1. Default Setting

1.1 VS

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
#define pq priority_queue
#define endl "\n"
const ll INF = 1e18 + 7;
const int inf = 1e9 + 7;

void solve(int tc) {

}
int main() {
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int T = 1;
    //cin >> T;
    for (int tc = 1; tc <= T; tc++) {
        solve(tc);
    }
    return 0;
}
```

1.2 pbds(gcc)

```
#include<ext/rope>
using namespace __gnu_cxx;
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
typedef tree<ll, null_type, less_equal<ll>, rb_tree_tag,
tree_order_statistics_node_update> ordered_multiset;
```

2. Hasher

```
struct VectorHasher { // unordered_set<vector<int>,VectorHasher>
    size_t operator()(const vector<int>& V) const {
        size_t hash = V.size();
        for (auto& i : V) {
            hash ^= i + 0x9e3779b9 + (hash << 6) + (hash >> 2);
        }
        return hash;
    }
};
struct PiiHasher { // unordered_set<pii, PiiHasher>
    size_t operator()(const pii& x) const {
        return hash<long long>()(((long long)x.first) ^ (((long long)x.second) <<
32));
    }
};
```

3. Matrix multiplication

```
vector<vector<ll>> matmul(const vector<vector<ll>>& a, const vector<vector<ll>>& b,
ll mod = 0) { // 행렬 곱
    assert(a[0].size() == b.size());
    vector<vector<ll>> ret(a.size(), vector<ll>(b[0].size(), 0));
    for (int i = 0; i < a.size(); i++) {
        for (int j = 0; j < b[0].size(); j++) {
            for (int k = 0; k < a[0].size(); k++) {
                ret[i][j] += a[i][k] * b[k][j];
                if (mod) ret[i][j] %= mod;
            }
        }
    }
    return ret;
}

vector<vector<ll>> powmat(const vector<vector<ll>>& mat, ll n, ll mod = 0) { // 행렬
거듭제곱
    if (n == 1) return mat;
    if (n % 2) {
        return matmul(powmat(mat, n - 1, mod), mat, mod);
    }
    vector<vector<ll>> half = powmat(mat, n / 2, mod);
    return matmul(half, half, mod);
}

ll fibo(ll n, ll mod = 0) { // 행렬 거듭제곱 이용 피보나치
    if (n < 2) {
        return n;
    }
    vector<vector<ll>> ret = matmul(powmat({ {1,1},{1,0} }, n - 1, mod), { {1},{0} },
mod);
    return ret[0][0];
}
```

4. KMP(String)

```
vector<int> getPi(const string s) { // KMP 선행 작업, pattern->전이함수
    vector<int> pi(s.size(), 0);
    int j = 0;
    for (int i = 1; i < s.size(); i++) {
        while ((j > 0) && (s[i] != s[j])) {
            j = pi[j - 1];
        }
        if (s[i] == s[j]) {
            j++;
            pi[i] = j;
        }
    }
    return pi;
}

vector<int> kmp(const string& s, const string& t, const vector<int>& pi) { //KMP,
kmp(pattern, txt, pi)
    vector<int> result;
    int j = 0;
    for (int i = 0; i < t.size(); i++) {
        while ((j > 0) && (t[i] != s[j])) {
            j = pi[j - 1];
        }
        if (t[i] == s[j]) {
            if (j == s.size() - 1) {
                result.push_back(i - s.size() + 1);
                j = pi[j];
            }
            else {
                j++;
            }
        }
    }
    return result;
}
```

4.1 KMP(sequence)

```
vector<int> getPi(const vector<int>& pattern) { // 수열용 KMP 선행작업
    vector<int> pi(pattern.size(), 0);
    int j = 0;
    for (int i = 1; i < pattern.size(); i++) {
        while (j > 0 && pattern[i] != pattern[j]) {
            j = pi[j - 1];
        }
        if (pattern[i] == pattern[j]) {
            j++;
            pi[i] = j;
        }
    }
    return pi;
}

vector<int> kmp(const vector<int>& pattern, const vector<int>& text, const
vector<int>& pi) { // 수열용 KMP
    vector<int> result;
    int j = 0;
    for (int i = 0; i < text.size(); i++) {
        while (j > 0 && text[i] != pattern[j]) {
            j = pi[j - 1];
        }
        if (text[i] == pattern[j]) {
            if (j == pattern.size() - 1) {
                result.push_back(i - pattern.size() + 1);
                j = pi[j];
            }
            else {
                j++;
            }
        }
    }
    return result;
}
```

5. Manacher(String)

```
string manacher_preprocess(string& s) { // 매내처 선행작업

    string ret = "";
    char tmp = '#';
    for (char x : s) {
        ret += tmp;
        ret += x;
    }
    ret += tmp;
    return ret;
}

vector<ll> manacher(string& s) { // O(s.length())
    int N = s.length();
    vector<ll> ret(N, 0);
    ll r = 0, p = 0;
    for (int i = 0; i < N; i++) {
        if (i <= r) ret[i] = min(ret[2 * p - i], r - i);
        else ret[i] = 0;

        while (i - ret[i] - 1 >= 0 && i + ret[i] + 1 < N && s[i - ret[i] - 1] == s[i
+ ret[i] + 1]) ret[i] += 1;
```

```
        if (r < i + ret[i]) {
            r = i + ret[i];
            p = i;
        }

    }
    return ret;
}
```

5.1 Manacher(sequence)

```
vector<int> manacher_preprocess(const vector<int>& seq) { // 수열용 매내처 선행작업
    vector<int> ret;
    ret.push_back(-1); // Sentinel value to simulate '#'
    for (int x : seq) {
        ret.push_back(x);
        ret.push_back(-1); // Sentinel value between elements
    }
    return ret;
}

vector<int> manacher(const vector<int>& seq) { // 수열용 매내처
    int N = seq.size();
    vector<int> ret(N, 0);
    int r = 0, p = 0;
    for (int i = 0; i < N; i++) {
        if (i <= r) ret[i] = min(ret[2 * p - i], r - i);
        else ret[i] = 0;

        while (i - ret[i] - 1 >= 0 && i + ret[i] + 1 < N && seq[i - ret[i] - 1] ==
seq[i + ret[i] + 1]) {
            ret[i] += 1;
        }

        if (r < i + ret[i]) {
            r = i + ret[i];
            p = i;
        }
    }
    return ret;
}
```

6. GCD & LCM

```
ll gcd(ll a, ll b) {
    if (a < b) swap(a, b);
    while (b != 0) {
        ll  n = a % b;
        a = b;
        b = n;
    }
    return a;
}
ll lcm(ll a, ll b) {
    ll g = gcd(a, b);
    return a / g * b;
}
ll xgcd(ll a, ll b) {
    ll r1 = a; ll r2 = b;
    ll s1 = 1; ll s2 = 0;
    ll t1 = 0; ll t2 = 1;
    while (1) {
        ll q = r1 / r2;
        ll r = r1 - q * r2;
        ll s = s1 - q * s2;
        ll t = t1 - q * t2;
        if (r == 0) return s2;
        r1 = r2; r2 = r;
        s1 = s2; s2 = s;
        t1 = t2; t2 = t;
    }
}
```

7. Powmod

```
ll powmod(ll x, ll n, ll mod) {
    if (n == 0) return 1;
    if (n % 2) return x * powmod(x, n - 1, mod) % mod;
    ll half = powmod(x, n / 2, mod);
    return half * half % mod;
}
ll modinv(ll x, ll mod) { // when mod is primenum
    return powmod(x, mod - 2, mod);
}
```

8. PopCount

```
int popcount(uint n) { // https://blog.naver.com/jinhan814/222540111549
    n = (n >> 1 & 0x55555555) + (n & 0x55555555);
    n = (n >> 2 & 0x33333333) + (n & 0x33333333);
    n = (n >> 4 & 0x0F0F0F0F) + (n & 0x0F0F0F0F);
    n = (n >> 8 & 0x00FF00FF) + (n & 0x00FF00FF);
    n = (n >> 16 & 0x0000FFFF) + (n & 0x0000FFFF);
    return n;
}
```

9. CCW & ConvexHull & Line-segment intersection

9.1 CCW

```
int ccw(const pair<int, int>& p1, const pair<int, int>& p2, const pair<int, int>& p3)
{ // CCW (Counter Clockwise) 판별 함수
    long long cross = 1LL * (p2.first - p1.first) * (p3.second - p1.second) -
        1LL * (p2.second - p1.second) * (p3.first - p1.first);
    return (cross > 0) - (cross < 0); // 1: CCW, -1: CW, 0: Collinear
}
```

9.2 ConvexHull

```
vector<pair<int, int>> convexHull(vector<pair<int, int>> points) { // 컨벡스 헐 계산
함수
    if (points.size() <= 2) return points; // 점이 2개 이하인 경우 그대로 반환

    // 좌표를 x 기준, 같으면 y 기준으로 정렬
    sort(points.begin(), points.end());

    vector<pair<int, int>> lower, upper;

    // 아래쪽 헐 계산
    for (const auto& point : points) {
        while (lower.size() >= 2 && ccw(lower[lower.size() - 2], lower[lower.size() -
1], point) <= 0) {
            lower.pop_back();
        }
        lower.push_back(point);
    }

    // 위쪽 헐 계산
    for (auto it = points.rbegin(); it != points.rend(); ++it) {
        while (upper.size() >= 2 && ccw(upper[upper.size() - 2], upper[upper.size() -
1], *it) <= 0) {
            upper.pop_back();
        }
        upper.push_back(*it);
    }

    // 마지막 점이 중복되므로 제거
    lower.pop_back();
    upper.pop_back();

    // 아래쪽과 위쪽을 합쳐 시계 방향으로 반환
    lower.insert(lower.end(), upper.begin(), upper.end());
    return lower;
}
```

9.3 Line-segment intersection

```
bool intersect(pll A, pll B, pll C, pll D) { // 선분 교차 판정 A-B , C-D
    if (A > B) swap(A, B);
    if (C > D) swap(C, D);

    ll l1 = ccw(A, B, C) * ccw(A, B, D);
    ll l2 = ccw(C, D, A) * ccw(C, D, B);

    if (l1 == 0 && l2 == 0) {
        return A <= D && C <= B;
    }
    return l1 <= 0 && l2 <= 0;
}
```

10. Flow

10.1 Maximum Flow (Dinic)

```
struct MF { // Dinic O(V^2*E)
    struct Edge {
        int to, rev;
        long long flow, cap;
        Edge(int to, int rev, long long cap) : to(to), rev(rev), flow(0), cap(cap) {}
    };

    int n;
    vector<vector<Edge>> adj; // get이후 edge vector를 이용해서 trace, cap<=flow인
간선은 continue하며 bfs.
    vector<int> level, ptr;

    MF(int n) : n(n), adj(n), level(n), ptr(n) {}

    void addEdge(int u, int v, long long cap, bool directed = true) {
        if (cap == 0) return;
        adj[u].emplace_back(v, adj[v].size(), cap);
        if (!directed) adj[v].emplace_back(u, adj[u].size() - 1, cap);
        else adj[v].emplace_back(u, adj[u].size() - 1, 0);
    }

    bool bfs(int source, int sink) {
        fill(level.begin(), level.end(), -1);
        level[source] = 0;
        queue<int> q;
        q.push(source);

        while (!q.empty()) {
            int node = q.front();
```

```
            q.pop();
            for (Edge& e : adj[node]) {
                if (level[e.to] == -1 && e.flow < e.cap) { // 잔여 용량이 있는 간선을
찾음
                    level[e.to] = level[node] + 1;
                    q.push(e.to);
                }
            }
        }
        return level[sink] != -1;
    }

    long long dfs(int node, int sink, long long pushed) {
        if (pushed == 0) return 0;
        if (node == sink) return pushed;

        for (int& cid = ptr[node]; cid < adj[node].size(); cid++) {
            Edge& e = adj[node][cid];
            if (level[e.to] != level[node] + 1 || e.flow == e.cap) continue;

            long long tr = dfs(e.to, sink, min(pushed, e.cap - e.flow));
            if (tr == 0) continue;

            e.flow += tr;
            adj[e.to][e.rev].flow -= tr;
            return tr;
        }
        return 0;
    }
    long long get(int source, int sink) {
        long long flow = 0;
        while (bfs(source, sink)) {
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(source, sink, LLONG_MAX)) {
                flow += pushed;
            }
        }
        return flow;
    }
    void trace(int src) {
        queue<int> q;
        q.push(src);
        vector<bool> visited(n, false);
        visited[src] = true;
        vector<int> ret;
        while (!q.empty()) {
            int cur = q.front();
            q.pop();
            for (Edge& e : adj[cur]) {
                if (e.cap <= e.flow) continue;
                if (visited[e.to]) continue;
                q.push(e.to);
                ret.push_back(e.to);
                visited[e.to] = true;
            }
        }
        cout << ret.size() << " ";
        for (int x : ret) {
            cout << x << " ";
        }
        cout << endl;
    }
};
```

10.2 MCMF (SPFA)

```
struct MCMF { // SPFA O(VEf) - Practically Ef
    struct Edge {
        int to, rev;
        long long cap, flow, cost;
        Edge(int to, int rev, long long cap, long long cost) : to(to), rev(rev),
cap(cap), flow(0), cost(cost) {}
    };

    int n;
    vector<vector<Edge>> adj;
    vector<long long> dist;
    vector<int> parent, parentEdge;
    vector<bool> inQueue;
    int source, sink;

    MCMF(int n) : n(n), adj(n), dist(n), parent(n), parentEdge(n), inQueue(n),
source(-1), sink(-1) {}

    void setSource(int s) {
        source = s;
    }

    void setSink(int t) {
        sink = t;
    }

    void addEdge(int u, int v, long long cap, long long cost, bool directed = true) {
        if (cap == 0) return;
        adj[u].emplace_back(v, adj[v].size(), cap, cost);
        adj[v].emplace_back(u, adj[u].size() - 1, 0, -cost); // 역방향 간선
        if (!directed) {
            adj[v].emplace_back(u, adj[u].size(), cap, cost);
            adj[u].emplace_back(v, adj[v].size() - 1, 0, -cost);
        }
    }
```

```
bool spfa() {
    fill(dist.begin(), dist.end(), LLONG_MAX);
    fill(inQueue.begin(), inQueue.end(), false);
    queue<int> q;
    dist[source] = 0;
    inQueue[source] = true;
    q.push(source);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inQueue[u] = false;

        for (int i = 0; i < adj[u].size(); ++i) {
            Edge& e = adj[u][i];
            if (e.flow < e.cap && dist[u] + e.cost < dist[e.to]) {
                dist[e.to] = dist[u] + e.cost;
                parent[e.to] = u;
                parentEdge[e.to] = i;

                if (!inQueue[e.to]) {
                    inQueue[e.to] = true;
                    q.push(e.to);
                }
            }
        }
    }

    return dist[sink] != LLONG_MAX;
}
pair<long long, long long> get(int _source = -1, int _sink = -1) {
    if (_source != -1) {
        setSource(_source);
        setSink(_sink);
    }
    long long maxFlow = 0, minCost = 0;

    while (spfa()) {
        long long flow = LLONG_MAX;

        for (int u = sink; u != source; u = parent[u]) {
            Edge& e = adj[parent[u]][parentEdge[u]];
            flow = min(flow, e.cap - e.flow);
        }

        for (int u = sink; u != source; u = parent[u]) {
            Edge& e = adj[parent[u]][parentEdge[u]];
            e.flow += flow;
            adj[u][e.rev].flow -= flow;
            minCost += flow * e.cost;
        }

        maxFlow += flow;
    }
    return { maxFlow, minCost };
}
};
```

10.3 Bipartite Matching (Hopcroft-Karp)

```
struct BiMatch { // Hopcroft-Karp O(E*sqrtV)
    int n, m; // n: left side vertices, m: right side vertices
    vector<vector<int>> adj; // adjacency list
    vector<int> pairU, pairV, dist;
    // pairU 또는 pairV를 출력해서 매칭 결과를 print 가능.

    BiMatch(int n, int m) : n(n), m(m), adj(n + 1), pairU(n + 1), pairV(m + 1),
dist(n + 1) {}

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

    bool bfs() {
        queue<int> q;
        for (int u = 1; u <= n; u++) {
            if (pairU[u] == 0) { // If u is free (not matched)
                dist[u] = 0;
                q.push(u);
            }
            else {
                dist[u] = INT_MAX;
            }
        }

        dist[0] = INT_MAX;
        while (!q.empty()) {
            int u = q.front();
            q.pop();

            if (dist[u] < dist[0]) {
                for (int v : adj[u]) {
                    if (dist[pairV[v]] == INT_MAX) { // If pairV[v] is not yet visited
                        dist[pairV[v]] = dist[u] + 1;
                        q.push(pairV[v]);
                    }
                }
            }
        }

        return dist[0] != INT_MAX; // If there's an augmenting path
    }
```

```
    bool dfs(int u) {
        if (u != 0) {
            for (int v : adj[u]) {
                if (dist[pairV[v]] == dist[u] + 1 && dfs(pairV[v])) {
                    pairV[v] = u;
                    pairU[u] = v;
                    return true;
                }
            }
            dist[u] = INT_MAX;
            return false;
        }
        return true;
    }

    int get() {
        fill(pairU.begin(), pairU.end(), 0);
        fill(pairV.begin(), pairV.end(), 0);
        int matching = 0;

        while (bfs()) {
            for (int u = 1; u <= n; u++) {
                if (pairU[u] == 0 && dfs(u)) {
                    matching++;
                }
            }
        }

        return matching;
    }

};
```

10.4 Hungarian

```
struct Hungarian { // O(N^3)
    int n;  // 문제의 크기 (n x n 행렬)
    vector<vector<long long>> cost;  // 비용 행렬
    vector<long long> u, v;  // 레이블
    vector<int> p, way;// 경로 추적
    vector<int> matchResult;   // 최종 매칭 결과 (작업자 -> 작업)

    Hungarian(int n) : n(n), cost(n, vector<long long>(n, 0)), u(n + 1, 0), v(n + 1,
0), p(n + 1, 0), way(n + 1, 0), matchResult(n, -1) {}

    void addEdge(int i, int j, long long weight) {
        cost[i][j] = weight;
    }

    long long get() {
        for (int i = 1; i <= n; i++) {
            vector<long long> minv(n + 1, LLONG_MAX);
            vector<bool> used(n + 1, false);
            int j0 = 0;
            p[0] = i;

            do {
                used[j0] = true;
                int i0 = p[j0];
                long long delta = LLONG_MAX;
                int j1;

                for (int j = 1; j <= n; j++) {
                    if (!used[j]) {
                        long long cur = cost[i0 - 1][j - 1] - u[i0] - v[j];
                        if (cur < minv[j]) {
                            minv[j] = cur;
                            way[j] = j0;
                        }
                        if (minv[j] < delta) {
                            delta = minv[j];
                            j1 = j;
                        }
                    }
                }

                for (int j = 0; j <= n; j++) {
                    if (used[j]) {
                        u[p[j]] += delta;
                        v[j] -= delta;
                    }
                    else {
                        minv[j] -= delta;
                    }
                }

                j0 = j1;
            } while (p[j0] != 0);

            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0 != 0);
        }

        // 매칭 결과 저장
        for (int j = 1; j <= n; j++) {
            matchResult[p[j] - 1] = j - 1;
        }
```

```
        return -v[0];  // 최저 비용 반환
    }

    void trace() {
        for (int i = 0; i < n; i++) {
            cout << i + 1 << " " << matchResult[i] + 1 << endl;
        }
    }

};
```

11 TwoSat & SCC

```
struct TwoSat { // for SCC and TwoSat.
    int n;
    vector<vector<int>> g, gr; // gr is the reversed graph
    vector<int> scc_id, topological_order, answer; // scc_id[v]: ID of the SCC
containing node v
    vector<bool> visited;

    TwoSat() {}

    TwoSat(int _n) { init(_n); }

    void init(int _n) {
        n = _n;
        g.assign(2 * n, vector<int>());
        gr.assign(2 * n, vector<int>());
        scc_id.resize(2 * n);
        visited.resize(2 * n);
        answer.resize(2 * n);
    }

    // Can be used conveniently for SCC tasks by using the edge function only. The
drawback is double memory usage.
    void add_edge(int u, int v) {
        g[u].push_back(v);
        gr[v].push_back(u);
    }

    // For the following three functions
    // int x, bool val: if 'val' is true, we take the variable to be x. Otherwise we
take it to be x's complement.

    // At least one of them is true
    void add_clause_or(int i, int j, bool f = true, bool g = true) {
        if (i < 0) f = !f;
        if (j < 0) g = !g;
        i = abs(i); j = abs(j);
        add_edge(i + (f ? n : 0), j + (g ? 0 : n));
        add_edge(j + (g ? n : 0), i + (f ? 0 : n));
    }

    // Only one of them is true
    void add_clause_xor(int i, int j, bool f = true, bool g = true) {
        if (i < 0) {
            add_clause_xor(-i, j, !f, g);
            return;
        }
        if (j < 0) {
            add_clause_xor(i, -j, f, !g);
            return;
        }
        add_clause_or(i, j, f, g);
        add_clause_or(i, j, !f, !g);
    }

    // Both of them have the same value
    void add_clause_xnor(int i, int j, bool f = true, bool g = true) {
        add_clause_xor(i, j, !f, g);
    }

    // Topological sort
    void dfs(int u) {
        visited[u] = true;

        for (const auto& v : g[u])
            if (!visited[v]) dfs(v);

        topological_order.push_back(u);
    }

    // Extracting strongly connected components
    void scc(int u, int id) {
        visited[u] = true;
        scc_id[u] = id;

        for (const auto& v : gr[u])
            if (!visited[v]) scc(v, id);
    }

    // Returns true if the given proposition is satisfiable and constructs a valid
assignment
    bool satisfiable() {
        fill(visited.begin(), visited.end(), false);

        for (int i = 0; i < 2 * n; i++)
            if (!visited[i]) dfs(i);

        fill(visited.begin(), visited.end(), false);
        reverse(topological_order.begin(), topological_order.end());
```

```
        int id = 0;
        for (const auto& v : topological_order)
            if (!visited[v]) scc(v, id++);

        // Constructing the answer
        for (int i = 0; i < n; i++) {
            if (scc_id[i] == scc_id[i + n]) return false;
            answer[i] = (scc_id[i] > scc_id[i + n] ? 1 : 0);
        }

        return true;
    }

};
```

12. 단절점 & 단절선

```
struct BridgeNArticulation { // 단절점, 단절선 O(V+E)
    int V; // Number of vertices
    vector<vector<int>> adj; // Adjacency list

    // Variables for finding bridges and articulation points
    vector<int> discoveryTime, low;
    vector<bool> visited;
    int time;
    unordered_set<pii, PiiHasher> bridges;
    unordered_set<int> articulationPoints;

    BridgeNArticulation(int vertices) : V(vertices), adj(vertices),
discoveryTime(vertices, -1), low(vertices, -1), visited(vertices, false), time(0) {}

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

    void dfs(int u, int parent) {
        visited[u] = true;
        discoveryTime[u] = low[u] = time++;
        int children = 0;

        for (int v : adj[u]) {
            if (!visited[v]) {
                ++children;
                dfs(v, u);

                // Update low value of u for parent function calls
                low[u] = min(low[u], low[v]);

                // Check if the edge u-v is a bridge
                if (low[v] > discoveryTime[u]) {
                    bridges.insert({ min(u, v), max(u, v) });
                }

                // Check if u is an articulation point
                if (parent != -1 && low[v] >= discoveryTime[u]) {
                    articulationPoints.insert(u);
                }
            }
            else if (v != parent) {
                // Update low value of u for back edge
                low[u] = min(low[u], discoveryTime[v]);
            }
        }

        // Special case for root of DFS tree
        if (parent == -1 && children > 1) {
            articulationPoints.insert(u);
        }
    }

    void get() {
        for (int i = 0; i < V; ++i) {
            if (!visited[i]) {
                dfs(i, -1);
            }
        }
    }
};
```

13. Mo's

```
ll ans[100000];
vector<vector<int>> query;
int sqrtN;
bool cmp(vector<int>& q1, vector<int>& q2) {
    if (q1[0] / sqrtN == q2[0] / sqrtN) {
        return q1[1] < q2[1];
    }
    return q1[0] < q2[0];
}

void solve(int tc) {
    int N, M;
    sqrtN = sqrt(N);
    query.resize(M);
    sort(query.begin(), query.end(), cmp);

    int s = 0;int e = 0;
    ll ret = 0;
    for (int i = 0; i < M; i++) {
```

```
        int l = query[i][0];
        int r = query[i][1] + 1;
        while (l < s) {
            s -= 1;
            //modify ret
        }
        while (e < r) {
            //modify ret
            e += 1;
        }
        while (s < l) {
            //modify ret
            s += 1;
        }
        while (r < e) {
            e -= 1;
            //modify ret
        }
        ans[query[i][2]] = ret;
    }
}
```

14. Centroid Decomp

```
vector<pll> E[100001];
int sz[100001];
bool visited[100001];

int getsz(int x, int p = -1) {
    sz[x] = 1;
    for (pll& e : E[x]) {
        int nx = e.first;
        if (nx == p || visited[nx]) continue;
        sz[x] += getsz(nx, x);
    }
    return sz[x];
}

int getcent(int x, int p, int s) {
    for (pll& e : E[x]) {
        int nx = e.first;
        if (nx == p || visited[nx]) continue;
        if (sz[nx] * 2 > s) return getcent(nx, x, s);
    }
    return x;
}

int parent[100001];
// unordered_map<int,ll> dist[100001]

void dnc(int x, int prvcent = -1) {
    int cent = getcent(x, -1, getsz(x));

    parent[cent] = prvcent;
    visited[cent] = true;

    queue<pii> q;
    for (pll& e : E[cent]) {
        int nx = e.first;
        if (visited[nx]) continue;
        q.push({ nx,cent }); // cur,prv
        // update dist, branch or sth
    }
    while (!q.empty()) {
        int cur = q.front().first;
        int prv = q.front().second;
        q.pop();
        for (pll& e : E[cur]) {
            int nx = e.first;
            if (nx == prv || visited[nx]) continue;
            q.push({ nx,cur });
            // update dist, branch or sth
        }
    }

    for (pll& e : E[cent]) {
        int nx = e.first;
        if (visited[nx]) continue;
        dnc(nx, cent);
    }

}
```