

# Contents

1 Basic	1
1.1 default	1
2 Matching & Flow	1
3 Graph	1
3.1 2-SAT	1
4 Data Structure	1
4.1 Blackmagic	1
4.2 Centroid Decomposition	1
5 Math	1
5.1 CRT	1
5.2 Factorize	2
5.3 NTT	2
5.4 Lucas	2
5.5 FloorSum	3
6 Geometry	3
6.1 Convex Hull	3
6.2 Dynamic Convex Hull	3
6.3 Half Plane Intersection	3
6.4 Minimal Enclosing Circle	3
6.5 Minkowski	4
7 Stringology	4
7.1 Z-algorithm	4
7.2 Manacher	4
7.3 SuffixArray	4
7.4 PalindromicTree	4
8 Misc	5

## 1 Basic

### 1.1 default

```
#include <bits/stdc++.h>
using namespace std;
#ifdef LOCAL
template<class ...T> void dbg(T ...x) { char e{}; ((
    cerr << e << x, e = ' ', ...); }
template<class T> void org(T l, T r) { while (l != r)
    cerr << ' ' << *l++; cerr << '\n'; }
#define debug(x...) dbg("(", #x, ") =", x, '\n')
#define orange(x...) dbg("[", #x, "] =", org(x))
#else
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#define debug(...) ((void)0)
#define orange(...) ((void)0)
#endif
#define ff first
#define ss second
#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()
template<class T> bool chmin(T &a, T b) { return b < a
    and (a = b, true); }
template<class T> bool chmax(T &a, T b) { return a < b
    and (a = b, true); }
```

## 2 Matching & Flow

## 3 Graph

### 3.1 2-SAT

```
struct TwoSAT {
    vector<vector<int>>> G;
    int n;
    TwoSAT(int _n) : n(_n), G(_n * 2) {}
    int ne(int x) { return x < n ? x + n : x - n; }
    void add_edge(int u, int v) { // u or v
        G[ne(u)].push_back(v);
        G[ne(v)].push_back(u);
    }
    vector<int> solve() {
        vector<int> ans(n * 2, -1), id(n * 2), stk, \
            low(n * 2), dfn(n * 2), vis(n * 2);
        int _t = 0, scc_cnt = 0;
        function<void(int)> dfs = [&](int u) {
            dfn[u] = low[u] = _t++;
            stk.push_back(u);
            vis[u] = 1;
            for (int v : G[u]) {
                if (!vis[v])
```

```
                dfs(v), chmin(low[u], low[v]);
            else if (vis[v] == 1)
                chmin(low[u], dfn[v]);
        }
        if (dfn[u] == low[u]) {
            for (int x = -1; x != u; ) {
                x = stk.back(); stk.pop_back();
                vis[x] = 2, id[x] = scc_cnt;
                if (ans[x] == -1) {
                    ans[x] = 1;
                    ans[ne(x)] = 0;
                }
            }
            scc_cnt++;
        }
    };
    for (int i = 0; i < n + n; i++)
        if (!vis[i]) dfs(i);
    for (int i = 0; i < n; i++)
        if (id[i] == id[ne(i)])
            return {};
    ans.resize(n);
    return ans;
}
```

## 4 Data Structure

### 4.1 Blackmagic

```
#include <bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
template<class T>
using BST = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
gnu_pbds::priority_queue<node, decltype(cmp),
    pairing_heap_tag> pq(cmp);
gp_hash_table<int, gnu_pbds::priority_queue<node>::
    point_iterator> pqPos;
bst.insert((x << 20) + i);
bst.erase(bst.lower_bound(x << 20));
bst.order_of_key(x << 20) + 1;
*bst.find_by_order(x - 1) >> 20;
*--bst.lower_bound(x << 20) >> 20;
*bst.upper_bound((x + 1) << 20) >> 20;
```

### 4.2 Centroid Decomposition

```
struct CenDec {
    vector<vector<pair<int, int>>> anc;
    vector<int> Mdis;
    CenDec(const vector<vector<int>>> &G) : anc(G.size()),
        Mdis(G.size(), INF) {
        const int n = G.size();
        vector<int> siz(n);
        vector<bool> vis(n);
        function<int(int, int)> getsiz = [&](int u, int f)
            {
                siz[u] = 1;
                for (int v : G[u]) if (v != f and !vis[v])
                    siz[u] += getsiz(v, u);
                return siz[u];
            };
        function<int(int, int, int)> find = [&](int u, int
            f, int s) {
                for (int v : G[u]) if (v != f and !vis[v])
                    if (siz[v] * 2 >= s) return find(v, u, s);
                return u;
            };
        function<void(int, int, int, int)> caldis = [&](int
            u, int f, int a, int d) {
                anc[u].emplace_back(a, d);
                for (int v : G[u]) if (v != f and !vis[v])
                    caldis(v, u, a, d + 1);
            };
        function<void(int)> build = [&](int u) {
            u = find(u, u, getsiz(u, u));
            vis[u] = 1;
            for (int v : G[u]) if (!vis[v]) {
```

```

        caldis(v, u, u, 1);
        build(v);
    }
    vis[u] = 0;
};
build(0);
}
void add(int p) {
    Mdis[p] = 0;
    for (auto [v, d] : anc[p])
        chmin(Mdis[v], d);
}
int que(int p) {
    int r = Mdis[p];
    for (auto [v, d] : anc[p])
        chmin(r, Mdis[v] + d);
    return r;
}
};
};

```

## 5 Math

### 5.1 CRT

```

i64 CRT(vector<pair<i64, i64>> E) {
    i128 R = 0, M = 1;
    for (auto [r, m] : E) {
        i128 d = r - R, g = gcd<i64>(M, m);
        if (d % g != 0) return -1;
        i128 x = exgcd(M / g, m / g).ff * d / g;
        R += M * x;
        M = M * m / g;
        R = (R % M + M) % M;
    }
    return R;
}

```

### 5.2 Factorize

```

struct Factorize {
    i64 fmul(i64 a, i64 b, i64 p) {
        return (i128)a * b % p;
    }
    i64 fpow(i64 a, i64 b, i64 p) {
        i64 res = 1;
        for (; b >= 1, a = fmul(a, a, p))
            if (b & 1) res = fmul(res, a, p);
        return res;
    }
    bool Check(i64 a, i64 u, i64 n, int t) {
        a = fpow(a, u, n);
        if (a == 0 or a == 1 or a == n - 1) return true;
        for (int i = 0; i < t; i++) {
            a = fmul(a, a, n);
            if (a == 1) return false;
            if (a == n - 1) return true;
        }
        return false;
    };
    bool IsPrime(i64 n) {
        constexpr array<i64, 7> kChk{2, 235, 9375, 28178,
            450775, 9780504, 1795265022};
        // for int: {2, 7, 61}
        if (n < 2) return false;
        if (n % 2 == 0) return n == 2;
        i64 u = n - 1;
        int t = 0;
        while (u % 2 == 0) u >>= 1, t++;
        for (auto v : kChk) if (!Check(v, u, n, t)) return false;
        return true;
    }
    i64 PollardRho(i64 n) {
        if (n % 2 == 0) return 2;
        i64 x = 2, y = 2, d = 1, p = 1;
        auto f = [](i64 x, i64 n, i64 p) -> i64 {
            return ((i128)x * x % n + p) % n;
        };
        while (true) {
            x = f(x, n, p);
            y = f(f(y, n, p), n, p);
            d = __gcd(abs(x - y), n);
            if (d != n and d != 1) return d;
        }
    }
};

```

```

        if (d == n) ++p;
    }
};

```

### 5.3 NTT

```

// 17 -> 3
// 97 -> 5
// 193 -> 5
// 998244353 -> 3
// 985661441 -> 3
constexpr i64 cpow(i64 a, i64 b, i64 m) {
    i64 ret = 1;
    for (; b >= 1, a = a * a % m)
        if (b & 1) ret = ret * a % m;
    return ret;
};
template<i64 M, i64 G>
struct NTT {
    static constexpr i64 iG = cpow(G, M - 2, M);
    void operator()(vector<i64> &v, bool inv) {
        int n = v.size();
        for (int i = 0, j = 0; i < n; i++) {
            if (i < j) swap(v[i], v[j]);
            for (int k = n / 2; (j ^= k) < k; k /= 2);
        }
        for (int mid = 1; mid < n; mid *= 2) {
            i64 w = cpow((inv ? iG : G), (M - 1) / (mid + mid), M);
            for (int i = 0; i < n; i += mid * 2) {
                i64 now = 1;
                for (int j = i; j < i + mid; j++, now = now * w % M) {
                    i64 x = v[j], y = v[j + mid];
                    v[j] = (x + y * now) % M;
                    v[j + mid] = (x - y * now) % M;
                }
            }
        }
        if (inv) {
            i64 in = cpow(n, M - 2, M);
            for (int i = 0; i < n; i++) v[i] = v[i] * in % M;
        }
    };
};
template<i64 M, i64 G>
vector<i64> convolution(vector<i64> f, vector<i64> g) {
    NTT<M, G> ntt;
    int sum = f.size() + g.size() - 1;
    int len = bit_ceil((u64)sum);
    f.resize(len); g.resize(len);
    ntt(f, 0), ntt(g, 0);
    for (int i = 0; i < len; i++) (f[i] *= g[i]) %= M;
    ntt(f, 1);
    f.resize(sum);
    for (int i = 0; i < sum; i++) if (f[i] < 0) f[i] += M;
    return f;
};
vector<i64> convolution_ll(const vector<i64> &f, const vector<i64> &g) {
    constexpr i64 M1 = 998244353, G1 = 3;
    constexpr i64 M2 = 985661441, G2 = 3;
    constexpr i64 M1M2 = M1 * M2;
    constexpr i64 M1m1 = M2 * cpow(M2, M1 - 2, M1);
    constexpr i64 M2m2 = M1 * cpow(M1, M2 - 2, M2);
    auto c1 = convolution<M1, G1>(f, g);
    auto c2 = convolution<M2, G2>(f, g);
    for (int i = 0; i < c1.size(); i++) {
        c1[i] = ((i128)c1[i] * M1m1 + (i128)c2[i] * M2m2) % M1M2;
    }
    return c1;
};

```

### 5.4 Lucas

```

i64 Lucas(i64 N, i64 M, i64 D) { // C(N, M) mod D
    auto Factor = [&](i64 x) -> vector<pair<i64, i64>> {
        vector<pair<i64, i64>> r;
        for (i64 i = 2; x > 1; i++)
            if (x % i == 0) {
                i64 c = 0;
            }
    };
}

```

```

    while (x % i == 0) x /= i, c++;
    r.emplace_back(i, c);
}
return r;
};
auto Pow = [&](i64 a, i64 b, i64 m) -> i64 {
    i64 r = 1;
    for (; b >= 1; a = a * a % m)
        if (b & 1) r = r * a % m;
    return r;
};
vector<pair<i64, i64>> E;
for (auto [p, q] : Factor(D)) {
    const i64 mod = Pow(p, q, 1 << 30);
    auto CountFact = [&](i64 x) -> i64 {
        i64 c = 0;
        while (x) c += (x /= p);
        return c;
    };
    auto CountBino = [&](i64 x, i64 y) { return
    CountFact(x) - CountFact(y) - CountFact(x - y); };
    auto Inv = [&](i64 x) -> i64 { return (exgcd(x, mod
    ).ff % mod + mod) % mod; };
    vector<i64> pre(mod + 1);
    pre[0] = pre[1] = 1;
    for (i64 i = 2; i <= mod; i++) pre[i] = (i % p == 0
    ? 1 : i) * pre[i - 1] % mod;
    function<i64(i64)> FactMod = [&](i64 n) -> i64 {
        if (n == 0) return 1;
        return FactMod(n / p) * Pow(pre[mod], n / mod,
        mod) % mod * pre[n % mod] % mod;
    };
    auto BinoMod = [&](i64 x, i64 y) -> i64 {
        return FactMod(x) * Inv(FactMod(y)) % mod * Inv(
        FactMod(x - y)) % mod;
    };
    i64 r = BinoMod(N, M) * Pow(p, CountBino(N, M), mod
    ) % mod;
    E.emplace_back(r, mod);
};
return CRT(E);
}

```

## 5.5 FloorSum

```

// sigma 0 ~ n-1: (a * i + b) / m
i64 floor_sum(i64 n, i64 m, i64 a, i64 b) {
    u64 ans = 0;
    if (a < 0) {
        u64 a2 = (a % m + m) % m;
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        u64 b2 = (b % m + m) % m;
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m);
            a %= m;
        }
        if (b >= m) {
            ans += n * (b / m);
            b %= m;
        }
        u64 y_max = a * n + b;
        if (y_max < m) break;
        n = y_max / m;
        b = y_max % m;
        swap(m, a);
    }
    return ans;
}

```

## 6 Geometry

### 6.1 Convex Hull

```

vector<Pt> Hull(vector<Pt> P) {
    sort(all(P));
    P.erase(unique(all(P)), P.end());
}

```

```

P.insert(P.end(), rall(P));
vector<Pt> stk;
for (auto p : P) {
    while (stk.size() >= 2 and \
        cro(++stk.rbegin(), stk.back(), p) <= 0 and \
        (++stk.rbegin() < stk.back()) == (stk.back() <
        p)) {
        stk.pop_back();
    }
    stk.push_back(p);
}
stk.pop_back();
return stk;
}

```

### 6.2 Dynamic Convex Hull

```

template<class T, class Comp = less<T>>
struct DynamicHull {
    set<T, Comp> H;
    DynamicHull() {}
    void insert(T p) {
        if (inside(p)) return;
        auto it = H.insert(p).ff;
        while (it != H.begin() and prev(it) != H.begin() \
            and cross(*prev(it), 2), *prev(it), *it) <= 0) {
            it = H.erase(--it);
        }
        while (it != --H.end() and next(it) != --H.end() \
            and cross(*it, *next(it), *next(it), 2) <= 0) {
            it = --H.erase(++it);
        }
    }
    bool inside(T p) {
        auto it = H.lower_bound(p);
        if (it == H.end()) return false;
        if (it == H.begin()) return p == *it;
        return cross(*prev(it), p, *it) <= 0;
    }
};

```

### 6.3 Half Plane Intersection

```

vector<Pt> HPI(vector<Line> P) {
    const int n = P.size();
    sort(all(P), [&](Line L, Line R) -> bool {
        Pt u = L.b - L.a, v = R.b - R.a;
        bool f = Pt(sig(u.ff), sig(u.ss)) < Pt{};
        bool g = Pt(sig(v.ff), sig(v.ss)) < Pt{};
        if (f != g) return f < g;
        return (sig(u ^ v) ? sig(u ^ v) : sig(cro(L.a, R.a,
        R.b))) > 0;
    });
    auto Same = [&](Line L, Line R) {
        Pt u = L.b - L.a, v = R.b - R.a;
        return sig(u ^ v) == 0 and sig(u * v) == 1;
    };
    deque<Pt> inter;
    deque<Line> seg;
    for (int i = 0; i < n; i++) if (i == 0 or !Same(P[i -
    1], P[i])) {
        while (seg.size() >= 2 and sig(cro(inter.back(), P[
        i].b, P[i].a)) == 1) {
            seg.pop_back(), inter.pop_back();
        }
        while (seg.size() >= 2 and sig(cro(inter[0], P[i].b
        , P[i].a)) == 1) {
            seg.pop_front(), inter.pop_front();
        }
        if (!seg.empty()) inter.push_back(Inter(seg.back(),
        P[i]));
        seg.push_back(P[i]);
    }
    while (seg.size() >= 2 and sig(cro(inter.back(), seg
    [0].b, seg[0].a)) == 1) {
        seg.pop_back(), inter.pop_back();
    }
    inter.push_back(Inter(seg[0], seg.back()));
    return vector<Pt>(all(inter));
}

```

### 6.4 Minimal Enclosing Circle

```

using circle = pair<Pt, double>;
struct MES {
    MES() {}
    bool inside(const circle &c, Pt p) {
        return abs(p - c.ff) <= c.ss + eps;
    };
    circle get_cir(Pt a, Pt b) {
        return circle((a + b) / 2., abs(a - b) / 2.);
    };
    circle get_cir(Pt a, Pt b, Pt c) {
        Pt p = (b - a) / 2.;
        p = Pt(-p.ss, p.ff);
        double t = ((c - a) * (c - b)) / (2 * (p * (c - a)));
        p = ((a + b) / 2.) + (p * t);
        return circle(p, abs(p - a));
    };
    circle get_mes(vector<Pt> P) {
        if (P.empty()) return circle(Pt(0, 0), 0);
        mt19937 rng(random_device{}());
        shuffle(all(P), rng);
        circle C(P[0], 0);
        for (int i = 1; i < P.size(); i++) {
            if (inside(C, P[i])) continue;
            C = get_cir(P[i], P[0]);
            for (int j = 1; j < i; j++) {
                if (inside(C, P[j])) continue;
                C = get_cir(P[i], P[j]);
                for (int k = 0; k < j; k++) {
                    if (inside(C, P[k])) continue;
                    C = get_cir(P[i], P[j], P[k]);
                }
            }
        }
        return C;
    };
};

```

## 6.5 Minkowski

```

vector<Pt> Minkowski(vector<Pt> P, vector<Pt> Q) {
    auto reorder = [&](auto &R) -> void {
        auto cmp = [&](Pt a, Pt b) -> bool {
            return Pt(a.ss, a.ff) < Pt(b.ss, b.ff);
        };
        rotate(R.begin(), min_element(all(R), cmp), R.end());
        R.push_back(R[0]), R.push_back(R[1]);
    };
    const int n = P.size(), m = Q.size();
    reorder(P), reorder(Q);
    vector<Pt> R;
    for (int i = 0, j = 0, s; i < n or j < m; ) {
        R.push_back(P[i] + Q[j]);
        s = sig((P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]));
        i += (s >= 0), j += (s <= 0);
    }
    return R;
}

```

# 7 Stringology

## 7.1 Z-algorithm

```

vector<int> zalgo(string s) {
    if (s.empty()) return {};
    int len = s.size();
    vector<int> z(len);
    z[0] = len;
    for (int i = 1, l = 1, r = 1; i < len; i++) {
        z[i] = i < r ? min(z[i - l], r - i) : 0;
        while (i + z[i] < len and s[i + z[i]] == s[z[i]]) z[i]++;
        if (i + z[i] > r) l = i, r = i + z[i];
    }
    return z;
}

```

## 7.2 Manacher

```

vector<int> manacher(const string &s) {
    string p = "@#";
    for (char c : s) p += c + '#';
    p += '$';
}

```

```

vector<int> dp(p.size());
int mid = 0, r = 1;
for (int i = 1; i < p.size() - 1; i++) {
    auto &k = dp[i];
    k = i < mid + r ? min(dp[mid * 2 - i], mid + r - i) : 0;
    while (p[i + k + 1] == p[i - k - 1]) k++;
    if (i + k > mid + r) mid = i, r = k;
}
return vector<int>(dp.begin() + 2, dp.end() - 2);
}

```

## 7.3 SuffixArray

```

namespace sfx {
#define fup(a, b) for (int i = a; i < b; i++)
#define fdn(a, b) for (int i = b - 1; i >= a; i--)
constexpr int N = 5e5 + 5;
bool _t[N * 2];
int H[N], RA[N], x[N], _p[N];
int SA[N * 2], _s[N * 2], _c[N * 2], _q[N * 2];
void pre(int *sa, int *c, int n, int z) {
    fill_n(sa, n, 0), copy_n(c, z, x);
}
void induce(int *sa, int *c, int *s, bool *t, int n, int z) {
    copy_n(c, z - 1, x + 1);
    fup(0, n) if (sa[i] and !t[sa[i] - 1])
        sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
    copy_n(c, z, x);
    fdn(0, n) if (sa[i] and t[sa[i] - 1])
        sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
}
void sais(int *s, int *sa, int *p, int *q, bool *t, int *c, int n, int z) {
    bool uniq = t[n - 1] = true;
    int nn = 0, nmzx = -1, *nsa = sa + n, *ns = s + n, last = -1;
    fill_n(c, z, 0);
    fup(0, n) uniq &= ++c[s[i]] < 2;
    partial_sum(c, c + z, c);
    if (uniq) { fup(0, n) sa[--c[s[i]]] = i; return; }
    fdn(0, n - 1)
        t[i] = (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
    pre(sa, c, n, z);
    fup(1, n) if (t[i] and !t[i - 1])
        sa[--x[s[i]]] = p[q[i] = nn++] = i;
    induce(sa, c, s, t, n, z);
    fup(0, n) if (sa[i] and t[sa[i]] and !t[sa[i] - 1]) {
        bool neq = last < 0 or !equal(s + sa[i], s + p[q[sa[i]] + 1], s + last);
        ns[q[last = sa[i]]] = nmzx += neq;
    }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmzx + 1);
    pre(sa, c, n, z);
    fdn(0, nn) sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
    induce(sa, c, s, t, n, z);
}
vector<int> build(vector<int> s, int n) {
    copy_n(begin(s), n, _s), _s[n] = 0;
    sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
    vector<int> sa(n);
    fup(0, n) sa[i] = SA[i + 1];
    return sa;
}
vector<int> lcp_array(vector<int> &s, vector<int> &sa) {
    int n = int(s.size());
    vector<int> rnk(n);
    fup(0, n) rnk[sa[i]] = i;
    vector<int> lcp(n - 1);
    int h = 0;
    fup(0, n) {
        if (h > 0) h--;
        if (rnk[i] == 0) continue;
        int j = sa[rnk[i] - 1];
        for (; j + h < n and i + h < n; h++)
            if (s[j + h] != s[i + h]) break;
        lcp[rnk[i] - 1] = h;
    }
}

```

```

    return lcp;
}
}

```

## 7.4 PalindromicTree

```

struct PAM {
    struct Node {
        int fail, len, dep;
        array<int, 26> ch;
        Node(int _len) : len{_len}, fail{}, ch{}, dep{} {};
    };
    vector<Node> g;
    vector<int> id;
    int odd, even, lst;
    string S;
    int new_node(int len) {
        g.emplace_back(len);
        return g.size() - 1;
    }
    PAM() : odd(new_node(-1)), even(new_node(0)) {
        lst = g[even].fail = odd;
    }
    int up(int p) {
        while (S.rbegin()[g[p].len + 1] != S.back())
            p = g[p].fail;
        return p;
    }
    int add(char c) {
        S += c;
        lst = up(lst);
        c -= 'a';
        if (!g[lst].ch[c]) g[lst].ch[c] = new_node(g[lst].len + 2);
        int p = g[lst].ch[c];
        g[p].fail = (lst == odd ? even : g[up(g[lst].fail).ch[c]]);
        lst = p;
        g[lst].dep = g[g[lst].fail].dep + 1;
        id.push_back(lst);
        return lst;
    }
    void del() {
        S.pop_back();
        id.pop_back();
        lst = id.empty() ? odd : id.back();
    }
};

```

## 8 Misc