b - p, l.a - p) && ((l.a.x - p.x) * (l.b.x - p.x) < eps) && ((l.a.y - p.y) * (l.b.y - p.y) < eps));
}

int dot_online_ex(Point p, line l) { // 判断点在线段上（不包含端点）
    return dot_online_in(p, l) && (!zero(p.x - l.a.x) || !zero(p.y - l.a.y)) && (!zero(p.x - l.b.x) || !zero(p.y - l.b.y));
}

int same_side(Point p1, Point p2, line l) { // 判断两点在线段同侧，点在线段上返回0
    return Cross(l.a - l.b, p1 - l.b) * Cross(l.a - l.b, p2 - l.b) > eps;
}

int opposite_side(Point p1, Point p2, line l) { // 判断两点在线段异侧，点在线段上返回0
    return Cross(l.a - l.b, p1 - l.b) * Cross(l.a - l.b, p2 - l.b) < -eps;
}

int parallel(line u, line v) { // 判断两直线平行
    return zero((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
}

int  perpendicular(line u, line v) { // 判断两直线垂直
    return zero((u.a.x - u.b.x) * (v.a.x - v.b.x) + (u.a.y - u.b.y) * (v.a.y - v.b.y));
}

int intersect_in(line u, line v) {// 判断两线段相交，包括端点和部分重合
    if(!dots_inline(u.a, u.b, v.a) || !dots_inline(u.a, u.b, v.b)) {
        return !same_side(u.a, u.b, v) && !same_side(v.a, v.b, u);
    }
    return dot_online_in(u.a, v) || dot_online_in(u.b, v) || dot_online_in(v.a, u) || dot_online_in(v.b, u);
}

int intersect_ex(line u, line v) {// 判断两线段相交，不包括端点和部分重合
    return opposite_side(u.a, u.b, v) && opposite_side(v.a, v.b, u);
}
```

```

// 计算两直线交点，注意事先判断直线是否相交
// 计算两线段交点，注意事先判断线段相交和平行
Point intersection(line u, line v) {
    Point ret = u.a;
    double t = ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) * (v.a.x - v.b.x))
    / ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
    ret.x += (u.b.x - u.a.x) * t;
    ret.y += (u.b.y - u.a.y) * t;
    return ret;
}

Point ptoline(Point p, line l) { // 点到直线最近点
    Point t = p;
    t.x += l.a.y - l.b.y;
    t.y += l.b.x - l.a.x;
    line u = {p, t};
    return intersection(u, l);
}

double disptoline(Point p, line l) { // 点到直线距离
    return fabs(Cross(p - l.b, l.a - l.b) / distance(l.a, l.b));
}

Point ptoseg(Point p, line l) { // 点到线段最近点
    Point t = p;
    t.x += l.a.y - l.b.y;
    t.y += l.b.x - l.a.x;
    if(Cross(l.a - p, t - p) * Cross(l.b - p, t - p) > eps)
        return distance(p, l.a) < distance(p, l.b) ? l.a : l.b;
    line u = {p, t};
    return intersection(u, l);
}

double disptoseg(Point p, line l) { // 点到线段距离
    Point t = p;
    t.x += l.a.y - l.b.y;
    t.y += l.b.x - l.a.x;
    if(Cross(l.a - p, t - p) * Cross(l.b - p, t - p) > eps) {
        double dis1 = distance(p, l.a);
        double dis2 = distance(p, l.b);
        return dis1 < dis2 ? dis1 : dis2;
    }
    return fabs(Cross(p - l.b, l.a - l.b) / distance(l.a, l.b));
}

/*
    -------------------------------------------------------------------------------------------------
    */
// 面积

double area_triangle(Point p1, Point p2, Point p3) { // 三角形面积（输入三顶点）
    return fabs(Cross(p1 - p3, p2 - p3)) / 2;
}

double area_triangle(double a, double b, double c) { // 三角形面积（输入三边长）
    double s = (a + b + c) / 2;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}
```

```

double area_polygon(int n, Point *p) { // 计算多边形面积，顶点按顺时针或逆时针输入
    double s1 = 0, s2 = 0;
    for(int i = 0;i < n; i++) {
        s1 += p[(i + 1) % n].y * p[i].x;
        s2 += p[(i + 1) % n].y * p[(i + 2) % n].x;
    }
    return fabs(s1 - s2) / 2;
}

/*
    -------------------------------------------------------------------------------------------------
    */
// 球面

//计算圆心角 lat 表示纬度,-90<=w<=90,lng 表示经度
//返回两点所在大圆劣弧对应圆心角,0<=angle<=pi

double angle(double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * pi / 180;
    while(dlng >= pi + pi) {
        dlng -= pi + pi;
    }
    if(dlng > pi)
    dlng = pi + pi - dlng;
    lat1 *= pi / 180;
    lat2 *= pi / 180;
    return acos(cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2));
}

// 计算两点距离

double line_dist(double r, double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * pi / 180;
    while(dlng >= pi + pi) {
        dlng -= pi + pi;
    }
    if(dlng > pi)
    dlng = pi + pi - dlng;
    lat1 *= pi / 180;
    lat2 *= pi / 180;
    return r * sqrt(2 - 2 * (cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2))
    );
}

// 计算球面距离

inline double sphere_dist(double r, double lng1, double lat1, double lng2, double lat2)
     {
    return r * angle(lng1, lat1, lng2, lat2);
}

/*
    -------------------------------------------------------------------------------------------------
    */
// 三角形

// 外心

```

```
Point circumcenter(Point a, Point b, Point c) {
    line u, v;
    u.a.x = (a.x + b.x) / 2;
    u.a.y = (a.y + b.y) / 2;
    u.b.x = u.a.x - a.y + b.y;
    u.b.y = u.a.y + a.x - b.x;
    v.a.x = (a.x + c.x) / 2;
    v.a.y = (a.y + c.y) / 2;
    v.b.x = v.a.x - a.y + c.y;
    v.b.y = v.a.y + a.x - c.y;
    return intersection(u, v);
}

// 内心

Point incenter(Point a, Point b, Point c) {
    line u, v;
    double m, n;
    u.a = a;
    m = atan2(b.y - a.y, b.x - a.x);
    n = atan2(c.y - a.y, c.x - a.x);
    u.b.x = u.a.x + cos((m + n) / 2);
    u.b.y = u.a.y + sin((m + n) / 2);
    v.a = b;
    m=atan2(a.y - b.y, a.x - b.x);
    n=atan2(c.y - b.y, c.x - b.x);
    v.b.x=v.a.x + cos((m + n) / 2);
    v.b.y=v.a.y + sin((m + n) / 2);
    return intersection(u, v);
}

// 垂心

Point perpencenter(Point a, Point b, Point c) {
    line u, v;
    u.a = c;
    u.b.x = u.a.x - a.y + b.y;
    u.b.y = u.a.y + a.x - b.x;
    v.a = b;
    v.b.x = v.a.x - a.y + c.y;
    v.b.y = v.a.y + a.x - c.x;
return intersection(u, v);
}

// 重心
//到三角形三顶点距离的平方和最小的点
//三角形内到三边距离之积最大的点

Point barycenter(Point a, Point b, Point c) {
    line u,v;
    u.a.x = (a.x + b.x) / 2;
    u.a.y = (a.y + b.y) / 2;
    u.b = c;
    v.a.x = (a.x + c.x) / 2;
    v.a.y = (a.y + c.y) / 2;
    v.b = b;
return intersection(u, v);
}

```

```
//费马点
//到三角形三顶点距离之和最小的点
Point fermentpoint(Point a, Point b, Point c) {
    Point u,v;
    double step = fabs(a.x) + fabs(a.y) + fabs(b.x) + fabs(b.y) + fabs(c.x) + fabs(c.y);
    int i, j, k;
    u.x = (a.x + b.x + c.x) / 3;
    u.y = (a.y + b.y + c.y) / 3;
    while(step > 1e-10)
    for(k = 0;k < 10; step /= 2, k++)
        for (i = -1;i <= 1; i++)
            for (j = -1;j <= 1; j++){
            v.x = u.x + step * i;
            v.y = u.y + step * j;
            if(distance(u,a) + distance(u,b) + distance(u,c) > distance(v,a) + distance(v,b) + distance(v,c))
                u = v;
            }
    return u;
}
```

6.71 三维几何

```
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point3{double x,y,z;};
struct line3{point3 a,b;};
struct plane3{point3 a,b,c;};
//计算 cross product U x V
point3 Cross(point3 u,point3 v){
    point3 ret;
    ret.x=u.y*v.z-v.y*u.z;
    ret.y=u.z*v.x-u.x*v.z;
    ret.z=u.x*v.y-u.y*v.x;
    return ret;
}
//计算 dot product U . V
double Dot(point3 u,point3 v){
    return u.x*v.x+u.y*v.y+u.z*v.z;
}
//矢量差 U - V
point3 subt(point3 u,point3 v){
    point3 ret;
    ret.x=u.x-v.x;
    ret.y=u.y-v.y;
    ret.z=u.z-v.z;
    return ret;
}
//取平面法向量
point3 pvec(plane3 s){
    return Cross(subt(s.a,s.b),subt(s.b,s.c));
}
point3 pvec(point3 s1,point3 s2,point3 s3){
    return Cross(subt(s1,s2),subt(s2,s3));
}
//两点距离,单参数取向量大小
```

```
double distance(point3 p1,point3 p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-p2.z));
}
//向量大小\
double vlen(point3 p){
    return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}
//判三点共线
int dots_inline(point3 p1,point3 p2,point3 p3){
    return vlen(Cross(subt(p1,p2),subt(p2,p3)))<eps;
}
//判四点共面
int dots_onplane(point3 a,point3 b,point3 c,point3 d){
    return zero(Dot(pvec(a,b,c),subt(d,a)));
}
//判点是否在线段上,包括端点和共线
int dot_online_in(point3 p,line3 l){
    return zero(vlen(Cross(subt(p,l.a),subt(p,l.b))))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&
           (l.a.y-p.y)*(l.b.y-p.y)<eps&&(l.a.z-p.z)*(l.b.z-p.z)<eps;
}
int dot_online_in(point3 p,point3 l1,point3 l2){
    return zero(vlen(Cross(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)*(l2.x-p.x)<eps&&
           (l1.y-p.y)*(l2.y-p.y)<eps&&(l1.z-p.z)*(l2.z-p.z)<eps;
}
//判点是否在线段上,不包括端点
int dot_online_ex(point3 p,line3 l){
    return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y)||!zero(p.z-l.a.z))&&
            (!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));
}
int dot_online_ex(point3 p,point3 l1,point3 l2){
    return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y)||!zero(p.z-l1.z))&&
           (!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
}
//判点是否在空间三角形上,包括边界,三点共线无意义
int dot_inplane_in(point3 p,plane3 s){
    return zero(vlen(Cross(subt(s.a,s.b),subt(s.a,s.c)))-vlen(Cross(subt(p,s.a),subt(p,s.b)))-
                vlen(Cross(subt(p,s.b),subt(p,s.c)))-vlen(Cross(subt(p,s.c),subt(p,s.a))));
}
int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3){
    return zero(vlen(Cross(subt(s1,s2),subt(s1,s3)))-vlen(Cross(subt(p,s1),subt(p,s2)))-
                vlen(Cross(subt(p,s2),subt(p,s3)))-vlen(Cross(subt(p,s3),subt(p,s1))));
}
//判点是否在空间三角形上,不包括边界,三点共线无意义
int dot_inplane_ex(point3 p,plane3 s){
    return dot_inplane_in(p,s)&&vlen(Cross(subt(p,s.a),subt(p,s.b)))>eps&&
           vlen(Cross(subt(p,s.b),subt(p,s.c)))>eps&&vlen(Cross(subt(p,s.c),subt(p,s.a)))>eps;
}
int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3){
    return dot_inplane_in(p,s1,s2,s3)&&vlen(Cross(subt(p,s1),subt(p,s2)))>eps&&
           vlen(Cross(subt(p,s2),subt(p,s3)))>eps&&vlen(Cross(subt(p,s3),subt(p,s1)))>eps;
}
//判两点在线段同侧,点在线段上返回 0,不共面无意义
```

```
int same_side(point3 p1,point3 p2,line3 l){
    return Dot(Cross(subt(l.a,l.b),subt(p1,l.b)),Cross(subt(l.a,l.b),subt(p2,l.b)))>eps;
}
int same_side(point3 p1,point3 p2,point3 l1,point3 l2){
    return Dot(Cross(subt(l1,l2),subt(p1,l2)),Cross(subt(l1,l2),subt(p2,l2)))>eps;
}
//判两点在线段异侧,点在线段上返回 0,不共面无意义
int opposite_side(point3 p1,point3 p2,line3 l){
    return Dot(Cross(subt(l.a,l.b),subt(p1,l.b)),Cross(subt(l.a,l.b),subt(p2,l.b)))<-eps;
}
int opposite_side(point3 p1,point3 p2,point3 l1,point3 l2){
    return Dot(Cross(subt(l1,l2),subt(p1,l2)),Cross(subt(l1,l2),subt(p2,l2)))<-eps;
}
//判两点在平面同侧,点在平面上返回 0
int same_side(point3 p1,point3 p2,plane3 s){
    return Dot(pvec(s),subt(p1,s.a))*Dot(pvec(s),subt(p2,s.a))>eps;
}
int same_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
    return Dot(pvec(s1,s2,s3),subt(p1,s1))*Dot(pvec(s1,s2,s3),subt(p2,s1))>eps;
}
//判两点在平面异侧,点在平面上返回 0
int opposite_side(point3 p1,point3 p2,plane3 s){
    return Dot(pvec(s),subt(p1,s.a))*Dot(pvec(s),subt(p2,s.a))<-eps;
}
int opposite_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
    return Dot(pvec(s1,s2,s3),subt(p1,s1))*Dot(pvec(s1,s2,s3),subt(p2,s1))<-eps;
}
//判两直线平行
int parallel(line3 u,line3 v){
    return vlen(Cross(subt(u.a,u.b),subt(v.a,v.b)))<eps;
}
int parallel(point3 u1,point3 u2,point3 v1,point3 v2){
    return vlen(Cross(subt(u1,u2),subt(v1,v2)))<eps;
}
//判两平面平行
int parallel(plane3 u,plane3 v){
    return vlen(Cross(pvec(u),pvec(v)))<eps;
}
int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return vlen(Cross(pvec(u1,u2,u3),pvec(v1,v2,v3)))<eps;
}
//判直线与平面平行
int parallel(line3 l,plane3 s){
    return zero(Dot(subt(l.a,l.b),pvec(s)));
}
int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return zero(Dot(subt(l1,l2),pvec(s1,s2,s3)));
}
//判两直线垂直
int perpendicular(line3 u,line3 v){
    return zero(Dot(subt(u.a,u.b),subt(v.a,v.b)));
}
int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2){
    return zero(Dot(subt(u1,u2),subt(v1,v2)));
}
//判两平面垂直
int perpendicular(plane3 u,plane3 v){
    return zero(Dot(pvec(u),pvec(v)));
}
int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return zero(Dot(pvec(u1,u2,u3),pvec(v1,v2,v3)));
}
//判直线与平面平行
int perpendicular(line3 l,plane3 s){
    return vlen(Cross(subt(l.a,l.b),pvec(s)))<eps;
}
int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return vlen(Cross(subt(l1,l2),pvec(s1,s2,s3)))<eps;
}
//判两线段相交,包括端点和部分重合
int intersect_in(line3 u,line3 v){
    if (!dots_onplane(u.a,u.b,v.a,v.b))
        return 0;
    if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
        return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
    return dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b,u);
}
int intersect_in(point3 u1,point3 u2,point3 v1,point3 v2){
    if (!dots_onplane(u1,u2,v1,v2))
        return 0;
    if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
        return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
    return
            dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||dot_online_in(v2,u1,u2);
}
//判两线段相交,不包括端点和部分重合
int intersect_ex(line3 u,line3 v){
    return dots_onplane(u.a,u.b,v.a,v.b)&&opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2){
    return  dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
  }
//判线段与空间三角形相交,包括交于边界和(部分)包含
int intersect_in(line3 l,plane3 s){
    return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&
           !same_side(s.b,s.c,l.a,l.b,s.a)&&!same_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return !same_side(l1,l2,s1,s2,s3)&&!same_side(s1,s2,l1,l2,s3)&&
           !same_side(s2,s3,l1,l2,s1)&&!same_side(s3,s1,l1,l2,s2);
}
//判线段与空间三角形相交,不包括交于边界和(部分)包含
int intersect_ex(line3 l,plane3 s){
    return opposite_side(l.a,l.b,s)&&opposite_side(s.a,s.b,l.a,l.b,s.c)&&
           opposite_side(s.b,s.c,l.a,l.b,s.a)&&opposite_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return opposite_side(l1,l2,s1,s2,s3)&&opposite_side(s1,s2,l1,l2,s3)&&
           opposite_side(s2,s3,l1,l2,s1)&&opposite_side(s3,s1,l1,l2,s2);
}
//计算两直线交点,注意事先判断直线是否共面和平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
```

```
point3 intersection(line3 u,line3 v){
    point3 ret=u.a;
    double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
             /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x+=(u.b.x-u.a.x)*t;
    ret.y+=(u.b.y-u.a.y)*t;
    ret.z+=(u.b.z-u.a.z)*t;
    return ret;
}
point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2){
    point3 ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
             /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
    ret.z+=(u2.z-u1.z)*t;
    return ret;
}
//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!
//线段和空间三角形交点请另外判断
point3 intersection(line3 l,plane3 s){
    point3 ret=pvec(s);
    double t=(ret.x*(s.a.x-l.a.x)+ret.y*(s.a.y-l.a.y)+ret.z*(s.a.z-l.a.z))/
             (ret.x*(l.b.x-l.a.x)+ret.y*(l.b.y-l.a.y)+ret.z*(l.b.z-l.a.z));
    ret.x=l.a.x+(l.b.x-l.a.x)*t;
    ret.y=l.a.y+(l.b.y-l.a.y)*t;
    ret.z=l.a.z+(l.b.z-l.a.z)*t;
    return ret;
}
point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    point3 ret=pvec(s1,s2,s3);
    double t=(ret.x*(s1.x-l1.x)+ret.y*(s1.y-l1.y)+ret.z*(s1.z-l1.z))/
             (ret.x*(l2.x-l1.x)+ret.y*(l2.y-l1.y)+ret.z*(l2.z-l1.z));
    ret.x=l1.x+(l2.x-l1.x)*t;
    ret.y=l1.y+(l2.y-l1.y)*t;
    ret.z=l1.z+(l2.z-l1.z)*t;
    return ret;
}
//计算两平面交线,注意事先判断是否平行,并保证三点不共线!
line3 intersection(plane3 u,plane3 v){
    line3 ret;
    ret.a=parallel(v.a,v.b,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersection(v.a,v.b,u.a,u.b,u.
            c);
    ret.b=parallel(v.c,v.a,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersection(v.c,v.a,u.a,u.b,u.
            c);
    return ret;
}
line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    line3 ret;
    ret.a=parallel(v1,v2,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v1,v2,u1,u2,u3);
    ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v3,v1,u1,u2,u3);
    return ret;
}
//点到直线距离
double ptoline(point3 p,line3 l){
    return vlen(Cross(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
}
double ptoline(point3 p,point3 l1,point3 l2){
    return vlen(Cross(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
}
//点到平面距离
double ptoplane(point3 p,plane3 s){
    return fabs(Dot(pvec(s),subt(p,s.a)))/vlen(pvec(s));
}
double ptoplane(point3 p,point3 s1,point3 s2,point3 s3){
    return fabs(Dot(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
}
//直线到直线距离
double linetoline(line3 u,line3 v){
    point3 n=Cross(subt(u.a,u.b),subt(v.a,v.b));
    return fabs(Dot(subt(u.a,v.a),n))/vlen(n);
}
double linetoline(point3 u1,point3 u2,point3 v1,point3 v2){
    point3 n=Cross(subt(u1,u2),subt(v1,v2));
    return fabs(Dot(subt(u1,v1),n))/vlen(n);
}
//两直线夹角 cos 值
double angle_cos(line3 u,line3 v){
    return Dot(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.b));
}
double angle_cos(point3 u1,point3 u2,point3 v1,point3 v2){
    return Dot(subt(u1,u2),subt(v1,v2))/vlen(subt(u1,u2))/vlen(subt(v1,v2));
}
//两平面夹角 cos 值
double angle_cos(plane3 u,plane3 v){
    return Dot(pvec(u),pvec(v))/vlen(pvec(u))/vlen(pvec(v));
}
double angle_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return Dot(pvec(u1,u2,u3),pvec(v1,v2,v3))/vlen(pvec(u1,u2,u3))/vlen(pvec(v1,v2,v3));
}
//直线平面夹角 sin 值
double angle_sin(line3 l,plane3 s){
    return Dot(subt(l.a,l.b),pvec(s))/vlen(subt(l.a,l.b))/vlen(pvec(s));
}
double angle_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return Dot(subt(l1,l2),pvec(s1,s2,s3))/vlen(subt(l1,l2))/vlen(pvec(s1,s2,s3));
}

// 球体相交
double vol_ints(double x1, double y1, double z1, double r1, double x2, double y2, double z2, double r2) {
    double sum = 4.00 / 3.00 * PI * r1 * r1 * r1 + 4.00 / 3.00 * PI * r2 * r2 * r2;
    double ans = 0;
    double dis = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2) + (z1 - z2) * (z1 - z2)); //球心距离
    if (dis >= r1 + r2) //没有交到的地方
    {
        ans = 0;
    } else if (dis + r1 <= r2)//重合
    {
        ans = (4.00 / 3.00) * PI * r1 * r1 * r1;
    } else if (dis + r2 <= r1) {
        ans = (4.00 / 3.00) * PI * r2 * r2 * r2;
    } else  //相交
    {
        double cal = (r1 * r1 + dis * dis - r2 * r2) / (2.00 * dis * r1);
        double h = r1 * (1 - cal);
        ans += (1.00 / 3.00) * PI * (3.00 * r1 - h) * h * h;
        cal = (r2 * r2 + dis * dis - r1 * r1) / (2.00 * dis * r2);
        h = r2 * (1.00 - cal);
        ans += (1.00 / 3.00) * PI * (3.00 * r2 - h) * h * h;
    }
    return ans;
}
```

6.72 Poly-Z

```
constexpr int P = 998244353;
using i64 = long long;
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
T power(T a, int b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
};

std::vector<int> rev;
std::vector<Z> roots{0, 1};
void dft(std::vector<Z> &a) {
    int n = a.size();

    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            Z e = power(Z(3), (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = roots[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                Z u = a[i + j];
                Z v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}
void idft(std::vector<Z> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    Z inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}
struct Poly {
    std::vector<Z> a;
    Poly() {}
    Poly(const std::vector<Z> &a) : a(a) {}
    int size() const {
        return a.size();
    }
    void resize(int n) {
        a.resize(n);
    }
    Z operator[](int idx) const {
        if (idx < 0 || idx >= size()) {
            return 0;
        }
        return a[idx];
    }
    Z &operator[](int idx) {
        return a[idx];
    }
    Poly mulxk(int k) const {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return Poly(b);
    }
    Poly modxk(int k) const {
        k = std::min(k, size());
        return Poly(std::vector<Z>(a.begin(), a.begin() + k));
    }
    Poly divxk(int k) const {
        if (size() <= k) {
            return Poly();
        }
        return Poly(std::vector<Z>(a.begin() + k, a.end()));
    }
    friend Poly operator+(const Poly &a, const Poly &b) {
        std::vector<Z> res(std::max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = a[i] + b[i];
        }
        return Poly(res);
    }
    friend Poly operator-(const Poly &a, const Poly &b) {
        std::vector<Z> res(std::max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = a[i] - b[i];
        }
        return Poly(res);
    }
    friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        int sz = 1, tot = a.size() + b.size() - 1;
        while (sz < tot)
            sz *= 2;
        a.a.resize(sz);
        b.a.resize(sz);
        dft(a.a);
        dft(b.a);
        for (int i = 0; i < sz; ++i) {
            a.a[i] = a[i] * b[i];
        }
        idft(a.a);
        a.resize(tot);
        return a;
    }
    friend Poly operator*(Z a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    friend Poly operator*(Poly a, Z b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    Poly deriv() const {
        if (a.empty()) {
            return Poly();
        }
        std::vector<Z> res(size() - 1);
        for (int i = 0; i < size() - 1; ++i) {
            res[i] = (i + 1) * a[i + 1];
        }
        return Poly(res);
    }
    Poly integr() const {
        std::vector<Z> res(size() + 1);
        for (int i = 0; i < size(); ++i) {
            res[i + 1] = a[i] / (i + 1);
        }
        return Poly(res);
    }
    Poly inv(int m) const {
        Poly x({a[0].inv()});
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly({2}) - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m) const {
        return (deriv() * inv(m)).integr().modxk(m);
    }
    Poly exp(int m) const {
        Poly x({1});
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly({1}) - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    Poly sqrt(int m) const {
        Poly x({1});
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
        }
        return x.modxk(m);
    }
    Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        std::reverse(b.a.begin(), b.a.end());
        return ((*this) * b).divxk(n - 1);
    }
    std::vector<Z> eval(std::vector<Z> x) const {
        if (size() == 0) {
            return std::vector<Z>(x.size(), 0);
        }
        const int n = std::max(int(x.size()), size());
        std::vector<Poly> q(4 * n);
        std::vector<Z> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = Poly({1, -x[l]});
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};
```

6.73 多项式快速幂

```
#include "bits/stdc++.h"
using namespace std;

const int N = 2e6 + 10;

const int P = 998244353, g3 = (P + 1) / 3;

int pow(int a,int b) {
    int r = 1;
    while(b) {
        if(b & 1) r = (ll)r * a % P;
        a = (ll)a * a % P;
        b >>= 1;
    }
    return r;
}
namespace poly {
    int rev[N];
    void NTT(int *A, int n, int inv) {
        for(int i = 0; i < n; ++i)
            if(i < rev[i]) swap(A[i], A[rev[i]]);
        for(int mid = 1; mid < n; mid <<= 1) {
            int tmp = pow(inv == 1 ? 3 : g3, (P - 1) / (mid << 1));
            for(int j = 0; j < n; j += (mid << 1)) {
                int omega = 1;
                for(int k = 0; k < mid; ++k, omega = (ll)omega * tmp % P) {
                    int x = A[j + k], y = (ll)omega * A[j + k + mid] % P;
                    A[j + k] = (x + y) % P;
                    A[j + k + mid] = (ll)(x - y + P) % P;
                }
            }
        }
        if(inv == 1) return;
        int invn = pow(n, P - 2);
        for(int i = 0; i < n; ++i)
            A[i] = (ll)A[i] * invn % P;
    }
    void Inv(int *a, int *b, int n) {
        static int B[N], A[N];
        b[0] = pow(a[0], P - 2);
        int len, lim;
        for(len = 1; len < (n << 1); len <<= 1) {
            lim = len << 1;
            for(int i = 0; i < len; i++)
                A[i] = a[i], B[i] = b[i];
            for(int i = 0; i < lim; i++)
                rev[i] = (rev[i >> 1] >> 1) | ((i & 1) ? len : 0);
            NTT(A, lim, 1), NTT(B, lim, 1);
            for(int i = 0; i < lim; i++)
                b[i] = ((2LL - 1LL * A[i] * B[i] % P) * B[i] % P + P) % P;
            NTT(b, lim, -1);
            for(int i = len; i < lim; i++)
                b[i] = 0;
        }
        for(int i = 0; i < len; i++)
            A[i] = B[i] = 0;
        for(int i = n; i < len; i++)
            b[i] = 0;
    }
    void derivative(int *a, int *b, int n) {
        b[n - 1] = 0;
        for(int i = 1; i < n; ++i)
            b[i - 1] = (ll)a[i] * i % P;
    }
    void inter(int *a, int *b, int n) {
        *b = 0;
        for(int i = n - 1; i >= 0; --i)
            b[i + 1] = a[i] * (ll)pow(i + 1, P - 2) % P;
    }
    void ln(int *a, int *b, int n) {
        static int F[N];
        derivative(a, F, n);
        Inv(a, b, n);
        int lim = 1;
        while(lim < (n << 1)) lim <<= 1;
        for(int i = 1; i < lim; i++)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) ? (lim >> 1) : 0);
        for(int i = n; i < lim; ++i)
            b[i] = F[i] = 0;
        NTT(F, lim, 1), NTT(b, lim, 1);
        for(int i = 0; i < lim; ++i)
            F[i] = (ll)b[i] * F[i] % P;
        NTT(F, lim, 0);
        inter(F, b, n);
        for(int i = n; i < lim; ++i)
            b[i] = 0;
    }
    void exp(int*a, int*F, int n) {
        if(n == 1)
            *F = 1;
        else {
            exp(a, F, n + 1 >> 1);
            static int F0[N], A[N];
            for(int i = 0; i <= (n << 1); ++i)
                F0[i] = 0, A[i] = a[i];
            ln(F, F0, n);
            int lim = 1;
            while(lim < (n << 1)) lim <<= 1;
            for(int i = 1; i < lim; i++)
                rev[i] = (rev[i >> 1] >> 1) | ((i & 1) ? (lim >> 1) : 0);
            for(int i = n; i < lim; ++i)
                A[i] = 0;
            NTT(A, lim, 1), NTT(F0, lim, 1), NTT(F, lim, 1);
            for(int i = 0; i < lim; ++i)
                F[i] = F[i] * (A[i] + 1LL - F0[i] + P) % P;
            NTT(F, lim, 0);
            for(int i = n; i < lim; ++i)
                F[i] = 0;
        }
    }
}
using namespace poly;

int a[N], b[N];

void solve() {
  int n, m, k; cin >> n >> m >> k;
  for(int i = 0;i < n; i++) cin >> a[i];
  ln(a, b, m);
  for(int i = 0;i < m; i++) b[i] = b[i] * k % mod;
  exp(b, a, m);
  for(int i = 0;i < m; i++) cout << a[i] << " ";
}
```

6.74 Geometry

```
using Point = std::complex<double>;

#define x real
#define y imag

double dot(const Point &a, const Point &b) {
    return (std::conj(a) * b).x();
}

double cross(const Point &a, const Point &b) {
    return (std::conj(a) * b).y();
}

struct Line {
    Point a;
    Point b;
    Line(const Point &a, const Point &b) : a(a), b(b) {}
};

Point rotate(const Point &a) {
    return Point(-a.y(), a.x());
}

int sgn(const Point &a) {
    return a.y() > 0 || (a.y() == 0 && a.x() > 0) ? 1 : -1;
}

```

```
bool onLeft(const Point &a, const Line &l) {
    return cross(l.b - l.a, a - l.a) > 0;
}

Point intersection(const Line &l1, const Line &l2) {
    return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1.a - l1.b));
}

std::vector<Point> hp(std::vector<Line> lines) {
    std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
        auto d1 = l1.b - l1.a;
        auto d2 = l2.b - l2.a;

        if (sgn(d1) != sgn(d2)) {
            return sgn(d1) == 1;
        }

        return cross(d1, d2) > 0;
    });

    std::deque<Line> ls;
    std::deque<Point> ps;
    for (auto l : lines) {
        if (ls.empty()) {
            ls.push_back(l);
            continue;
        }

        while (!ps.empty() && !onLeft(ps.back(), l)) {
            ps.pop_back();
            ls.pop_back();
        }

        while (!ps.empty() && !onLeft(ps[0], l)) {
            ps.pop_front();
            ls.pop_front();
        }

        if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
            if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {

                if (!onLeft(ls.back().a, l)) {
                    assert(ls.size() == 1);
                    ls[0] = l;
                }
                continue;
            }
            return {};
        }

        ps.push_back(intersection(ls.back(), l));
        ls.push_back(l);
    }

    while (!ps.empty() && !onLeft(ps.back(), ls[0])) {
        ps.pop_back();
        ls.pop_back();
    }
    if (ls.size() <= 2) {
        return {};
    }
    ps.push_back(intersection(ls[0], ls.back()));

    return std::vector<Point>(ps.begin(), ps.end());
}
```

7 图论

7.1 graph

```
template <typename T>
class graph {
public:
    struct edge {
        int from;
        int to;
        T cost;
    };

    const int n;
    std::vector<edge> edges;
    std::vector<std::vector<int>> g;

    graph(int _n) : n(_n), g(n) {}

    virtual int add(int from, int to, T const) = 0;
};

template <typename T>
class digraph : public graph<T> {
public:
    using graph<T>::edges;
    using graph<T>::g;
    using graph<T>::n;

    digraph(int _n) : graph<T>(_n) {}

    int add(int from, int to, T cost = 1) {
        assert(0 <= from && from < n && 0 <= to && to < n);
        int id = (int) edges.size();
        g[from].push_back(id);
        edges.push_back({from, to, cost});
        return id;
    }

    digraph<T> reverse() const {
        digraph<T> rev(n);
        for (auto &e : edges) {
            rev.add(e.to, e.from, e.cost);
        }
        return rev;
    }
};


template <typename T>
class undigraph : public graph<T> {
public:
    using graph<T>::edges;
    using graph<T>::g;
    using graph<T>::n;

    undigraph(int _n) : graph<T>(_n) {}

    int add(int from, int to, T cost = 1) {
        assert(0 <= from && from < n && 0 <= to && to < n);
        int id = (int) edges.size();
        g[from].push_back(id);
        g[to].push_back(id);
        edges.push_back({from, to, cost});
        return id;
    }
};
```

7.2 isBipartiteGraph

```
template <typename T>
bool isBipartiteGraph(const graph<T>& g) {
    std::vector<int> color(g.n);
    bool flag = true;
    std::function<bool(int, int)> dfs = [&](int u, int x) -> bool {
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (!color[to]) {
                dfs(to, 3 - x);
            }
            if (color[to] == color[u]) {
                flag = false;
            }
        }
    };
    for (int i = 0; i < g.n; i++) {
        if (!color[i]) {
            dfs(i, 1);
        }
    }
    return flag;
}
```

7.3 hungry

```
template <typename T>
int hungry(const digraph<T>& g) {
    std::vector<bool> was(g.n);
    std::vector<int> match(g.n, -1);
    std::function<bool(int)> dfs = [&](int u) -> bool {
        for (int id : g.g[u]) {
            auto& e : g.edges[id];
            int to = e.to;
            if (!was[to]) {
                was[to] = true;
                if (match[to] == -1 || dfs(match[to])) {
                    match[to] = u;
                    return true;
                }
            }
        }
        return false;
    };

    int ans = 0;
    for (int i = 0; i < n; i++) {
        vis.assign(g.n, false);
        if (dfs(i)) ans++;
    }
    return ans;
}
```

7.4 KM

```

template <typename T>
class hungarian {  // km
public :
  int n;
  std::vector<int> matchx;  // 左集合对应的匹配点
  std::vector<int> matchy;  // 右集合对应的匹配点
  std::vector<int> pre;     // 连接右集合的左点
  std::vector<bool> visx;   // 拜访数组 左
  std::vector<bool> visy;   // 拜访数组 右
  std::vector<T> lx;
  std::vector<T> ly;
  std::vector<vector<T> > g;
  std::vector<T> slack;
  T inf;
  T res;
  std::queue<int> q;
  int org_n;
  int org_m;

  hungarian(int _n, int _m) {
    org_n = _n;
    org_m = _m;
    n = max(_n, _m);
    inf = numeric_limits<T>::max();
    res = 0;
    g = vector<vector<T> >(n, vector<T>(n));
    matchx = vector<int>(n, -1);
    matchy = vector<int>(n, -1);
    pre = vector<int>(n);
    visx = vector<bool>(n);
    visy = vector<bool>(n);
    lx = vector<T>(n, -inf);
    ly = vector<T>(n);
    slack = vector<T>(n);
  }

  void addEdge(int u, int v, int w) {
    g[u][v] = max(w, 0);  // 负值还不如不匹配 因此设为0不影响
  }

  bool check(int v) {
    visy[v] = true;
    if (matchy[v] != -1) {
      q.push(matchy[v]);
      visx[matchy[v]] = true;  // in S
      return false;
    }
    // 找到新的未匹配点 更新匹配点 pre 数组记录着"非匹配边"上与之相连的点
    while (v != -1) {
      matchy[v] = pre[v];
      swap(v, matchx[pre[v]]);
    }
    return true;
  }

  void bfs(int i) {
    while (!q.empty()) {
      q.pop();
    }
    q.push(i);
    visx[i] = true;
    while (true) {
      while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v = 0; v < n; v++) {
          if (!visy[v]) {
            T delta = lx[u] + ly[v] - g[u][v];
            if (slack[v] >= delta) {
              pre[v] = u;
              if (delta) {
                slack[v] = delta;
              } else if (check(v)) {  // delta=0 代表有机会加入相等子图 找增广路
                                      // 找到就return 重建交错树
                return;
              }
            }
          }
        }
      }
      // 没有增广路 修改顶标
      T a = inf;
      for (int j = 0; j < n; j++) {
        if (!visy[j]) {
          a = min(a, slack[j]);
        }
      }
      for (int j = 0; j < n; j++) {
        if (visx[j]) {  // S
          lx[j] -= a;
        }
        if (visy[j]) {  // T
          ly[j] += a;
        } else {  // T'
          slack[j] -= a;
        }
      }
      for (int j = 0; j < n; j++) {
        if (!visy[j] && slack[j] == 0 && check(j)) {
          return;
        }
      }
    }
  }

  void solve() {
    // 初始顶标
```

```
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        lx[i] = max(lx[i], g[i][j]);
      }
    }

    for (int i = 0; i < n; i++) {
      fill(slack.begin(), slack.end(), inf);
      fill(visx.begin(), visx.end(), false);
      fill(visy.begin(), visy.end(), false);
      bfs(i);
    }

    // custom
    for (int i = 0; i < n; i++) {
      if (g[i][matchx[i]] > 0) {
        res += g[i][matchx[i]];
      } else {
        matchx[i] = -1;
      }
    }
    // cout << res << "\n";
    // for (int i = 0; i < org_n; i++) {
    //   cout << matchx[i] + 1 << " ";
    // }
    // cout << "\n";
  }
};
```

7.5 galeShapley

```
#include<iostream>
using namespace std;

const int N=4;

void GaleShapley(const int (&man)[MAX][MAX], const int (&woman)[MAX][MAX], int (&match)[MAX]) {
    int wm[MAX][MAX];    // wm[i][j]: rank from girl i to boy j
    int choose[MAX];    // choose[i]: current boyfriend of girl i
    int manIndex[MAX]; //    manIndex[i]: how many girls that have rejected boy i
    int i, j;
    int w, m;
    for (i = 0; i < N; i++) {
        match[i] = -1;
        choose[i] = -1;
        manIndex[i] = 0;
        for (j = 0; j < N; j++)
            wm[i][woman[i][j]] = j;
    }

    bool bSingle = false;
    while (!bSingle) {
        bSingle = true;
        for (i = 0; i < N; i++) {
            if (match[i] != -1) // boy i already have a girlfriend
                continue;
            bSingle = false;
            j = manIndex[i]++; // the jth girl that boy i like most
            w = man[i][j];
            m = choose[w];    // current girl w's boyfriend
            if (m == -1 || wm[w][i] < wm[w][m]) { // if girl w prefer boy i
                match[i] = w;
                choose[w] = i;
                if (m != -1)
                    match[m] = -1;
            }
        }
    }
}


void Print(const int(&match)[MAX], int N) {
    for (int i = 0; i < N; i++)
        cout << i << " " << match[i] << endl;
}


int main(){
    int man[N][N]={
        {2,3,1,0},
        {2,1,3,0},
        {0,2,3,1},
        {1,3,2,0},
    };
    int woman[N][N]={
        {0,3,2,1},
        {0,1,2,3},
        {0,2,3,1},
        {1,0,3,2},
    };

    int match[N];
    GaleShapley(man,woman,match);
    Print(match,N);

    return 0;
}
```

7.6 bellman-ford

```
template <typename T>
std::vector<std::vector<T>> bellman_ford(const graph<T>& g, int st) {
    std::vector<T> dist(g.n, std::numeric_limits<T>::max());
    dist[st] = 0;

    // Relax all edges |V| - 1 times. A simple
    // shortest path from src to any other vertex can have
    // at-most |V| - 1 edges
    for (int i = 1; i < g.n; i++) {
        for (auto& e : g.edges) {
            int from = e.from, to = e.to, cost = e.cost;
            if (dist[from] != std::numeric_limits<T>::max() && dist[from] + cost < dist[to]) {
                dist[to] = dist[from] + e.from;
            }
        }
    }

    for (auto& e : g.edges) {
        int from = e.from, to = e.to, cost = e.cost;
        if (dist[from] != std::numeric_limits<T>::max() && dist[from] + cost < dist[to]) {
            // Graph contains negative weight cycle
            return {};
        }
    }

    return dist;
}
```

7.7 dijkatra

```
template <typename T>
std::vector<T> dijkstra(const graph<T>& g, int st) {
    assert(0 <= st && st < g.n);
    std::vector<T> dist(g.n, std::numeric_limits<T>::max());
    std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T, int>> > q;
    dist[st] = 0;
    q.emplace(dist[st], st);
    while (!q.empty()) {
        T expected = q.top().first;
        int u = q.top().second; q.pop();
        if (dist[u] != expected) {
            continue ;
        }
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (dist[to] > dist[u] + e.cost) {
                dist[to] = dist[u] + e.cost;
                q.emplace(dist[to], to);
            }
        }
    }
    return dist;
    // returns numeric_limits<T>::max() if there's no path
}
```

7.8 floyd

```
template <typename T>
void floyd(std::vector<std::vector<T>>& dist) {
    for (int k = 0; k < dist.size(); k++) {
        for (int i = 0; i < dist.size(); i++) {
            for (int j = 0; j < dist.size(); j++) {
                dist[i][j] = std::min(dist[i][j], dist[i][k], dist[k][j]);
            }
        }
    }
}
```

7.9 spfa

```cpp
template <typename T>
std::vector<T> spfa(const graph<T>& g, int st) {
    std::vector<T> dist(g.n, std::numeric_limits<T>::max());
    std::vector<bool> vis(g.n);
    std::vector<int> cnt(g.n);
    std::vector<int> x(1, st);
    dist[st] = 0; vis[st] = true;
    for (int ptr = 0; ptr < x.size(); ptr++) {
        int u = x[ptr];
        vis[u] = false;
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (dist[to] > dist[u] + e.cost) {
                dist[to] = dist[u] + e.cost;
                if (!vis[to]) {
                    cnt[to]++;
                    vis[to] = true;
                    if (cnt[to] >= g.n) {
                        return std::vector<T>();
                    }
                    x.push_back(to);
                }
            }
        }
    }
    return dist;
}
```

7.10 Kruskal

```cpp
template <typename T>
std::vector<int> find_mst(const undigraph<T> &g, T& ans) {
    std::vector<int> order(g.edges.size());
    iota(order.begin(), order.end(), 0);
    sort(order.begin(), order.end(), [&g](int a, int b) {
        return g.edges[a].cost < g.edges[b].cost;
    });
    DSU d(g.n);
    std::vector<int> ans_list;
    ans = 0;
    for (int id : order) {
        auto &e = g.edges[id];
        if (!d.same(e.from, e.to)) {
            d.merge(e.from, e.to);
            ans_list.push_back(id);
            ans += e.cost;
        }
    }
    return ans_list;
    // returns edge ids of minimum "spanning" forest
}
```

7.11 prim

```cpp
template <typename T>
bool find_mst(const undigraph<T> &g, T& ans) {
    std::vector<bool> vis(g.n);
    std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater
    <std::pair<T, int>> > q;
    q.push({0, 0});
    int cnt = 0; ans = 0;
    while (!q.empty() && cnt < g.n) {
        T expected = q.top().first;
        int u = q.top().second; q.pop();
        if (vis[u]) continue;
        vis[u] = true;
        ans += expected; cnt++;
        for (int id : g.g[u]) {
            auto &e = g.edges[id];
            int to = e.from ^ e.to ^ u;
            if (!vis[to]) {
                q.push({e.cost, to});
            }
        }
    }
    return cnt == g.n;
    // returns false if there's not connected
}
```

7.12 topSort

```cpp
template <typename T>
std::vector<int> find_topsort(const digraph<T> &g) {
    std::vector<int> deg(g.n, 0);
    for (int id = 0; id < (int) g.edges.size(); id++) {
        deg[g.edges[id].to]++;
    }
    std::vector<int> x;
    for (int i = 0; i < g.n; i++) {
        if (deg[i] == 0) {
            x.push_back(i);
        }
    }

    for (int ptr = 0; ptr < (int) x.size(); ptr++) {
        int i = x[ptr];
        for (int id : g.g[i]) {
            auto &e = g.edges[id];
            int to = e.to;
            if (--deg[to] == 0) {
                x.push_back(to);
            }
        }
    }

    if ((int) x.size() != g.n) {
        return std::vector<int>();
    }
    return x;
}
```

7.13 scc

```
template <typename T>
std::vector<int> find_scc(const digraph<T> &g, int &cnt) {
    digraph<T> g_rev = g.reverse();
    std::vector<int> order;
    std::vector<bool> was(g.n, false);
    std::function<void(int)> dfs1 = [&](int v) {
        was[v] = true;
        for (int id : g.g[v]) {
            auto &e = g.edges[id];
            int to = e.to;
            if (!was[to]) {
                dfs1(to);
            }
        }
        order.push_back(v);
    };
    for (int i = 0; i < g.n; i++) {
        if (!was[i]) {
            dfs1(i);
        }
    }
    std::vector<int> c(g.n, -1);
    std::function<void(int)> dfs2 = [&](int v) {
        for (int id : g_rev.g[v]) {
            auto &e = g_rev.edges[id];
            int to = e.to;
            if (c[to] == -1) {
                c[to] = c[v];
                dfs2(to);
            }
        }
    };
    cnt = 0;
    for (int id = g.n - 1; id >= 0; id--) {
        int i = order[id];
        if (c[i] != -1) {
            continue;
        }
        c[i] = cnt++;
        dfs2(i);
    }
    return c;
}
```

7.14 cycles

```
template <typename T>
std::vector<std::vector<int>> find_cycles(const graph<T> &g, int bound_cnt = 1 << 30,
    int bound_size = 1 << 30) {
    std::vector<int> was(g.n, -1);
    std::vector<int> st;
    std::vector<std::vector<int>> cycles;
    int total_size = 0;
    std::function<void(int, int)> dfs = [&](int v, int pe) {
        if ((int) cycles.size() >= bound_cnt || total_size >= bound_size) {
            return ;
        }
        was[v] = (int) st.size();
        for (int id : g.g[v]) {
            if (id == pe) {
                continue ;
            }
            auto &e = g.edges[id];
            int to = e.from ^ e.to ^ v;
            if (was[to] >= 0) {
                std::vector<int> cycle(1, id);
                for (int j = was[to]; j < (int) st.size(); j++) {
                    cycle.push_back(st[j]);
                }
                cycles.push_back(cycle);
                total_size += (int) cycle.size();
                if ((int) cycles.size() >= bound_cnt || total_size >= bound_size) {
                    return;
                }
                continue;
            }
            if (was[to] == -1) {
                st.push_back(id);
                dfs(to, id);
                st.pop_back();
            }
        }
        was[v] = -2;
    };
    for (int i = 0; i < g.n; i++) {
        if (was[i] == -1) {
            dfs(i, -1);
        }
    }
    return cycles;
    // cycles are given by edge ids, all cycles are simple
    // breaks after getting bound_cnt cycles or total_size >= bound_size
    // digraph: finds at least one cycle in every connected component (if not broken)
    // undigraph: finds cycle basis
}

template <typename T>
std::vector<int> edges_to_vertices(const graph<T> &g, const std::vector<int> &edge_cycle) {
    int sz = (int) edge_cycle.size();
    std::vector<int> vertex_cycle;
    if (sz <= 2) {
        vertex_cycle.push_back(g.edges[edge_cycle[0]].from);
        if (sz == 2) {
            vertex_cycle.push_back(g.edges[edge_cycle[0]].to);
        }
    } else {
        for (int i = 0; i < sz; i++) {
            int j = (i + 1) % sz;
            auto &e = g.edges[edge_cycle[i]];
            auto &other = g.edges[edge_cycle[j]];
            if (other.from == e.from || other.to == e.from) {
                vertex_cycle.push_back(e.to);
            } else {
                vertex_cycle.push_back(e.from);
            }
        }
    }
    return vertex_cycle;
    // only for simple cycles!
}
```

7.15 ternaryRingCount

```
template <typename T>
int ternaryRingCount(const digraph<T>& g) {
    std::vector<int> d(g.n);
    for (auto& e : g.edges) {
        int from = e.from, to = e.to;
        d[from]++;
        d[to]++;
    }
    digraph<T>& ng(g.n);
    for (auto& e : g.edges) {
        int from = e.from, to = e.to;
        if (d[from] < d[to] || (d[from] == d[to] && from > to)) std::swap(from, to);
        ng.add(from, to);
    }
    int ans = 0;
    std::vector<bool> was(ng.n);
    for (int u = 0; u < ng.n; u++) {
        for (int id : ng.g[u]) {
            auto& e = ng.edges[id];
            int to = e.to;
            was[to] = u;
        }
        for (int id1 : ng.g[u]) {
            auto& e1 = ng.edges[id1];
            int to1 = e1.to;
            for (int id2 : ng.g[to1]) {
                auto& e2 = ng.g[id2];
                int to2 = e2.to;
                if (was[to2] == u) ans++;
            }
        }
    }
    return ans;
}
```

7.16 eulerian-path

```
template <typename T>
std::vector<int> find_eulerian_path(const graph<T> &g, int &root) {
    // in_deg and out_deg are fake for undigraph!
    std::vector<int> in_deg(g.n, 0);
    std::vector<int> out_deg(g.n, 0);
    int cnt_edges = 0;
    for (int id = 0; id < (int) g.edges.size(); id++) {
        cnt_edges++;
        auto &e = g.edges[id];
        out_deg[e.from]++;
        in_deg[e.to]++;
    }
    root = -1;
    int odd = 0;
    for (int i = 0; i < g.n; i++) {
        if ((in_deg[i] + out_deg[i]) % 2 == 1) {
            odd++;
            if (root == -1 || out_deg[i] - in_deg[i] > out_deg[root] - in_deg[root]) {
                root = i;
            }
        }
    }
    if (odd > 2) {
        root = -1;
        return std::vector<int>();
    }
    if (root == -1) {
        root = 0;
        while (root < g.n && in_deg[root] + out_deg[root] == 0) {
            root++;
        }
        if (root == g.n) {
            // an empty path
            root = 0;
            return std::vector<int>();
        }
    }
    std::vector<bool> used(g.edges.size(), false);
    std::vector<int> ptr(g.n, 0);
    std::vector<int> balance(g.n, 0);
    std::vector<int> res(cnt_edges);
    int stack_ptr = 0;
    int write_ptr = cnt_edges;
    int v = root;
    while (true) {
        bool found = false;
        while (ptr[v] < (int) g.g[v].size()) {
            int id = g.g[v][ptr[v]++];
            if (used[id]) {
            continue;
            }
            used[id] = true;
            res[stack_ptr++] = id;
            auto &e = g.edges[id];
            balance[v]++;
            v ^= e.from ^ e.to;
            balance[v]--;
            found = true;
            break;
        }
        if (!found) {
            if (stack_ptr == 0) {
                break;
            }
            int id = res[--stack_ptr];
            res[--write_ptr] = id;
            auto &e = g.edges[id];
            v ^= e.from ^ e.to;
        }
    }
    int disbalance = 0;
    for (int i = 0; i < g.n; i++) {
        disbalance += abs(balance[i]);
    }
    if (write_ptr != 0 || disbalance > 2) {
        root = -1;
        return std::vector<int>();
    }
    return res;
}
```

7.17 twoSat

```
class twosat {
public:
    digraph<int> g;
    int n;

    twosat(int _n) : g(digraph<int>(2 * _n)), n(_n) {}

    inline void add(int x, int value_x) {
        assert(0 <= x && x < n);
        assert(0 <= value_x && value_x <= 1);
        g.add(2 * x + (value_x ^ 1), 2 * x + value_x);
    }

    inline void add(int x, int value_x, int y, int value_y) {
        assert(0 <= x && x < n && 0 <= y && y < n);
        assert(0 <= value_x && value_x <= 1 && 0 <= value_y && value_y <= 1);
        g.add(2 * x + (value_x ^ 1), 2 * y + value_y);
        g.add(2 * y + (value_y ^ 1), 2 * x + value_x);
    }

    inline std::vector<int> solve() {
        int cnt;
        std::vector<int> c = find_scc(g, cnt);
        std::vector<int> res(n);
        for (int i = 0; i < n; i++) {
            if (c[2 * i] == c[2 * i + 1]) {
                return std::vector<int>();
            }
            res[i] = (c[2 * i] < c[2 * i + 1]);
        }
        return res;
    }
};
```

7.18 maxAssignment

```
template<class T>
struct MaxAssignment {
    public:
        T solve(int nx, int ny, std::vector<std::vector<T>> a) {
            assert(0 <= nx && nx <= ny);
            assert(int(a.size()) == nx);
            for (int i = 0; i < nx; ++i) {
                assert(int(a[i].size()) == ny);
                for (auto x : a[i])
                    assert(x >= 0);
            }

            auto update = [&](int x) {
                for (int y = 0; y < ny; ++y) {
                if (lx[x] + ly[y] - a[x][y] < slack[y]) {
                    slack[y] = lx[x] + ly[y] - a[x][y];
                        slackx[y] = x;
                    }
                }
            };

            costs.resize(nx + 1);
            costs[0] = 0;
            lx.assign(nx, std::numeric_limits<T>::max());
            ly.assign(ny, 0);
            xy.assign(nx, -1);
            yx.assign(ny, -1);
            slackx.resize(ny);
            for (int cur = 0; cur < nx; ++cur) {
                std::queue<int> que;
                visx.assign(nx, false);
                visy.assign(ny, false);
                slack.assign(ny, std::numeric_limits<T>::max());
                p.assign(nx, -1);

                for (int x = 0; x < nx; ++x) {
                    if (xy[x] == -1) {
                        que.push(x);
                        visx[x] = true;
                        update(x);
                    }
                }

                int ex, ey;
                bool found = false;
                while (!found) {
                    while (!que.empty() && !found) {
                        auto x = que.front();
                        que.pop();
                        for (int y = 0; y < ny; ++y) {
                            if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
                                if (yx[y] == -1) {
                                    ex = x;
                                    ey = y;
                                    found = true;
                                    break;
                                }
                                que.push(yx[y]);
                                p[yx[y]] = x;
                                visy[y] = visx[yx[y]] = true;
                                update(yx[y]);
                            }
                        }
                    }
                    if (found)
                        break;

                    T delta = std::numeric_limits<T>::max();
                    for (int y = 0; y < ny; ++y)
                        if (!visy[y])
                            delta = std::min(delta, slack[y]);
                    for (int x = 0; x < nx; ++x)
                        if (visx[x])
                            lx[x] -= delta;
                    for (int y = 0; y < ny; ++y) {
                        if (visy[y]) {
                            ly[y] += delta;
                        } else {
                            slack[y] -= delta;
                        }
                    }
                    for (int y = 0; y < ny; ++y) {
                        if (!visy[y] && slack[y] == 0) {
                            if (yx[y] == -1) {
                                ex = slackx[y];
                                ey = y;
                                found = true;
                                break;
                            }
                            que.push(yx[y]);
                            p[yx[y]] = slackx[y];
                            visy[y] = visx[yx[y]] = true;
                            update(yx[y]);
                        }
                    }
                }

                costs[cur + 1] = costs[cur];
                for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
                    costs[cur + 1] += a[x][y];
                    if (xy[x] != -1)
                        costs[cur + 1] -= a[x][xy[x]];
                    ty = xy[x];
                    xy[x] = y;
                    yx[y] = x;
                }
            }
            return costs[nx];
        }
        std::vector<int> assignment() {
            return xy;
        }
        std::pair<std::vector<T>, std::vector<T>> labels() {
            return std::make_pair(lx, ly);
        }
        std::vector<T> weights() {
            return costs;
        }
    private:
        std::vector<T> lx, ly, slack, costs;
        std::vector<int> xy, yx, p, slackx;
        std::vector<bool> visx, visy;
};
```

7.19 tarjan

```
template <typename T>
class tarjan {
public:
    digraph<T> &g;
    int tim;
    std::vector<int> dfn;
    std::vector<int> low;
    std::vector<bool> vis;
    std::stack<int> s;
    std::set<int> cutVertexs;
    std::vector<std::array<int, 2>> cutBridges;
    std::vector<std::vector<int>> cycles;
    int root;
    tarjan(digraph<T> &_g) : g(_g) {
        dfn.resize(g.n);
        low.resize(g.n);
        vis.resize(g.n);
        tim = 1;
    }

    void dfs(int u) {
        s.push(u);
        dfn[u] = low[u] = tim++;
        vis[u] = true;
        int child = 0;
        for (int id : g.g[u]) {
            auto &e = g.edges[id];
            int to = e.to;
            if (!dfn[to]) {
                dfs(to);
                low[u] = std::min(low[u], low[to]);
                // if u is not root and low value of one of its child is more than dfn value of u
                // if u is root of DFS tree and has two or more children
                if (low[to] >= dfn[u]) {
                    child++;
                    if (u != root || child >= 2) {
                        cutVertexs.insert(u);
                    }
                }
                // if the lowest vertex reachable from subtree under v is below u is DFS tree, the u-b is a bridge
                if (low[to] > dfn[u]) {
                    cutBridges.push_back({std::min(u, to), std::max(u, to)});
                }
            } else if (vis[to]) {
                low[u] = std::min(low[u], dfn[to]);
            }
        }

        // find a strongly connected component
        if (dfn[u] == low[u]) {
            int from, cnt = 0;
            std::vector<int> cycle;
            do {
                from = s.top();
                cycle.push_back(from);
                vis[from] = false;
                s.pop();
                cnt++;
                // TODO ... num of connected component = cnt
            } while (from != u);
            cycles.push_back(cycle);
        }
    }

    void solve() {
        std::vector<std::vector<int>> cycles;
        for (int i = 0; i < g.n; i++) {
            if (!dfn[i]) {
                root = i;
                dfs(i);
            }
        }
    }
};
```

7.20 biconnected-components

```
template <typename T>
std::vector<int> find_bicone(dfs_undigraph<T> &g, int &cnt) {
    g.dfs_all();
    std::vector<int> vertex_comp(g.n);
    cnt = 0;
    for (int i : g.order) {
        if (g.pv[i] == -1 || g.min_depth[i] == g.depth[i]) {
            vertex_comp[i] = cnt++;
        } else {
            vertex_comp[i] = vertex_comp[g.pv[i]];
        }
    }
    return vertex_comp;
}
```

7.21 differenceConstraints

```

/*
差分约束是解决这样一类问题
给出n个形如x[j] - x[i] <= k的式子，求x[n] - x[1]的最大/最小值
最大值—>把所有式子整理为x[j] - x[i] <= k，从i向j连一条边权为k的边，跑最短路
最小值—>把所有式子整理为x[j] - x[i] >= k，从i向j连一条边权为k的边，跑最长路
注意初始化 有时候需要超级源点0
*/

//dfs跑差分约束最短路
template <typename T>
bool differenceConstraints(const graph<T>& g) {
    std::vector<bool> was(g.n);
    std::vector<T> dist(g.n, std::numeric_limits<T>::max());
    std::function<bool(int)> spfa = [&](int u) -> bool {
        was[u] = true;
        for (int id : g.g[u]) {
            auto& e = g.edges[id];
            int to = e.from ^ e.to ^ to;
            if (dist[u] + e.cost < dist[to]) {
                if (was[to]) return false;
                dist[to] = dist[u] + e.cost;
                if (!spfa(to)) return false;
            }
        }
        was[u] = false;
        return true;
    };
    return spfa(0);
}
```

7.22 AHU

```
//用来判断两棵树是否同构
// AHU :判断两棵树是否是同构
//同构:在更换节点的标号之后两棵树能完全相同

const int N = 1e5 + 5;
const int maxn = N << 1;

int n;
struct Edge {
    int v, nxt;
} e[maxn << 1];
int head[maxn], sz[maxn], f[maxn], maxv[maxn], tag[maxn], tot, Max;
vector<int> center[2], L[maxn], subtree_tags[maxn];
void addedge(int u, int v) {
    e[tot].v = v;
    e[tot].nxt = head[u];
    head[u] = tot++;
    e[tot].v = u;
    e[tot].nxt = head[v];
    head[v] = tot++;
}

void dfs_size(int u, int fa) {
    sz[u] = 1;
    maxv[u] = 0;
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if (v == fa) continue;
        dfs_size(v, u);
        sz[u] += sz[v];
        maxv[u] = max(maxv[u], sz[v]);
    }
}

void dfs_center(int rt, int u, int fa, int id) {
    maxv[u] = max(maxv[u], sz[rt] - sz[u]);
    if (Max > maxv[u]) {
        center[id].clear();
        Max = maxv[u];
    }
    if (Max == maxv[u]) center[id].push_back(u);
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if (v == fa) continue;
        dfs_center(rt, v, u, id);
    }
}

int dfs_height(int u, int fa, int depth) {
    L[depth].push_back(u);
    f[u] = fa;
    int h = 0;
    for (int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if (v == fa) continue;
        h = max(h, dfs_height(v, u, depth + 1));
    }
    return h + 1;
}

void init(int n) {
    for (int i = 1; i <= 2 * n; i++) head[i] = 0;
    tot = 1;
    center[0].clear();
    center[1].clear();

    int u, v;
    for (int i = 1; i <= n - 1; i++) {//在这里输入第一棵树的边
        scanf("%d %d", &u, &v);
        addedge(u, v);
    }
    dfs_size(1, -1);
    Max = n;
    dfs_center(1, 1, -1, 0);

    for (int i = 1; i <= n - 1; i++) {//在这里输入第二棵树的边
        scanf("%d %d", &u, &v);
        addedge(u + n, v + n);
    }
    dfs_size(1 + n, -1);
    Max = n;
    dfs_center(1 + n, 1 + n, -1, 1);
}

bool cmp(int u, int v) { return subtree_tags[u] < subtree_tags[v]; }

bool rootedTreeIsomorphism(int rt1, int rt2) {
    for (int i = 0; i <= 2 * n + 1; i++) L[i].clear(), subtree_tags[i].clear();
    int h1 = dfs_height(rt1, -1, 0);
    int h2 = dfs_height(rt2, -1, 0);
    if (h1 != h2) return false;
    int h = h1 - 1;
    for (int j = 0; j < (int)L[h].size(); j++) tag[L[h][j]] = 0;
    for (int i = h - 1; i >= 0; i--) {
        for (int j = 0; j < (int)L[i + 1].size(); j++) {
            int v = L[i + 1][j];
            subtree_tags[f[v]].push_back(tag[v]);
        }

        sort(L[i].begin(), L[i].end(), cmp);

        for (int j = 0, cnt = 0; j < (int)L[i].size(); j++) {
            if (j && subtree_tags[L[i][j]] != subtree_tags[L[i][j - 1]]) ++cnt;
            tag[L[i][j]] = cnt;
        }
    }
    return subtree_tags[rt1] == subtree_tags[rt2];
}

bool treeIsomorphism() {
    if (center[0].size() == center[1].size()) {
        if (rootedTreeIsomorphism(center[0][0], center[1][0])) return true;
        if (center[0].size() > 1)
            return rootedTreeIsomorphism(center[0][0], center[1][1]);
    }
    return false;
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        init(n);
        puts(treeIsomorphism() ? "YES" : "NO");
    }
    return 0;
}
```

7.23 Astar

```
#include "bits/stdc++.h"
using namespace std;
//A*
//用来计算点A到点B的第k短的路径

const int MAXN = 55;
const int MAXM = MAXN * MAXN;

int dis[MAXN];
int n, m, k, a, b, cnt;
bool hav = false;

namespace G1{//反图
    int to[MAXM], val[MAXM], head[MAXN], nxt[MAXM], cnt;
    bool vis[MAXN];

    void AddEdge(int u, int v, int w) {
        cnt++;
        to[cnt] = v;
        val[cnt] = w;
        nxt[cnt] = head[u];
        head[u] = cnt;
    }

    void Spfa(int s, int t) {//SPFA+SLF跑最短路
        memset(dis, 0x7f, sizeof(dis)); dis[s] = 0;
        deque<int> q; q.push_back(s); vis[s] = true;
        while (!q.empty()) {
            int u = q.front(); q.pop_front(); vis[u] = false;
            for (int i = head[u]; i; i = nxt[i]) {
                int v = to[i];
                if (dis[v] > dis[u] + val[i]) {
                    dis[v] = dis[u] + val[i];
                    if (!vis[v]) {
                        vis[v] = true;
                        if (!q.empty() && dis[v] < dis[q.front()]) {
                            q.push_front(v);
                        } else {
                            q.push_back(v);
                        }
                    }
                }
            }
        }
    }
}

namespace G2{//原图
    int to[MAXM], val[MAXM], nxt[MAXM], head[MAXN], cnt;

    void AddEdge(int u, int v, int w) {
        cnt++;
        to[cnt] = v;
        val[cnt] = w;
        nxt[cnt] = head[u];
        head[u] = cnt;
    }

    struct Data{//当前位置，走过的距离，s->now->t总距离，走的步骤
        int now, pas, val;
        vector<int> route;
        /*
        bool operator < (const Data &b) const {return val > b.val;}
        */
        bool operator < (const Data &b) const {//重载
            if (val != b.val) return val > b.val;
            int sz = min(route.size(), b.route.size());
            for (int i = 0; i < sz; i++) {
                if (route[i] != b.route[i]) return route[i] > b.route[i];
            }
            return route.size() > b.route.size();
        }
    };

    void Astar(int s, int t) {//A*
        priority_queue<Data> q;
        Data st;
        st.now = s; st.pas = 0; st.val = dis[s]; st.route = vector<int>{s};
        q.push(st);
        vector<int> vec;
        while (!q.empty()) {
            Data u = q.top(); q.pop();
            if (u.now == t) {//更新路径数
                :: cnt++;
                if (:: cnt == k) {//最终答案
                    cout << u.route[0];
                    for (int i = 1, sz = u.route.size(); i < sz; i++)
                        cout << '-' << u.route[i];
                    hav = true;
                    return;
                }
            }
            for (int i = head[u.now]; i; i = nxt[i]) {//广搜
                int v = to[i];
                vec = u.route;
                bool visit = false;
                for (int j = 0, sz = vec.size(); j < sz; j++) {//记录是否重复经过
                    if (vec[j] == v) {
                        visit = true;
                        break;
                    }
                }
                if (visit) continue;
                Data nx = u;
                nx.now = v;
                nx.pas = u.pas + val[i];
                nx.val = dis[v] + nx.pas;
                nx.route.push_back(v);
                q.push(nx);
            }
        }
    }
}

int main() {
    cin >> n >> m >> k >> a >> b;
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        G1 :: AddEdge(v, u, w);
        G2 :: AddEdge(u, v, w);
    }
    G1 :: Spfa(b, a);
    G2 :: Astar(a, b);
    if (!hav) cout << "No" << endl;
    return 0;
}
```

7.24 dinic

```
template <typename T>
class flow_graph {
public:
    static constexpr T eps = (T) 1e-9;

    struct edge {
        int from;
        int to;
        T c;
        T f;
    };

    const int n;
    std::vector<edge> edges;
    std::vector<std::vector<int>> g;
    int st;
    int fin;
    T flow;

    flow_graph(int _n, int _st, int _fin) : n(n), st(_st), fin(_fin) {
        assert(0 <= st && st < n && 0 <= fin && fin < n && st != fin);
        g.resize(n);
        flow = 0;
    }

    void clear_flow() {
        for (const edge& e : edges) {
            e.f = 0;
        }
        flow = 0;
    }

    int add(int from, int to, T forward_cap, T backward_cap) {
        assert(0 <= from && from < n && 0 <= to && to < n);
        int id = (int) edges.size();
        g[from].push_back(id);
        edges.push_back({from, to, forward_cap, 0});
        g[to].push_back(id + 1);
        edges.push_back({to, from, backward_cap, 0});
        return id;
    }
};

template <typename T>
class dinic {
public:
    flow_graph<T> &g;
    std::vector<int> ptr;
    std::vector<int> d;
    std::vector<int> q;

    dinic(flow_graph<T> &_g) : g(_g) {
        ptr.resize(g.n);
        d.resize(g.n);
        q.resize(g.n);
    }

    bool expath() {
        fill(d.begin(), d.end(), -1);
        q[0] = g.fin;
        d[g.fin] = 0;
        int beg = 0, end = 1;
        while (beg < end) {
            int i = q[beg++];
            for (int id : g.g[i]) {
                const auto &e = g.edges[id];
                const auto &back = g.edges[id ^ 1];
                if (back.c - back.f > g.eps && d[e.to] != -1) {
                    d[e.to] = d[i] + 1;
                    if (e.to == g.st) {
                        return true;
                    }
                    q[end++] = e.to;
                }
            }
        }
        return false;
    }

    T dfs(int v, int w) {
        if (v == g.fin) {
            return w;
        }
        int &j = ptr[v];
        while (j >= 0) {
            int id = g.g[v][j];
            const auto& e = g.edges[id];
            if (e.c - e.f > g.eps && d[e.to] == d[v] - 1) {
                T t = dfs(e.to, std::min(e.c - e.f), w);
                if (t > g.eps) {
                    g.edges[id].f += t;
                    g.edges[id ^ 1].f -= t;
                    return t;
                }
            }
            j--;
        }
        return 0;
    }

    T max_flow() {
        while (expath()) {
            for (int i = 0; i < g.n; i++) {
                ptr[i] = (int) g.g[i].size() - 1;
            }
            T big_add = 0;
            while (true) {
                T add = dfs(g.st, std::numeric_limits<T>::max());
                if (add <= g.eps) {
                  break;
                }
                big_add += add;
            }
            if (big_add <= g.eps) {
                break;
            }
            g.flow += big_add;
        }
        return g.flow;
    }

    std::vector<bool> min_cut() {
        max_flow();
        std::vector<bool> ret(g.n);
        for (int i = 0; i < g.n; i++) {
            ret[i] = (d[i] != -1);
        }
        return ret;
    }
};
```

7.25 ISAP

```
struct ISAP {
    const static int N = ...;//node size
    struct Edge {
        int from, to, cap, flow;
        bool operator < (const Edge &rhs) const {
            return from < rhs.from || (from == rhs.from && to < rhs.to);
        }
    };
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> g[N];
    bool vis[N];
    int dep[N], cur[N], p[N], num[N];

    void addEdge(int from, int to, int cap) {
        edges.push_back(Edge{from, to, cap, 0});
        edges.push_back(Edge{to, from, 0, 0});
        m = edges.size();
        g[from].push_back(m - 2);
        g[to].push_back(m - 1);
    }

    bool bfs() {
        memset(vis, 0, sizeof(vis));
        queue<int> q; q.push(t); vis[t] = 1, dep[t] = 0;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto &v: g[u]) {
                Edge &e = edges[v ^ 1];
                if (!vis[e.from] && e.cap > e.flow) {
                    dep[e.from] = dep[u] + (vis[e.from] = 1);
                    q.push(e.from);
                }
            }
        }
        return vis[s];
    }

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

    int augment() {
        int u = t, a = INF;
        while (u != s) {
            Edge &e = edges[p[u]];
            a = min(a, e.cap - e.flow);
            u = edges[p[u]].from;
        }
        u = t;
        while (u != s) {
            edges[p[u]].flow += a;
            edges[p[u] ^ 1].flow -= a;
            u = edges[p[u]].from;
        }
        return a;
    }

    int maxFlow(int S, int T) {
        s = S, t = T;
        int flow = 0; bfs();
        memset(num, 0, sizeof(num));
        for (int i = 0; i < n; i++) num[dep[i]]++;
        int u = S;
        memset(cur, 0, sizeof(cur));
        while (dep[S] < n) {
            if (u == T) {
                flow += augment();
                u = S;
            }
            int ok = 0;
            for (int i = cur[u]; i < g[u].size(); i++) {
                Edge &e = edges[g[u][i]];
                if (e.cap > e.flow && dep[u] == dep[e.to] + 1) {
                    ok = 1;
                    p[e.to] = g[u][i];
                    cur[u] = i;
                    u = e.to;
                    break;
                }
            }
            if (!ok) {
                int mn = n - 1;
                for (int i = 0; i < g[u].size(); i++) {
                    Edge &e = edges[g[u][i]];
                    if (e.cap > e.flow) mn = min(mn, dep[e.to]);
                }
                if (--num[dep[u]] == 0) break;
                num[dep[u] = mn + 1]++;
                cur[u] = 0;
                if (u != S) u = edges[p[u]].from;
            }
        }
        return flow;
    }

} flow;
```

7.26 mcmf

```
struct MCMF {
    struct Edge {
    int from, to, cap, flow, cost;
    Edge(int u, int v, int c, int f, int cc)
        : from(u), to(v), cap(c), flow(f), cost(cc) {}
    };
    static constexpr int INF = 1e9;
    int n, m;

    std::vector<Edge> edges;
    std::vector<std::vector<int>> G;
    std::vector<int> inq;
    std::vector<int> d;
    std::vector<int> p;
    std::vector<int> a;
```

```cpp


    MCMF(int n) : n(n), G(n), inq(n), d(n), p(n), a(n) {}

    void add(int from, int to, int cap, int cost) {
        edges.emplace_back(Edge(from, to, cap, 0, cost));
        edges.emplace_back(Edge(to, from, 0, 0, -cost));
        m = int(edges.size());
        G[from].emplace_back(m - 2);
        G[to].emplace_back(m - 1);
    }

    bool spfa(int s, int t, int &flow, int &cost) {
        for (int i = 1; i < n; ++i) d[i] = INF;
        inq.assign(n, 0);
        d[s] = 0;
        inq[s] = 1;
        p[s] = 0;
        std::queue<int> q;
        a[s] = INF;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for (int i = 0; i < int(G[u].size()); ++i) {
                Edge &e = edges[G[u][i]];
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost) {
                    d[e.to] = d[u] + e.cost;
                    p[e.to] = G[u][i];
                    a[e.to] = std::min(a[u], e.cap - e.flow);
                    if (!inq[e.to]) {
                        q.push(e.to);
                        inq[e.to] = 1;
                    }
                }
            }
        }
        if (d[t] == INF) return false;
        flow += a[t];
        cost += d[t] * a[t];
        for (int u = t; u != s; u = edges[p[u]].from) {
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
        }
        return true;
    }

    int MincostMaxflow(int s, int t, int &cost) {
        int flow = 0;
        cost = 0;
        while (spfa(s, t, flow, cost));
        return flow;
    }
};
```

7.27 conclusion.md

8 其他

8.1 simpleMO

```
// O(n*sqrt(n))
const int N = 1e5 + 10;

int a[N], cnt[N], ans[N];
int belong[N];

struct Q {
    int l, r, id;
}q[N];

int Size, bnum;

bool cmp(Q a, Q b) {
    return (belong[a.l] ^ belong[b.l]) ? belong[a.l] < belong[b.l] : belong[a.l] & 1 ? a.r < b.r : a.r > b.r;
}

int now = 0;

inline void add(int pos) {
    if(!cnt[a[pos]]) now++;
    ++cnt[a[pos]];
}

inline void del(int pos) {
    --cnt[a[pos]];
    if(!cnt[a[pos]]) --now;
}

int main() {
    int n, m;
    cin >> n >> m;
    Size = sqrt(n);
    bnum = ceil((double)n / Size);
    for(int i = 1;i <= bnum; i++) {
        for(int j = (i - 1) * Size + 1;j <= i * Size; j++) {
            belong[j] = i;
        }
    }
    for(int i = 1;i <= n; i++) cin >> a[i];
    for(int i = 1;i <= m; i++) {
        cin >> q[i].l >> q[i].r;
        q[i].id = i;
    }
    sort(q + 1, q + m + 1, cmp);
    int l = 1, r = 0;
    for(int i = 1;i <= m; i++) {
        int ql = q[i].l, qr = q[i].r;
        while(r < qr) add(++r);
        while(r > qr) del(r--);
        while(l < ql) del(l++);
        while(l > ql) add(--l);
        ans[q[i].id] = now;
    }
    for(int i = 1;i <= m; i++) cout << ans[i] << endl;
}
```

8.2 modifyMO

```
//带修莫队模板题
//查询[ql, qr]间不同颜色数量，带修改

// O(n^{5/3})

int n, m;
int c[N], cnt[N];
int belong[N], size, totq, totm;
int ans[N], res;

struct Query {
    int l, r, t, id;
    bool operator < (const Query &rhs) const {
        return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] :
               (belong[r] ^ belong[rhs.r] ? belong[r] < belong[rhs.r] : t < rhs.t);
    }
} q[N];

struct Modify {
    int pos, val;
} modify[N];

void add(int x) {
    if (!cnt[c[x]]) res++;
    cnt[c[x]]++;
}

void del(int x) {
    cnt[c[x]]--;
    if (!cnt[c[x]]) res--;
}

void upd(int x, int ql, int qr) {
    int pos = modify[x].pos;
    if (ql <= pos && pos <= qr) {
        cnt[c[pos]]--; if (!cnt[c[pos]]) res--;
        if (!cnt[modify[x].val]) res++; cnt[modify[x].val]++;
    }
    swap(modify[x].val, c[pos]);//□ □ □ ĵ□ □ □ □ ´□ □ Ļ□ ȥ
}

int main() {
#ifdef ACM_LOCAL
    freopen("input.in", "r", stdin);
    freopen("output.out", "w", stdout);
#endif
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &c[i]);
    for (int i = 1; i <= m; i++) {
        char op[10]; scanf("%s", op);
        if (op[0] == 'Q') {
            int ql, qr; scanf("%d%d", &ql, &qr); totq++;
            q[totq] = Query{ql, qr, totm, totq};
        }
        else {
            int pos, val; scanf("%d%d", &pos, &val); totm++;
            modify[totm] = Modify{pos, val};
        }
    }

    //size = N ^ (2 / 3), (N * totm) ^ (1 / 3)
    size = ceil(pow(n, (long double)2.0 / 3)); int num = ceil((long double)n / size);
    for (int i = 1, j = 1; i <= num; i++)
        while (j <= i * size && j <= n)
            belong[j++] = i;

    sort(q + 1, q + 1 + totq);

    int l = 1, r = 0, t = 0;
    for (int i = 1; i <= totq; i++) {
        int ql = q[i].l, qr = q[i].r, qt = q[i].t;
        while (l < ql) del(l++);
        while (l > ql) add(--l);
        while (r < qr) add(++r);
        while (r > qr) del(r--);
        while (t < qt) upd(++t, ql, qr);
        while (t > qt) upd(t--, ql, qr);
        ans[q[i].id] = res;
    }

    for (int i = 1; i <= totq; i++) printf("%d\n", ans[i]);

    return 0;
}
```

8.3 rollbackMO

```
//问题可以莫队（询问可以离线，不带修改）
//区间伸长的时候很好维护信息
//区间缩短的时候不太好维护信息（如最大值，删除以后不知道次大值是多少）
// O(nsqrt(n))

struct Hash {
    int b[N], tot;
    void init() { tot = 0; }
    void insert(int x) { b[++tot] = x; }
    void build() {
        sort(b + 1, b + 1 + tot);
        tot = unique(b + 1, b + 1 + tot) - (b + 1);
    }
    int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
} ha;

int n, m;
int c[N], pos[N], cnt[N], cntt[N];
int belong[N], sizz;
ll ans[N], res;

struct Query {
    int l, r, id;
    bool operator < (const Query &rhs) const {
        return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] : r < rhs.r;
    }
} q[N];

ll bruteForce(int ql, int qr) {
    ll result = 0;
    for (int i = ql; i <= qr; i++) {
        cntt[pos[i]]++;
        result = max(result, 1ll * c[i] * cntt[pos[i]]);
    }
    for (int i = ql; i <= qr; i++) cntt[pos[i]]--;
    return result;
}

void add(int x) {
    cnt[pos[x]]++;
    res = max(res, 1ll * c[x] * cnt[pos[x]]);
}

void del(int x) {
    cnt[pos[x]]--;
}

int main() {


    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &c[i]), ha.insert(c[i]);
    ha.build();
    for (int i = 1; i <= n; i++) pos[i] = ha.pos(c[i]);

    sizz = sqrt(n); int num = ceil((long double)n / sizz);
    for (int i = 1, j = 1; i <= num; i++)
        while (j <= i * sizz && j <= n)
            belong[j++] = i;

    for (int i = 1; i <= m; i++) scanf("%d%d", &q[i].l, &q[i].r), q[i].id = i;
    sort(q + 1, q + 1 + m);

    for (int i = 1, j = 1; i <= num; i++) {
        memset(cnt, 0, sizeof(cnt));
        int right = min(i * sizz, n);
        res = 0;
        for (int l = right + 1, r = right; j <= m && belong[q[j].l] == i; j++, l = right + 1) {
            int ql = q[j].l, qr = q[j].r;
            if (qr - ql + 1 <= sizz) {
                ans[q[j].id] = bruteForce(ql, qr);
                continue;
            }
            while (r < qr) add(++r);
            ll tmp = res;
            while (l > ql) add(--l);
            ans[q[j].id] = res;
            res = tmp;
            while (l < right + 1) del(l++);
        }
    }

    for (int i = 1; i <= m; i++) printf("%lld\n", ans[i]);

    return 0;
}
```

8.4 treeMO

```
const int N = 1e5 + 10;

struct Hash {
    int b[N], tot;
    void init() { tot = 0; }
    void insert(int x) { b[++tot] = x; }
    void build() {
        sort(b + 1, b + 1 + tot);
        tot = unique(b + 1, b + 1 + tot) - (b + 1);
    }
    int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
} ha;

int n, m;
int c[N], cnt[N];
vector<int> g[N];
int st[N], ed[N], dfnt, nodeOf[N << 1], tag[N];
int belong[N], sizz;
int ans[N], res;

struct Query {
    int l, r, id, k;
    bool operator < (const Query &rhs) const {
        return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] : r < rhs.r;
    }
} q[N];

int son[N], siz[N], top[N], fa[N], dep[N];
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    int max_son = -1; nodeOf[st[u] = ++dfnt] = u;
    for (auto &v: g[u])
        if (v != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (max_son < siz[v])
                son[u] = v, max_son = siz[v];
        }
    nodeOf[ed[u] = ++dfnt] = u;
}
void dfs2(int u, int topf) {
    top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (auto &v: g[u])
        if (v != fa[u] && v != son[u]) dfs2(v, v);
}
int lca(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        x = fa[top[x]];
    }
    return dep[x] < dep[y] ? x : y;
}

void upd(int x) {
    x = nodeOf[x];
    if (tag[x]) {
        cnt[c[x]]--;
        if (!cnt[c[x]]) res--;
    }
    else {
        if (!cnt[c[x]]) res++;
        cnt[c[x]]++;
    }
    tag[x] ^= 1;
}

int main() {
#ifdef ACM_LOCAL
    freopen("input.in", "r", stdin);
    freopen("output.out", "w", stdout);
#endif
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &c[i]), ha.insert(c[i]);
    ha.build();
    for (int i = 1; i <= n; i++) c[i] = ha.pos(c[i]);

    for (int i = 1; i < n; i++) {
        int u, v; scanf("%d%d", &u, &v);
        g[u].push_back(v); g[v].push_back(u);
    }
    int rt = 1; dfs(rt, 0); dfs2(rt, rt);

    sizz = sqrt(dfnt); int num = ceil((long double)dfnt / sizz);
    for (int i = 1, j = 1; i <= num; i++)
        while (j <= i * sizz && j <= dfnt)
            belong[j++] = i;
    for (int i = 1; i <= m; i++) {
        int u, v; scanf("%d%d", &u, &v);
        int tlca = lca(u, v);
        if (st[u] > st[v]) swap(u, v);
        if (u == tlca) q[i] = Query{st[u], st[v], i, 0};
        else q[i] = Query{ed[u], st[v], i, tlca};

    }
    sort(q + 1, q + 1 + m);

    int l = 1, r = 0;
    for (int i = 1; i <= m; i++) {
        int ql = q[i].l, qr = q[i].r;
        while (l < ql) upd(l++);
        while (l > ql) upd(--l);
        while (r < qr) upd(++r);
        while (r > qr) upd(r--);
        ans[q[i].id] = res + (q[i].k ? (cnt[c[q[i].k]] == 0) : 0);
    }

    for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);

    return 0;
```

```
}
```

8.5 fastIO

```
namespace FastIO {
    char gc(void) {
        const int S = 1 << 17;
        static char buf[S], *s = buf, *t = buf;
        if (s == t) t = buf + fread(s = buf, 1, S, stdin);
        if (s == t) return EOF;
        return *s++;
    }

    int read(void) {
        int a = 0, b = 1, c = gc();
        for (; !isdigit(c); c = gc()) b ^= (c == '-');
        for (; isdigit(c); c = gc()) a = a * 10 + c - '0';
        return b ? a : -a;
    }
}
using namespace FastIO;


static struct FastInput {
    static constexpr int BUF_SIZE = 1 << 20;
    char buf[BUF_SIZE];
    size_t chars_read = 0;
    size_t buf_pos = 0;
    FILE *in = stdin;
    char cur = 0;

    inline char get_char() {
        if (buf_pos >= chars_read) {
            chars_read = fread(buf, 1, BUF_SIZE, in);
            buf_pos = 0;
            buf[0] = (chars_read == 0 ? -1 : buf[0]);
        }
        return cur = buf[buf_pos++];
    }

    inline void tie(int) {}

    inline explicit operator bool() {
        return cur != -1;
    }

    inline static bool is_blank(char c) {
        return c <= ' ';
    }

    inline bool skip_blanks() {
        while (is_blank(cur) && cur != -1) {
            get_char();
        }
        return cur != -1;
    }

    inline FastInput& operator>>(char& c) {
        skip_blanks();
        c = cur;
        return *this;
    }

    inline FastInput& operator>>(std::string& s) {
        if (skip_blanks()) {
            s.clear();
            do {
                s += cur;
            } while (!is_blank(get_char()));
        }
        return *this;
    }

    template <typename T>
    inline FastInput& read_integer(T& n) {
        // unsafe, doesn't check that characters are actually digits
        n = 0;
        if (skip_blanks()) {
            int sign = +1;
            if (cur == '-') {
                sign = -1;
                get_char();
            }
            do {
                n += n + (n << 3) + cur - '0';
            } while (!is_blank(get_char()));
            n *= sign;
        }
        return *this;
    }

    template <typename T>
    inline typename std::enable_if<std::is_integral<T>::value, FastInput&>::type operator>>(T& n) {
        return read_integer(n);
    }

    #if !defined(_WIN32) || defined(_WIN64)
    inline FastInput& operator>>(__int128& n) {
        return read_integer(n);
    }
    #endif

    template <typename T>
    inline typename std::enable_if<std::is_floating_point<T>::value, FastInput&>::type operator>>(T& n) {
        // not sure if really fast, for compatibility only
        n = 0;
        if (skip_blanks()) {
            std::string s;
            (*this) >> s;
            sscanf(s.c_str(), "%lf", &n);
        }
        return *this;
    }
} fast_input;

```

```
#define cin fast_input

static struct FastOutput {
    static constexpr int BUF_SIZE = 1 << 20;
    char buf[BUF_SIZE];
    size_t buf_pos = 0;
    static constexpr int TMP_SIZE = 1 << 20;
    char tmp[TMP_SIZE];
    FILE *out = stdout;

    inline void put_char(char c) {
        buf[buf_pos++] = c;
        if (buf_pos == BUF_SIZE) {
            fwrite(buf, 1, buf_pos, out);
            buf_pos = 0;
        }
    }

    ~FastOutput() {
        fwrite(buf, 1, buf_pos, out);
    }

    inline FastOutput& operator<<(char c) {
        put_char(c);
        return *this;
    }

    inline FastOutput& operator<<(const char* s) {
        while (*s) {
            put_char(*s++);
        }
        return *this;
    }

    inline FastOutput& operator<<(const std::string& s) {
        for (int i = 0; i < (int) s.size(); i++) {
            put_char(s[i]);
        }
        return *this;
    }

    template <typename T>
    inline char* integer_to_string(T n) {
        // beware of TMP_SIZE
        char* p = tmp + TMP_SIZE - 1;
        if (n == 0) {
            *--p = '0';
        } else {
            bool is_negative = false;
            if (n < 0) {
                is_negative = true;
                n = -n;
            }
            while (n > 0) {
                *--p = (char) ('0' + n % 10);
                n /= 10;
            }
            if (is_negative) {
                *--p = '-';
```

```cpp
            }
        }
        return p;
    }

    template <typename T>
    inline typename std::enable_if<std::is_integral<T>::value, char*>::type stringify(T n) {
        return integer_to_string(n);
    }

    #if !defined(_WIN32) || defined(_WIN64)
    inline char* stringify(__int128 n) {
        return integer_to_string(n);
    }
    #endif

    template <typename T>
    inline typename std::enable_if<std::is_floating_point<T>::value, char*>::type stringify(T n) {
        sprintf(tmp, "%.17f", n);
        return tmp;
    }

    template <typename T>
    inline FastOutput& operator<<(const T& n) {
        auto p = stringify(n);
        for (; *p != 0; p++) {
            put_char(*p);
        }
        return *this;
    }
} fast_output;

#define cout fast_output
```

8.6 simulatedAnnealing

```cpp
const double DOWN = 0.996;
const double START_T = 5000;
double ansx, ansy, ansz, anse;

void initAns() {
    //初始化一个答案点(可以选任意点)
}

double getEnergy(double x, double y, double z) {
    //具体分析题目
}

void SA() {
    double T = START_T;
    while (T > 1e-15) {
        double newx = ansx + (rand() * 2 - RAND_MAX) * T;
        double newy = ansy + (rand() * 2 - RAND_MAX) * T;
        double newz = ansz + (rand() * 2 - RAND_MAX) * T;
        double newe = getEnergy(newx, newy, newz);
        double delta = newe - anse;
        if (delta < 0) ansx = newx, ansy = newy, ansz = newz, anse = newe;
        else if (exp(-delta / T) * RAND_MAX > rand())
            ansx = newx, ansy = newy, ansz = newz;
        T *= DOWN;
    }
}

void solve() {
    initAns();
    while ((double) clock() / CLOCKS_PER_SEC < 2.0) SA();
}
```

8.7 Cantor

```

//主要应用为将N维的排列状态压缩成数字id
//然后需要知道具体状态时用逆Cantor得到

int N;
int a[MAX], c[MAX];

void upd(int p, int k) { for (; p <= N; p += lowbit(p)) c[p] += k; }
int query(int p) {
    int res = 0;
    for (; p; p -= lowbit(p)) res += c[p];
    return res;
}

int cantor() {
    //ans = 1 + \sum_{i = 1} ^ {N} fac[N - i] * (\sum_{j = i + 1} ^ {N} x[i] > x[j])
    int res = 0, fac = 1;
    for (int i = N; i >= 1; i--) {
        upd(a[i], 1);
        res = (res + 1ll * fac * query(a[i] - 1) % mod) % mod;
        fac = 1ll * fac * (N - i + 1) % mod;
    }
    return res + 1;
}

//逆Cantor
#define lc  u<<1
#define rc  u<<1|1
#define mid (l+r)/2
int sum[MAX << 4];
void push_up(int u) { sum[u] = sum[lc] + sum[rc]; }
void build(int u, int l, int r) {
    if (l == r) {
        sum[u] = 1;
        return;
    }
    build(lc, l, mid); build(rc, mid + 1, r);
    push_up(u);
}
int query(int u, int l, int r, int k) {//查找第k大并且删除该数
    sum[u]--;
    if (l == r) return l;
    if (k <= sum[lc]) return query(lc, l, mid, k);
    else return query(rc, mid + 1, r, k - sum[lc]);
}

vector<int> inCantor(int x, int n) {
    x--;
    vector<int> res;
    ll fac = 1;
    build(1, 1, n);
    for (int i = 1; i <= n; i++) fac = fac * i;
    for (int i = 1; i <= n; i++) {
        fac = fac / (n - i + 1);
        int k = x / fac + 1;//比当前这位大的有x / fac位
        res.push_back(query(1, 1, n, k));//找到没被选的第k大
        x %= fac;
    }
    return res;
}
```

8.8 BigInteger

```
struct BigInteger {
    typedef unsigned long long LL;

    static const int BASE = 100000000;
    static const int WIDTH = 8;
    vector<int> s;

    BigInteger& clean() { while (!s.back() && s.size()>1)s.pop_back(); return *this; }
    BigInteger(LL num = 0) { *this = num; }
    BigInteger(string s) { *this = s; }
    BigInteger& operator = (long long num) {
        s.clear();
        do {
            s.push_back(num % BASE);
            num /= BASE;
        } while (num > 0);
        return *this;
    }
    BigInteger& operator = (const string& str) {
        s.clear();
        int x, len = (str.length() - 1) / WIDTH + 1;
        for (int i = 0; i < len; i++) {
            int end = str.length() - i*WIDTH;
            int start = max(0, end - WIDTH);
            sscanf(str.substr(start, end - start).c_str(), "%d", &x);
            s.push_back(x);
        }
        return (*this).clean();
    }

    BigInteger operator + (const BigInteger& b) const {
        BigInteger c; c.s.clear();
        for (int i = 0, g = 0; ; i++) {
            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
            int x = g;
            if (i < s.size()) x += s[i];
            if (i < b.s.size()) x += b.s[i];
            c.s.push_back(x % BASE);
            g = x / BASE;
        }
        return c;
    }
    BigInteger operator - (const BigInteger& b) const {
        assert(b <= *this); // 减数不能大于被减数
        BigInteger c; c.s.clear();
        for (int i = 0, g = 0; ; i++) {
            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
            int x = s[i] + g;
            if (i < b.s.size()) x -= b.s[i];
            if (x < 0) { g = -1; x += BASE; }
            else g = 0;
            c.s.push_back(x);
        }
        return c.clean();
    }
    BigInteger operator * (const BigInteger& b) const {
        int i, j; LL g;
        vector<LL> v(s.size() + b.s.size(), 0);
        BigInteger c; c.s.clear();
        for (i = 0; i<s.size(); i++) for (j = 0; j<b.s.size(); j++) v[i + j] += LL(s[i])*b.s[j];
        for (i = 0, g = 0; ; i++) {
            if (g == 0 && i >= v.size()) break;
            LL x = v[i] + g;
            c.s.push_back(x % BASE);
            g = x / BASE;
        }
        return c.clean();
    }
    BigInteger operator / (const BigInteger& b) const {
        assert(b > 0);  // 除数必须大于0
        BigInteger c = *this;       // 商:主要是让c.s和(*this).s的vector一样大
        BigInteger m;               // 余数:初始化为0
        for (int i = s.size() - 1; i >= 0; i--) {
            m = m*BASE + s[i];
            c.s[i] = bsearch(b, m);
            m -= b*c.s[i];
        }
        return c.clean();
    }
    BigInteger operator % (const BigInteger& b) const { //方法与除法相同
        BigInteger c = *this;
        BigInteger m;
        for (int i = s.size() - 1; i >= 0; i--) {
            m = m*BASE + s[i];
            c.s[i] = bsearch(b, m);
            m -= b*c.s[i];
        }
        return m;
    }
    // 二分法找出满足bx<=m的最大的x
    int bsearch(const BigInteger& b, const BigInteger& m) const {
        int L = 0, R = BASE - 1, x;
        while (1) {
            x = (L + R) >> 1;
            if (b*x <= m) { if (b*(x + 1)>m) return x; else L = x; }
            else R = x;
        }
    }
    BigInteger& operator += (const BigInteger& b) { *this = *this + b; return *this; }
    BigInteger& operator -= (const BigInteger& b) { *this = *this - b; return *this; }
    BigInteger& operator *= (const BigInteger& b) { *this = *this * b; return *this; }
    BigInteger& operator /= (const BigInteger& b) { *this = *this / b; return *this; }
    BigInteger& operator %= (const BigInteger& b) { *this = *this % b; return *this; }

    bool operator < (const BigInteger& b) const {
        if (s.size() != b.s.size()) return s.size() < b.s.size();
        for (int i = s.size() - 1; i >= 0; i--)
            if (s[i] != b.s[i]) return s[i] < b.s[i];
        return false;
    }
    bool operator >(const BigInteger& b) const { return b < *this; }
    bool operator<=(const BigInteger& b) const { return !(b < *this); }
    bool operator>=(const BigInteger& b) const { return !(*this < b); }
    bool operator!=(const BigInteger& b) const { return b < *this || *this < b; }
    bool operator==(const BigInteger& b) const { return !(b < *this) && !(b > *this); }
};

ostream& operator << (ostream& out, const BigInteger& x) {
    out << x.s.back();
    for (int i = x.s.size() - 2; i >= 0; i--) {
        char buf[20];
        sprintf(buf, "%08d", x.s[i]);
        for (int j = 0; j < strlen(buf); j++) out << buf[j];
    }
    return out;
}

istream& operator >> (istream& in, BigInteger& x) {
    string s;
    if (!(in >> s)) return in;
    x = s;
    return in;
}

int main()
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        BigInteger a, b;
        cin >> a >> b;
        cout << a + b << endl;
    }
}
```

8.9 debug

```
using std::string;
using std::to_string;
template <typename A, typename B>
std::string to_string(std::pair<A, B> p);

template <typename A, typename B, typename C>
std::string to_string(std::tuple<A, B, C> p);
```

```cpp

template <typename A, typename B, typename C, typename D>
std::string to_string(std::tuple<A, B, C, D> p);

std::string to_string(const std::string& s) {
  return '"' + s + '"';
}

std::string to_string(const char* s) {
  return to_string((std::string) s);
}

std::string to_string(bool b) {
  return (b ? "true" : "false");
}

std::string to_string(std::vector<bool> v) {
  bool first = true;
  std::string res = "{";
  for (int i = 0; i < static_cast<int>(v.size()); i++) {
    if (!first) {
      res += ", ";
    }
    first = false;
    res += to_string(v[i]);
  }
  res += "}";
  return res;
}

template <size_t N>
std::string to_string(std::bitset<N> v) {
  std::string res = "";
  for (size_t i = 0; i < N; i++) {
    res += static_cast<char>('0' + v[i]);
  }
  return res;
}

template <typename A>
std::string to_string(A v) {
  bool first = true;
  std::string res = "{";
  for (const auto &x : v) {
    if (!first) {
      res += ", ";
    }
    first = false;
    res += to_string(x);
  }
  res += "}";
  return res;
}


template <typename A, typename B>
std::string to_string(std::pair<A, B> p) {
  return "(" + to_string(p.first) + ", " + to_string(p.second) + ")";
}
```

```
template <typename A, typename B, typename C>
std::string to_string(std::tuple<A, B, C> p) {
  return "(" + to_string(std::get<0>(p)) + ", " + to_string(std::get<1>(p)) + ", " + to_string(std::get<2>(p)) + ")";
}

template <typename A, typename B, typename C, typename D>
std::string to_string(std::tuple<A, B, C, D> p) {
  return "(" + to_string(std::get<0>(p)) + ", " + to_string(std::get<1>(p)) + ", " + to_string(std::get<2>(p)) + ", " + to_string(std::get<3>(p)) + ")";
}

void debug_out() { std::cerr << std::endl; }

template <typename Head, typename... Tail>
void debug_out(Head H, Tail... T) {
  std::cerr << " " << to_string(H);
  debug_out(T...);
}

#ifdef LOCAL
#define debug(...) std::cerr << "[" << #__VA_ARGS__ << "]:", debug_out(__VA_ARGS__)
#else
#define debug(...) 42
#endif
```

8.10 int128

```
using i128 = __int128;

std::istream &operator>>(std::istream &is, i128 &n) {
    n = 0;
    std::string s;
    is >> s;
    for (auto c : s) {
        n = 10 * n + c - '0';
    }
    return is;
}

std::ostream &operator<<(std::ostream &os, i128 n) {
    if (n == 0) {
        return os << 0;
    }
    std::string s;
    while (n > 0) {
        s += '0' + n % 10;
        n /= 10;
    }
    std::reverse(s.begin(), s.end());
    return os << s;
}
```