

<b>Contents</b>	<b>8</b>	<b>Extra</b>	<b>17</b>
<b>1 All Macros</b>	<b>1</b>	8.1 Stress Tester . . . . .	17
<b>2 Data Structure</b>	<b>1</b>	8.2 Sublime Build . . . . .	17
2.1 Segment Tree . . . . .	1	8.3 vimrc . . . . .	17
2.2 Persistent Segment Tree . . .	1	<b>9 Equations and Formulas</b>	<b>19</b>
2.3 Lazy Persistent Segment Tree	2	9.1 Catalan Numbers . . . . .	19
2.4 Implicit Segment Tree . . . .	2	9.2 Stirling Numbers First Kind .	19
2.5 DSU With Rollbacks . . . . .	2	9.3 Stirling Numbers Second Kind	19
2.6 BIT-2D . . . . .	3	9.4 Other Combinatorial Identities	19
2.7 Merge Sort Tree . . . . .	3	9.5 Different Math Formulas . . .	19
2.8 MO with Update . . . . .	3	9.6 GCD and LCM . . . . .	19
2.9 SparseTable (Rectangle Query) . . . . .	3		
2.10 Sparse Table . . . . .	4		
<b>3 DP</b>	<b>4</b>		
3.1 Convex Hull Trick . . . . .	4		
3.2 Dynamic CHT . . . . .	4		
3.3 Li Chao Tree . . . . .	5		
<b>4 Geometry</b>	<b>5</b>		
4.1 Point . . . . .	5		
4.2 Linear . . . . .	5		
4.3 Circular . . . . .	6		
4.4 Convex . . . . .	7		
4.5 Polygon . . . . .	9		
<b>5 Graph</b>	<b>9</b>		
5.1 LCA, ETT, VT . . . . .	9		
5.2 SCC . . . . .	10		
5.3 Euler Tour on Edge . . . . .	10		
5.4 LCA In $O(1)$ . . . . .	10		
5.5 HLD . . . . .	11		
5.6 Centroid Tree . . . . .	11		
5.7 Dinic Max Flow . . . . .	11		
5.8 Min Cost Max Flow . . . . .	12		
5.9 Bridge Tree . . . . .	12		
5.10 Tree Isomorphism . . . . .	13		
5.11 Grundy . . . . .	13		
<b>6 Math</b>	<b>13</b>		
6.1 Linear Sieve . . . . .	13		
6.2 Pollard Rho . . . . .	13		
6.3 Extended Euclidean . . . . .	13		
6.4 Chinese Remainder Theorem	14		
6.5 Mobius Function . . . . .	14		
6.6 FFT . . . . .	14		
6.7 NTT . . . . .	14		
<b>7 String</b>	<b>15</b>		
7.1 Aho Corasick . . . . .	15		
7.2 Double hash . . . . .	15		
7.3 KMP . . . . .	16		
7.4 Manacher's . . . . .	16		
7.5 String Match FFT . . . . .	16		
7.6 Suffix Array . . . . .	16		
7.7 Trie . . . . .	17		
7.8 Z Algo . . . . .	17		

## 1 All Macros

```

#pragma GCC optimize("Ofast")
#pragma GCC optimization ("O3")
#pragma comment(linker, "/stack
:2000000000")
#pragma GCC optimize("unroll-loops")
#pragma GCC target("sse,sse2,sse3,
ssse3,sse4,popcnt,abm,mmx,avx,tune=
native")

#include <ext/pb_ds/assoc_container.hpp>
>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
//find_by_order(k) --> returns
iterator to the kth largest
element counting from 0
//order_of_key(val) --> returns the
number of items in a set that
are strictly smaller than our
item

template <typename DT>
using ordered_set = tree<DT, null_type
, less<DT>, rb_tree_tag,
tree_order_statistics_node_update>;

/*--- DEBUG TEMPLATE STARTS HERE ---*/
#ifdef LOCAL
void show(int x) {cerr << x;}
void show(long long x) {cerr << x;}
void show(double x) {cerr << x;}
void show(char x) {cerr << '\'' << x <<
'\'';}
void show(const string &x) {cerr << '\''
' << x << '\'';}
void show(bool x) {cerr << (x ? "true"
: "false");}

template<typename T, typename V>
void show(pair<T, V> x) { cerr << '{';
show(x.first); cerr << ", "; show(x
.second); cerr << '}';}
template<typename T>
void show(T x) {int f = 0; cerr << "{";
for (auto &i: x) cerr << (f++ ? ",
" : ""), show(i); cerr << "}";}

void debug_out(string s) {
cerr << '\n';
}
template <typename T, typename... V>
void debug_out(string s, T t, V... v) {
s.erase(remove(s.begin(), s.end(), '
'), s.end());
cerr << " "; // 8 spaces
cerr << s.substr(0, s.find(','));
s = s.substr(s.find(',') + 1);
cerr << " = ";
show(t);
cerr << endl;
if(sizeof...(v)) debug_out(s, v...);
}

#define dbg(x...) cerr << "LINE: " <<
__LINE__ << endl; debug_out(#x, x);
cerr << endl;
#else
#define dbg(x...)

```

```

#endif

```

```

const int RANDOM = chrono::
high_resolution_clock::now().
time_since_epoch().count();
unsigned hash_f(unsigned x) {
x = ((x >> 16) ^ x) * 0x45d9f3b;
x = ((x >> 16) ^ x) * 0x45d9f3b;
return x = (x >> 16) ^ x;
}

unsigned hash_combine(unsigned a,
unsigned b) { return a * 31 + b; }
struct chash {
int operator()(int x) const { return
hash_f(x); }
};
typedef gp_hash_table<int, int, chash>
gp;
gp table;

mt19937_64 rng(atoi(argv[1]));

long long random(long long l, long long
r) {
uniform_int_distribution<long long>
dist(l, r);
return dist(rng);
}

```

## 2 Data Structure

### 2.1 Segment Tree

```

namespace segtree {
const int N = 1000006;

using DT = long long;
using LT = long long;
constexpr DT I = 0;
constexpr LT None = 0;

DT val[N<<2];
LT lz[N<<2];
int L, R;
void apply(int u, const LT &U, int l,
int r) {
if (U != None) val[u] += (r - l + 1)
* U;
lz[u] += U;
}
DT merge(const DT &a, const DT &b,
int l, int r) {
return a + b;
}
/* -- Do Not Touch Anything Below
This -- */
void push(int l, int r, int u) {
if(l == r) return;
apply(u << 1, lz[u], l, (l + r) >>
1);
apply(u << 1 | 1, lz[u], (l + r + 2)
>> 1, r);
lz[u] = None;
}
void build(int l, int r, vector<DT>
const &v, int u = 1) {
lz[u] = None;
if(l == r) {

```

```

val[u] = v[l];
return;
}
int m = (l + r) >> 1, lft = u << 1,
ryt = lft | 1;
build(l, m, v, lft);
build(m + 1, r, v, ryt);
val[u] = merge(val[lft], val[ryt], l
, r);
}
void update(int ql, int qr, LT uval,
int l = L, int r = R, int u = 1)
{
if (qr < l or ql > r) return;
if(ql <= l and r <= qr) {
apply(u, uval, l, r);
return;
}
push(l, r, u);
int m = (l + r) >> 1, lft = u << 1,
ryt = lft | 1;
update(ql, qr, uval, l, m, lft);
update(ql, qr, uval, m + 1, r, ryt);
val[u] = merge(val[lft], val[ryt], l
, r);
}
DT query(int ql, int qr, int l = L,
int r = R, int u = 1) {
if (qr < l or ql > r) return I;
if (ql <= l and r <= qr) return val[
u];
push(l, r, u);
int m = (l + r) >> 1, lft = u << 1,
ryt = lft | 1;
DT ans1 = query(ql, qr, l, m, lft);
DT ansr = query(ql, qr, m + 1, r,
ryt);
return merge(ans1, ansr, l, r);
}
void init(int _L, int _R, vector<DT>
v) {
L = _L, R = _R;
build(L, R, v);
}
};

```

### 2.2 Persistent Segment Tree

```

struct Node {
int l = 0, r = 0, val = 0;
} tr[20 * N];
int ptr = 0;
int build(int st, int en) {
int u = ++ptr;
if (st == en) return u;
int mid = (st + en) / 2;
auto& [l, r, val] = tr[u];
l = build(st, mid);
r = build(mid + 1, en);
val = tr[l].val + tr[r].val;
return u;
}
int update(int pre, int st, int en, int
idx, int v) {
int u = ++ptr;
tr[u] = tr[pre];
if (st == en) {
tr[u].val += v;
return u;
}

```



```

}
};

2.6 BIT-2D

const int N = 1008;
int bit[N][N], n, m;
int a[N][N], q;
void update(int x, int y, int val) {
    for (; x < N; x += -x & x)
        for (int j = y; j < N; j += -j & j)
            bit[x][j] += val;
}
int get(int x, int y) {
    int ans = 0;
    for (; x; x -= x & -x)
        for (int j = y; j; j -= j & -j) ans
            += bit[x][j];
    return ans;
}
int get(int x1, int y1, int x2, int y2)
{
    return get(x2, y2) - get(x1 - 1, y2)
        - get(x2, y1 - 1) + get(x1 - 1,
            y1 - 1);
}
}

```

## 2.7 Merge Sort Tree

```

vector<LL> Tree[4 * MAXN];
LL arr[MAXN];

vector<LL> merge(vector<LL> v1, vector<
    LL> v2) {
    LL i = 0, j = 0;
    vector<LL> ret;

    while (i < v1.size() || j < v2.size()) {
        if (i == v1.size()) {
            ret.push_back(v2[j]);
            j++;
        } else if (j == v2.size()) {
            ret.push_back(v1[i]);
            i++;
        } else {
            if (v1[i] < v2[j]) {
                ret.push_back(v1[i]);
                i++;
            } else {
                ret.push_back(v2[j]);
                j++;
            }
        }
    }

    return ret;
}

void Build(LL node, LL bg, LL ed) {
    if (bg == ed) {
        Tree[node].push_back(arr[bg]);
        return;
    }

    LL leftNode = 2 * node, rightNode = 2
        * node + 1;
    LL mid = (bg + ed) / 2;

    Build(leftNode, bg, mid);
    Build(rightNode, mid + 1, ed);
}

```

```

Tree[node] = merge(Tree[leftNode],
    Tree[rightNode]);
}

LL query(LL node, LL bg, LL ed, LL l,
    LL r, LL k) {
    if (ed < l || bg > r) return 0;

    if (l <= bg && ed <= r)
        return upper_bound(Tree[node].begin
            (), Tree[node].end(), k) -
            Tree[node].begin();

    LL leftNode = 2 * node, rightNode = 2
        * node + 1;
    LL mid = (bg + ed) / 2;

    return query(leftNode, bg, mid, l, r,
        k) +
        query(rightNode, mid + 1, ed, l
            , r, k);
}

```

## 2.8 MO with Update

```

const int N = 1e5 + 5, sz = 2700, bs =
    25;
int arr[N], freq[2 * N], cnt[2 * N], id
    [N], ans[N];
struct query {
    int l, r, t, L, R;
    query(int l = 1, int r = 0, int t =
        1, int id = -1)
        : l(l), r(r), t(t), L(l / sz), R(r
            / sz) {}
    bool operator<(const query &rhs)
        const {
            return (L < rhs.L) or (L == rhs.L
                and R < rhs.R) or
                (L == rhs.L and R == rhs.R
                    and t < rhs.t);
        }
} Q[N];
struct update {
    int idx, val, last;
} Up[N];
int qi = 0, ui = 0;
int l = 1, r = 0, t = 0;

void add(int idx) {
    --cnt[freq[arr[idx]]];
    freq[arr[idx]]++;
    cnt[freq[arr[idx]]]++;
}

void remove(int idx) {
    --cnt[freq[arr[idx]]];
    freq[arr[idx]]--;
    cnt[freq[arr[idx]]]++;
}

void apply(int t) {
    const bool f = l <= Up[t].idx and Up[
        t].idx <= r;
    if (f) remove(Up[t].idx);
    arr[Up[t].idx] = Up[t].val;
    if (f) add(Up[t].idx);
}

void undo(int t) {
    const bool f = l <= Up[t].idx and Up[
        t].idx <= r;
}

```

```

if (f) remove(Up[t].idx);
arr[Up[t].idx] = Up[t].last;
if (f) add(Up[t].idx);
}

int mex() {
    for (int i = 1; i <= N; i++)
        if (!cnt[i]) return i;
    assert(0);
}

int main() {
    int n, q;
    cin >> n >> q;
    int counter = 0;
    map<int, int> M;
    for (int i = 1; i <= n; i++) {
        cin >> arr[i];
        if (!M[arr[i]]) M[arr[i]] = ++
            counter;
        arr[i] = M[arr[i]];
    }
    iota(id, id + N, 0);
    while (q--) {
        int tp, x, y;
        cin >> tp >> x >> y;
        if (tp == 1)
            Q[++qi] = query(x, y, ui);
        else {
            if (!M[y]) M[y] = ++counter;
            y = M[y];
            Up[++ui] = {x, y, arr[x]};
            arr[x] = y;
        }
    }
    t = ui;
    cnt[0] = 3 * n;
    sort(id + 1, id + qi + 1, [&](int x,
        int y) { return Q[x] < Q[y]; });
    for (int i = 1; i <= qi; i++) {
        int x = id[i];
        while (Q[x].t > t) apply(++t);
        while (Q[x].t < t) undo(t--);
        while (Q[x].l < l) add(--l);
        while (Q[x].r > r) add(++r);
        while (Q[x].l > l) remove(l++);
        while (Q[x].r < r) remove(r--);
        ans[x] = mex();
    }
    for (int i = 1; i <= qi; i++) cout <<
        ans[i] << '\n';
}

```

## 2.9 SparseTable (Rectangle Query)

```

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

const int MAXN = 505;
const int LOGN = 9;

// O(n^2 (logn)^2)
// Supports Rectangular Query
int A[MAXN][MAXN];
int M[MAXN][MAXN][LOGN][LOGN];

void Build2DSparse(int N) {
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            M[i][j][0][0] = A[i][j];
        }
    }
}

```

```

for (int q = 1; (1 << q) <= N; q++)
{
    int add = 1 << (q - 1);
    for (int j = 1; j + add <= N; j++)
    {
        M[i][j][0][q] = max(M[i][j][0][q - 1], M[i][j + add][0][q - 1]);
    }
}

for (int p = 1; (1 << p) <= N; p++) {
    int add = 1 << (p - 1);
    for (int i = 1; i + add <= N; i++) {
        for (int q = 0; (1 << q) <= N; q++) {
            for (int j = 1; j <= N; j++) {
                M[i][j][p][q] = max(M[i][j][p - 1][q], M[i + add][j][p - 1][q]);
            }
        }
    }
}

```

// returns max of all  $A[i][j]$ , where  $x1 \leq i \leq x2$  and  $y1 \leq j \leq y2$

```

int Query(int x1, int y1, int x2, int y2) {
    int kX = log2(x2 - x1 + 1);
    int kY = log2(y2 - y1 + 1);
    int addX = 1 << kX;
    int addY = 1 << kY;

    int ret1 = max(M[x1][y1][kX][kY], M[x1][y2 - addY + 1][kX][kY]);
    int ret2 = max(M[x2 - addX + 1][y1][kX][kY], M[x2 - addX + 1][y2 - addY + 1][kX][kY]);

    return max(ret1, ret2);
}

```

## 2.10 Sparse Table

```

// 0-based indexing, query finds in range [first, last]
#define lg(x) (31 - __builtin_clz(x))
const int N = 1e5 + 1;
const int K = lg(N);
int a[N], tr[N][K + 1];
namespace sparse_table {
    int f(int p1, int p2) { return min(p1, p2); }
    void build() {
        for (int i = 0; i < n; i++)
            tr[i][0] = a[i];
        for (int j = 1; j <= K; j++) {
            for (int i = 0; i + (1 << j) <= n; i++)
                tr[i][j] = f(tr[i][j - 1], tr[i + (1 << (j - 1))][j - 1]);
        }
    }
    int query(int l, int r) {
        int d = lg(r - l + 1);
        return f(table[l][d], table[r - (1 << d) + 1][d]);
    }
}

```

```

}
}

```

## 3 DP

### 3.1 Convex Hull Trick

```

struct line {
    ll m, c;
    line() {}
    line(ll m, ll c) : m(m), c(c) {}
};

struct convex_hull_trick {
    vector<line> lines;
    int ptr = 0;
    convex_hull_trick() {}
    bool bad(line a, line b, line c) {
        return 1.0 * (c.c - a.c) * (a.m - b.m) < 1.0 * (b.c - a.c) * (a.m - c.m);
    }
    void add(line L) {
        int sz = lines.size();
        while (sz >= 2 && bad(lines[sz - 2], lines[sz - 1], L)) {
            lines.pop_back();
            sz--;
        }
        lines.pb(L);
    }
    ll get(int idx, int x) { return (1ll * lines[idx].m * x + lines[idx].c); }
    ll query(int x) {
        if (lines.empty()) return 0;
        if (ptr >= lines.size()) ptr = lines.size() - 1;
        while (ptr < lines.size() - 1 && get(ptr, x) > get(ptr + 1, x)) ptr++;
        return get(ptr, x);
    }
};

ll sum[MAX];
ll dp[MAX];
int arr[MAX];
int main() {
    fastio;
    int t;
    cin >> t;
    while (t--) {
        int n, a, b, c;
        cin >> n >> a >> b >> c;
        for (int i = 1; i <= n; i++) cin >> sum[i];
        for (int i = 1; i <= n; i++) dp[i] = 0, sum[i] += sum[i - 1];
        convex_hull_trick cht;
        cht.add(line(0, 0));
        for (int pos = 1; pos <= n; pos++) {
            dp[pos] = cht.query(sum[pos]) - 1ll * a * sum[pos] - c;
            cht.add(line(2ll * a * sum[pos], dp[pos] - a * sum[pos]));
        }
        ll ans = (-1ll * dp[n]);
        ans += (1ll * sum[n] * b);
        cout << ans << "\n";
    }
}

```

## 3.2 Dynamic CHT

```

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const ll IS_QUERY = -(1LL << 62);

struct line {
    ll m, b;
    mutable function <const line*>() succ;
};

bool operator < (const line &rhs) const {
    if (rhs.b != IS_QUERY) return m < rhs.m;
    const line *s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
};

struct HullDynamic : public multiset < line > {
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return 1.0 * (x->b - y->b) * (z->m - y->m) >= 1.0 * (y->b - z->b) * (y->m - x->m);
    }

    void insert_line(ll m, ll b) {
        auto y = insert({m, b});
        y->succ = [=] {return next(y) == end() ? 0 : &*next(y);};
        if (bad(y)) {erase(y); return;}
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }

    ll eval(ll x) {
        auto l = *lower_bound((line){x, IS_QUERY});
        return l.m * x + l.b;
    }
};

int main() {
    HullDynamic hull;
    hull.insert_line(1, 1);
    hull.insert_line(-1, 1);
    cout << hull.eval(69) << endl;
    cout << hull.eval(420) << endl;
    return 0;
}

```

}

### 3.3 Li Chao Tree

```

struct line {
    LL m, c;
    line(LL m = 0, LL c = 0) : m(m), c(c) {}
};

LL calc(line L, LL x) { return 1LL * L.m * x + L.c; }

struct node {
    LL m, c;
    line L;
    node *lft, *rt;
    node(LL m = 0, LL c = 0, node *lft = NULL, node *rt = NULL) : L(line(m, c)), lft(lft), rt(rt) {}
};

struct LiChao {
    node *root;
    LiChao() { root = new node(); }
    void update(node *now, int L, int R, line newline) {
        int mid = L + (R - L) / 2;
        line lo = now->L, hi = newline;
        if (calc(lo, L) > calc(hi, L)) swap(lo, hi);
        if (calc(lo, R) <= calc(hi, R)) {
            now->L = hi;
            return;
        }
        if (calc(lo, mid) < calc(hi, mid)) {
            now->L = hi;
            if (now->rt == NULL) now->rt = new node();
            update(now->rt, mid + 1, R, lo);
        } else {
            now->L = lo;
            if (now->lft == NULL) now->lft = new node();
            update(now->lft, L, mid, hi);
        }
    }
    LL query(node *now, int L, int R, LL x) {
        if (now == NULL) return -inf;
        int mid = L + (R - L) / 2;
        if (x <= mid)
            return max(calc(now->L, x), query(now->lft, L, mid, x));
        else
            return max(calc(now->L, x), query(now->rt, mid + 1, R, x));
    }
};

```

## 4 Geometry

### 4.1 Point

```

typedef double Tf;
typedef double Ti; /// use long long for exactness
const Tf PI = acos(-1), EPS = 1e-9;
int dcmp(Tf x) { return abs(x) < EPS ? 0 : (x < 0 ? -1 : 1); }

struct Point {
    Ti x, y;

```

```

    Point(Ti x = 0, Ti y = 0) : x(x), y(y) {}

    Point operator+(const Point& u) const { return Point(x + u.x, y + u.y); }
    Point operator-(const Point& u) const { return Point(x - u.x, y - u.y); }
    Point operator*(const LL u) const { return Point(x * u, y * u); }
    Point operator*(const Tf u) const { return Point(x * u, y * u); }
    Point operator/(const Tf u) const { return Point(x / u, y / u); }

    bool operator==(const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }
    bool operator!=(const Point& u) const { return !(*this == u); }
    bool operator<(const Point& u) const { return dcmp(x - u.x) < 0 || (dcmp(x - u.x) == 0 && dcmp(y - u.y) < 0); }

    Ti dot(Point a, Point b) { return a.x * b.x + a.y * b.y; }
    Ti cross(Point a, Point b) { return a.x * b.y - a.y * b.x; }
    Tf length(Point a) { return sqrt(dot(a, a)); }
    Ti sqLength(Point a) { return dot(a, a); }
    Tf distance(Point a, Point b) { return length(a - b); }
    Tf angle(Point u) { return atan2(u.y, u.x); }

    /// returns angle between oa, ob in (-PI, PI]
    Tf angleBetween(Point a, Point b) {
        Tf ans = angle(b) - angle(a);
        return ans <= -PI ? ans + 2 * PI : (ans > PI ? ans - 2 * PI : ans);
    }
    /// Rotate a ccw by rad radians, Tf Ti same
    Point rotate(Point a, Tf rad) {
        return Point(a.x * cos(rad) - a.y * sin(rad),
                    a.x * sin(rad) + a.y * cos(rad));
    }
    /// rotate a ccw by angle th with cos(th) = co && sin(th) = si, tf ti same
    Point rotatePrecise(Point a, Tf co, Tf si) {
        return Point(a.x * co - a.y * si, a.y * co + a.x * si);
    }
    Point rotate90(Point a) { return Point(-a.y, a.x); }
    /// scales vector a by s such that length of a becomes s, Tf Ti same

```

```

    Point scale(Point a, Tf s) { return a / length(a) * s; }
    /// returns an unit vector perpendicular to vector a, Tf Ti same
    Point normal(Point a) {
        Tf l = length(a);
        return Point(-a.y / l, a.x / l);
    }
    /// returns 1 if c is left of ab, 0 if on ab && -1 if right of ab
    int orient(Point a, Point b, Point c) { return dcmp(cross(b - a, c - a)); }
    /// Use as sort(v.begin(), v.end()), polarComp(0, dir)
    /// Polar comparator around 0 starting at direction dir
    struct polarComp {
        Point o, dir;
        polarComp(Point o = Point(0, 0), Point dir = Point(1, 0)) : o(o), dir(dir) {}
        bool half(Point p) { return dcmp(cross(dir, p)) < 0 || (dcmp(cross(dir, p)) == 0 && dcmp(dot(dir, p)) > 0); }
        bool operator()(Point p, Point q) { return make_tuple(half(p), 0) < make_tuple(half(q), cross(p, q)); }
    };
    struct Segment {
        Point a, b;
        Segment(Point aa, Point bb) : a(aa), b(bb) {}
    };
    typedef Segment Line;
    struct Circle {
        Point o;
        Tf r;
        Circle(Point o = Point(0, 0), Tf r = 0) : o(o), r(r) {}
        /// returns true if point p is in || on the circle
        bool contains(Point p) { return dcmp(sqLength(p - o) - r * r) <= 0; }
        /// returns a point on the circle rad radians away from +X CCW
        Point point(Tf rad) {
            static_assert(is_same<Tf, Ti>::value, "");
            return Point(o.x + cos(rad) * r, o.y + sin(rad) * r);
        }
        /// area of a circular sector with central angle rad
        Tf area(Tf rad = PI + PI) { return rad * r * r / 2; }
        /// area of the circular sector cut by a chord with central angle alpha
        Tf sector(Tf alpha) { return r * r * 0.5 * (alpha - sin(alpha)); }
    };

```

### 4.2 Linear

```

    /// **** LINE LINE INTERSECTION START
    ****

```



```

// returns true if point p is on
segment s
bool onSegment(Point p, Segment s) {
    return dcmp(cross(s.a - p, s.b - p))
        == 0 && dcmp(dot(s.a - p, s.b - p)
        ) <= 0;
}
// returns true if segment p && q touch
or intersect
bool segmentsIntersect(Segment p,
    Segment q) {
    if (onSegment(p.a, q) || onSegment(p.
        b, q)) return true;
    if (onSegment(q.a, p) || onSegment(q.
        b, p)) return true;

    Ti c1 = cross(p.b - p.a, q.a - p.a);
    Ti c2 = cross(p.b - p.a, q.b - p.a);
    Ti c3 = cross(q.b - q.a, p.a - q.a);
    Ti c4 = cross(q.b - q.a, p.b - q.a);
    return dcmp(c1) * dcmp(c2) < 0 &&
        dcmp(c3) * dcmp(c4) < 0;
}
bool linesParallel(Line p, Line q) {
    return dcmp(cross(p.b - p.a, q.b - q.
        a)) == 0;
}
// lines are represented as a ray from
a point: (point, vector)
// returns false if two lines (p, v) &&
(q, w) are parallel or collinear
// true otherwise, intersection point
is stored at o via reference, Tf Ti
Same
bool lineLineIntersection(Point p,
    Point v, Point q, Point w, Point& o
    ) {
    if (dcmp(cross(v, w)) == 0) return
        false;
    Point u = p - q;
    o = p + v * (cross(w, u) / cross(v, w
        ));
    return true;
}
// returns false if two lines p && q
are parallel or collinear
// true otherwise, intersection point
is stored at o via reference
bool lineLineIntersection(Line p, Line
    q, Point& o) {
    return lineLineIntersection(p.a, p.b
        - p.a, q.a, q.b - q.a, o);
}
// returns the distance from point a to
line l
// **** LINE LINE INTERSECTION FINISH
****
Tf distancePointLine(Point p, Line l) {
    return abs(cross(l.b - l.a, p - l.a)
        / length(l.b - l.a));
}
// returns the shortest distance from
point a to segment s
Tf distancePointSegment(Point p,
    Segment s) {
    if (s.a == s.b) return length(p - s.a
        );
    Point v1 = s.b - s.a, v2 = p - s.a,
        v3 = p - s.b;

```

```

    if (dcmp(dot(v1, v2)) < 0)
        return length(v2);
    else if (dcmp(dot(v1, v3)) > 0)
        return length(v3);
    else
        return abs(cross(v1, v2) / length(v1
            ));
}
// returns the shortest distance from
segment p to segment q
Tf distanceSegmentSegment(Segment p,
    Segment q) {
    if (segmentsIntersect(p, q)) return
        0;
    Tf ans = distancePointSegment(p.a, q)
        ;
    ans = min(ans, distancePointSegment(p
        .b, q));
    ans = min(ans, distancePointSegment(q
        .a, p));
    ans = min(ans, distancePointSegment(q
        .b, p));
    return ans;
}
// returns the projection of point p on
line l, Tf Ti Same
Point projectPointLine(Point p, Line l)
    {
        Point v = l.b - l.a;
        return l.a + v * ((Tf)dot(v, p - l.a)
            / dot(v, v));
    }
}
4.3 Circular
// Extremely inaccurate for finding
near touches
// compute intersection of line l with
circle c
// The intersections are given in order
of the ray (l.a, l.b), Tf Ti same
vector<Point> circleLineIntersection(
    Circle c, Line l) {
    vector<Point> ret;
    Point b = l.b - l.a, a = l.a - c.o;
    Tf A = dot(b, b), B = dot(a, b);
    Tf C = dot(a, a) - c.r * c.r, D = B *
        B - A * C;
    if (D < -EPS) return ret;
    ret.push_back(l.a + b * (-B - sqrt(D
        + EPS)) / A);
    if (D > EPS) ret.push_back(l.a + b *
        (-B + sqrt(D)) / A);
    return ret;
}
// signed area of intersection of
circle(c.o, c.r) &&
// triangle(c.o, s.a, s.b) [cross(a-o,
b-o)/2]
Tf circleTriangleIntersectionArea(
    Circle c, Segment s) {
    using Linear::distancePointSegment;
    Tf OA = length(c.o - s.a);
    Tf OB = length(c.o - s.b);
    // sector
    if (dcmp(distancePointSegment(c.o, s)
        - c.r) >= 0)
        return angleBetween(s.a - c.o, s.b -
            c.o) * (c.r * c.r) / 2.0;
    // triangle

```

```

    if (dcmp(OA - c.r) <= 0 && dcmp(OB -
        c.r) <= 0)
        return cross(c.o - s.b, s.a - s.b) /
            2.0;
    // three part: (A, a) (a, b) (b, B)
    vector<Point> Sect =
        circleLineIntersection(c, s);
    return circleTriangleIntersectionArea
        (c, Segment(s.a, Sect[0])) +
        circleTriangleIntersectionArea(
            c, Segment(Sect[0], Sect
                [1])) +
        circleTriangleIntersectionArea(
            c, Segment(Sect[1], s.b));
}
// area of intersecion of circle(c.o, c
.r) && simple polyson(p[])
Tf circlePolyIntersectionArea(Circle c,
    Polygon p) {
    Tf res = 0;
    int n = p.size();
    for (int i = 0; i < n; ++i)
        res +=
            circleTriangleIntersectionArea(
                c, Segment(p[i], p[(i + 1) % n
                    ]));
    return abs(res);
}
// locates circle c2 relative to c1
// interior (d < R - r)
// ----> -2
// interior tangents (d = R - r)
// ----> -1
// concentric (d = 0)
// secants (R - r < d < R + r)
// ----> 0
// exterior tangents (d = R + r)
// ----> 1
// exterior (d > R + r)
// ----> 2
int circleCirclePosition(Circle c1,
    Circle c2) {
    Tf d = length(c1.o - c2.o);
    int in = dcmp(d - abs(c1.r - c2.r)),
        ex = dcmp(d - (c1.r + c2.r));
    return in < 0 ? -2 : in == 0 ? -1 :
        ex == 0 ? 1 : ex > 0 ? 2 : 0;
}
// compute the intersection points
between two circles c1 && c2, Tf Ti
same
vector<Point> circleCircleIntersection(
    Circle c1, Circle c2) {
    vector<Point> ret;
    Tf d = length(c1.o - c2.o);
    if (dcmp(d) == 0) return ret;
    if (dcmp(c1.r + c2.r - d) < 0) return
        ret;
    if (dcmp(abs(c1.r - c2.r) - d) > 0)
        return ret;

    Point v = c2.o - c1.o;
    Tf co = (c1.r * c1.r + sqLength(v) -
        c2.r * c2.r) / (2 * c1.r * length
            (v));
    Tf si = sqrt(abs(1.0 - co * co));
    Point p1 = scale(rotatePrecise(v, co,
        -si), c1.r) + c1.o;

```

```

Point p2 = scale(rotatePrecise(v, co,
    si), c1.r) + c1.o;

ret.push_back(p1);
if (p1 != p2) ret.push_back(p2);
return ret;
}

// intersection area between two
// circles c1, c2
Tf circleCircleIntersectionArea(Circle
    c1, Circle c2) {
    Point AB = c2.o - c1.o;
    Tf d = length(AB);
    if (d >= c1.r + c2.r) return 0;
    if (d + c1.r <= c2.r) return PI * c1.
        r * c1.r;
    if (d + c2.r <= c1.r) return PI * c2.
        r * c2.r;

    Tf alpha1 = acos((c1.r * c1.r + d * d
        - c2.r * c2.r) / (2.0 * c1.r * d
        ));
    Tf alpha2 = acos((c2.r * c2.r + d * d
        - c1.r * c1.r) / (2.0 * c2.r * d
        ));
    return c1.sector(2 * alpha1) + c2.
        sector(2 * alpha2);
}

// returns tangents from a point p to
// circle c, Tf Ti same
vector<Point> pointCircleTangents(Point
    p, Circle c) {
    vector<Point> ret;
    Point u = c.o - p;
    Tf d = length(u);
    if (d < c.r)
        ;
    else if (dcmp(d - c.r) == 0) {
        ret = {rotate(u, PI / 2)};
    } else {
        Tf ang = asin(c.r / d);
        ret = {rotate(u, -ang), rotate(u,
            ang)};
    }
    return ret;
}

// returns the points on tangents that
// touches the circle, Tf Ti Same
vector<Point> pointCircleTangencyPoints
    (Point p, Circle c) {
    Point u = p - c.o;
    Tf d = length(u);
    if (d < c.r)
        return {};
    else if (dcmp(d - c.r) == 0)
        return {c.o + u};
    else {
        Tf ang = acos(c.r / d);
        u = u / length(u) * c.r;
        return {c.o + rotate(u, -ang), c.o +
            rotate(u, ang)};
    }
}

// for two circles c1 && c2, returns
// two list of points a && b
// such that a[i] is on c1 && b[i] is
// c2 && for every i
// Line(a[i], b[i]) is a tangent to
// both circles

// CAUTION: a[i] = b[i] in case they
// touch | -1 for c1 = c2
int circleCircleTangencyPoints(Circle
    c1, Circle c2, vector<Point> &a,
    vector<
        Point>
        &b) {
    a.clear(), b.clear();
    int cnt = 0;
    if (dcmp(c1.r - c2.r) < 0) {
        swap(c1, c2);
        swap(a, b);
    }
    Tf d2 = sqLength(c1.o - c2.o);
    Tf rdif = c1.r - c2.r, rsum = c1.r +
        c2.r;
    if (dcmp(d2 - rdif * rdif) < 0)
        return 0;
    if (dcmp(d2) == 0 && dcmp(c1.r - c2.r
        ) == 0) return -1;

    Tf base = angle(c2.o - c1.o);
    if (dcmp(d2 - rdif * rdif) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(base));
        cnt++;
        return cnt;
    }

    Tf ang = acos((c1.r - c2.r) / sqrt(d2
        ));
    a.push_back(c1.point(base + ang));
    b.push_back(c2.point(base + ang));
    cnt++;
    a.push_back(c1.point(base - ang));
    b.push_back(c2.point(base - ang));
    cnt++;

    if (dcmp(d2 - rsum * rsum) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(PI + base));
        cnt++;
    } else if (dcmp(d2 - rsum * rsum) >
        0) {
        Tf ang = acos((c1.r + c2.r) / sqrt(
            d2));
        a.push_back(c1.point(base + ang));
        b.push_back(c2.point(PI + base + ang
            ));
        cnt++;
        a.push_back(c1.point(base - ang));
        b.push_back(c2.point(PI + base - ang
            ));
        cnt++;
    }
    return cnt;
}

4.4 Convex

// minkowski sum of two polygons in 0(
// n)
Polygon minkowskiSum(Polygon A, Polygon
    B) {
    int n = A.size(), m = B.size();
    rotate(A.begin(), min_element(A.begin
        (), A.end()), A.end());
    rotate(B.begin(), min_element(B.begin
        (), B.end()), B.end());
    A.push_back(A[0]);
    B.push_back(B[0]);
    for (int i = 0; i < n; i++) A[i] = A[
        i + 1] - A[i];
    for (int i = 0; i < m; i++) B[i] = B[
        i + 1] - B[i];

    Polygon C(n + m + 1);
    C[0] = A.back() + B.back();
    merge(A.begin(), A.end() - 1, B.begin
        (), B.end() - 1, C.begin() + 1,
        polarComp(Point(0, 0), Point(0,
            -1)));
    for (int i = 1; i < C.size(); i++) C[
        i] = C[i] + C[i - 1];
    C.pop_back();
    return C;
}

// finds the rectangle with minimum
// area enclosing a convex polygon and
// the rectangle with minimum perimeter
// enclosing a convex polygon
// Tf Ti Same
pair<Tf, Tf>
    rotatingCalipersBoundingBox(const
        Polygon &p) {
    using Linear::distancePointLine;
    int n = p.size();
    int l = 1, r = 1, j = 1;
    Tf area = 1e100;
    Tf perimeter = 1e100;
    for (int i = 0; i < n; i++) {
        Point v = (p[(i + 1) % n] - p[i]) /
            length(p[(i + 1) % n] - p[i]);
        while (dcmp(dot(v, p[r % n] - p[i])
            - dot(v, p[(r + 1) % n] - p[i])
            ) < 0)
            r++;
        while (j < r || dcmp(cross(v, p[j %
            n] - p[i]) -
                cross(v, p[(j +
                    1) % n] - p[
                    i])) < 0)
            j++;
        while (l < j ||
            dcmp(dot(v, p[l % n] - p[i])
                - dot(v, p[(l + 1) % n] -
                    p[i])) > 0)
            l++;
        Tf w = dot(v, p[r % n] - p[i]) - dot
            (v, p[l % n] - p[i]);
        Tf h = distancePointLine(p[j % n],
            Line(p[i], p[(i + 1) % n]));
        area = min(area, w * h);
        perimeter = min(perimeter, 2 * w + 2
            * h);
    }
    return make_pair(area, perimeter);
}

// returns the left side of polygon u
// after cutting it by ray a->b
Polygon cutPolygon(Polygon u, Point a,
    Point b) {
    using Linear::lineLineIntersection;
    using Linear::onSegment;

    Polygon ret;
    int n = u.size();
    for (int i = 0; i < n; i++) {

```



```

Point c = u[i], d = u[(i + 1) % n];
if (dcmp(cross(b - a, c - a)) >= 0)
    ret.push_back(c);
if (dcmp(cross(b - a, d - c)) != 0)
{
    Point t;
    lineLineIntersection(a, b - a, c,
        d - c, t);
    if (onSegment(t, Segment(c, d)))
        ret.push_back(t);
}
}
return ret;
}
// returns true if point p is in or on
// triangle abc
bool pointInTriangle(Point a, Point b,
    Point c, Point p) {
    return dcmp(cross(b - a, p - a)) >= 0
        && dcmp(cross(c - b, p - b)) >=
            0 &&
            dcmp(cross(a - c, p - c)) >= 0;
}
// pt must be in ccw order with no
// three collinear points
// returns inside = -1, on = 0, outside
// = 1
int pointInConvexPolygon(const Polygon
    &pt, Point p) {
    int n = pt.size();
    assert(n >= 3);

    int lo = 1, hi = n - 1;
    while (hi - lo > 1) {
        int mid = (lo + hi) / 2;
        if (dcmp(cross(pt[mid] - pt[0], p -
            pt[0])) > 0)
            lo = mid;
        else
            hi = mid;
    }

    bool in = pointInTriangle(pt[0], pt[
        lo], pt[hi], p);
    if (!in) return 1;

    if (dcmp(cross(pt[lo] - pt[lo - 1], p
        - pt[lo - 1])) == 0) return 0;
    if (dcmp(cross(pt[hi] - pt[lo], p -
        pt[lo])) == 0) return 0;
    if (dcmp(cross(pt[hi] - pt[(hi + 1) %
        n], p - pt[(hi + 1) % n])) == 0)
        return 0;
    return -1;
}
// Extreme Point for a direction is the
// farthest point in that direction
// u is the direction for extremeness
int extremePoint(const Polygon &poly,
    Point u) {
    int n = (int)poly.size();
    int a = 0, b = n;
    while (b - a > 1) {
        int c = (a + b) / 2;
        if (dcmp(dot(poly[c] - poly[(c + 1)
            % n], u)) >= 0 &&
            dcmp(dot(poly[c] - poly[(c - 1 +
                n) % n], u)) >= 0) {
            return c;
        }
    }
}

}

bool a_up = dcmp(dot(poly[(a + 1) %
    n] - poly[a], u)) >= 0;
bool c_up = dcmp(dot(poly[(c + 1) %
    n] - poly[c], u)) >= 0;
bool a_above_c = dcmp(dot(poly[a] -
    poly[c], u)) > 0;

if (a_up && !c_up)
    b = c;
else if (!a_up && c_up)
    a = c;
else if (a_up && c_up) {
    if (a_above_c)
        b = c;
    else
        a = c;
} else {
    if (!a_above_c)
        b = c;
    else
        a = c;
}
}
}

if (dcmp(dot(poly[a] - poly[(a + 1) %
    n], u)) > 0 &&
    dcmp(dot(poly[a] - poly[(a - 1 + n
        ) % n], u)) > 0)
    return a;
return b % n;
}

// For a convex polygon p and a line l,
// returns a list of segments
// of p that touch or intersect line l.
// the i'th segment is considered (p[i
    ], p[(i + 1) modulo |p|])
// #1 If a segment is collinear with
// the line, only that is returned
// #2 Else if l goes through i'th point
// , the i'th segment is added
// Complexity: O(lg |p|)
vector<int> lineConvexPolyIntersection(
    const Polygon &p, Line l) {
    assert((int)p.size() >= 3);
    assert(l.a != l.b);

    int n = p.size();
    vector<int> ret;

    Point v = l.b - l.a;
    int lf = extremePoint(p, rotate90(v))
        ;
    int rt = extremePoint(p, rotate90(v)
        * Ti(-1));
    int olf = orient(l.a, l.b, p[lf]);
    int ort = orient(l.a, l.b, p[rt]);

    if (!olf || !ort) {
        int idx = (!olf ? lf : rt);
        if (orient(l.a, l.b, p[(idx - 1 + n)
            % n]) == 0)
            ret.push_back((idx - 1 + n) % n);
        else
            ret.push_back(idx);
        return ret;
    }
    if (olf == ort) return ret;
}

for (int i = 0; i < 2; ++i) {
    int lo = i ? rt : lf;
    int hi = i ? lf : rt;
    int olo = i ? ort : olf;

    while (true) {
        int gap = (hi - lo + n) % n;
        if (gap < 2) break;

        int mid = (lo + gap / 2) % n;
        int omid = orient(l.a, l.b, p[mid
            ]);
        if (!omid) {
            lo = mid;
            break;
        }
        if (omid == olo)
            lo = mid;
        else
            hi = mid;
    }
    ret.push_back(lo);
}
return ret;
}
// Calculate [ACW, CW] tangent pair
// from an external point
constexpr int CW = -1, ACW = 1;
bool isGood(Point u, Point v, Point Q,
    int dir) {
    return orient(Q, u, v) != -dir;
}
Point better(Point u, Point v, Point Q,
    int dir) {
    return orient(Q, u, v) == dir ? u : v
        ;
}
Point pointPolyTangent(const Polygon &
    pt, Point Q, int dir, int lo, int
    hi) {
    while (hi - lo > 1) {
        int mid = (lo + hi) / 2;
        bool pvs = isGood(pt[mid], pt[mid -
            1], Q, dir);
        bool nxt = isGood(pt[mid], pt[mid +
            1], Q, dir);

        if (pvs && nxt) return pt[mid];
        if (!(pvs || nxt)) {
            Point p1 = pointPolyTangent(pt, Q,
                dir, mid + 1, hi);
            Point p2 = pointPolyTangent(pt, Q,
                dir, lo, mid - 1);
            return better(p1, p2, Q, dir);
        }
        if (!pvs) {
            if (orient(Q, pt[mid], pt[lo]) ==
                dir)
                hi = mid - 1;
            else if (better(pt[lo], pt[hi], Q,
                dir) == pt[lo])
                hi = mid - 1;
            else
                lo = mid + 1;
        }
        if (!nxt) {

```

```

    if (orient(Q, pt[mid], pt[lo]) ==
        dir)
        lo = mid + 1;
    else if (better(pt[lo], pt[hi], Q,
        dir) == pt[lo])
        hi = mid - 1;
    else
        lo = mid + 1;
}
}

Point ret = pt[lo];
for (int i = lo + 1; i <= hi; i++)
    ret = better(ret, pt[i], Q, dir);
return ret;
}

// [ACW, CW] Tangent
pair<Point, Point> pointPolyTangents(
    const Polygon &pt, Point Q) {
    int n = pt.size();
    Point acw_tan = pointPolyTangent(pt,
        Q, ACW, 0, n - 1);
    Point cw_tan = pointPolyTangent(pt, Q,
        CW, 0, n - 1);
    return make_pair(acw_tan, cw_tan);
}

```

#### 4.5 Polygon

```

typedef vector<Point> Polygon;
// removes redundant colinear points
// polygon can't be all colinear points
Polygon RemoveCollinear(const Polygon &
    poly) {
    Polygon ret;
    int n = poly.size();
    for (int i = 0; i < n; i++) {
        Point a = poly[i];
        Point b = poly[(i + 1) % n];
        Point c = poly[(i + 2) % n];
        if (dcmp(cross(b - a, c - b)) != 0
            && (ret.empty() || b != ret.
                back()))
            ret.push_back(b);
    }
    return ret;
}

// returns the signed area of polygon p
// of n vertices
Tf signedPolygonArea(const Polygon &p)
{
    Tf ret = 0;
    for (int i = 0; i < (int)p.size() -
        1; i++)
        ret += cross(p[i] - p[0], p[i + 1] -
            p[0]);
    return ret / 2;
}

// given a polygon p of n vertices,
// generates the convex hull in in CCW
// Tested on https://acm.timus.ru/
// problem.aspx?space=1&num=1185
// Caution: when all points are
// colinear AND removeRedundant ==
// false
// output will be contain duplicate
// points (from upper hull) at back
Polygon convexHull(Polygon p, bool
    removeRedundant) {
    int check = removeRedundant ? 0 : -1;

```

```

    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.
        end());

    int n = p.size();
    Polygon ch(n + n);
    int m = 0; // preparing lower hull
    for (int i = 0; i < n; i++) {
        while (m > 1 &&
            dcmp(cross(ch[m - 1] - ch[m -
                2], p[i] - ch[m - 1]))
                <= check)
            m--;
        ch[m++] = p[i];
    }
    int k = m; // preparing upper hull
    for (int i = n - 2; i >= 0; i--) {
        while (m > k &&
            dcmp(cross(ch[m - 1] - ch[m -
                2], p[i] - ch[m - 2]))
                <= check)
            m--;
        ch[m++] = p[i];
    }
    if (n > 1) m--;
    ch.resize(m);
    return ch;
}

// returns inside = -1, on = 0, outside
// = 1
int pointInPolygon(const Polygon &p,
    Point o) {
    using Linear::onSegment;
    int wn = 0, n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        if (onSegment(o, Segment(p[i], p[j])
            ) || o == p[i]) return 0;
        int k = dcmp(cross(p[j] - p[i], o -
            p[i]));
        int d1 = dcmp(p[i].y - o.y);
        int d2 = dcmp(p[j].y - o.y);
        if (k > 0 && d1 <= 0 && d2 > 0) wn
            ++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn
            --;
    }
    return wn ? -1 : 1;
}

```

```

// Given a simple polygon p, and a line
// l, returns (x, y)
// x = longest segment of l in p, y =
// total length of l in p.
pair<Tf, Tf> linePolygonIntersection(
    Line l, const Polygon &p) {
    using Linear::lineLineIntersection;
    int n = p.size();
    vector<pair<Tf, int>> ev;
    for (int i = 0; i < n; ++i) {
        Point a = p[i], b = p[(i + 1) % n],
            z = p[(i - 1 + n) % n];
        int ora = orient(l.a, l.b, a), orb =
            orient(l.a, l.b, b),
            orz = orient(l.a, l.b, z);
        if (!ora) {
            Tf d = dot(a - l.a, l.b - l.a);
            if (orz && orb) {
                if (orz != orb) ev.emplace_back(
                    d, 0);

```

```

        // else // Point Touch
    } else if (orz)
        ev.emplace_back(d, orz);
    else if (orb)
        ev.emplace_back(d, orb);
} else if (ora == -orb) {
    Point ins;
    lineLineIntersection(l, Line(a, b),
        ins);
    ev.emplace_back(dot(ins - l.a, l.b -
        l.a), 0);
}
}
sort(ev.begin(), ev.end());

Tf ans = 0, len = 0, last = 0, tot =
    0;
bool active = false;
int sign = 0;
for (auto &qq : ev) {
    int tp = qq.second;
    Tf d = qq.first; // current Segment
    // is (last, d)
    if (sign) { // On Border
        len += d - last;
        tot += d - last;
        ans = max(ans, len);
        if (tp != sign) active = !active;
        sign = 0;
    } else {
        if (active) { // Strictly Inside
            len += d - last;
            tot += d - last;
            ans = max(ans, len);
        }
        if (tp == 0)
            active = !active;
        else
            sign = tp;
    }
    last = d;
    if (!active) len = 0;
}
ans /= length(l.b - l.a);
tot /= length(l.b - l.a);
return {ans, tot};
}

```

## 5 Graph

### 5.1 LCA, ETT, VT

```

#define lg(n) (31 - __builtin_clz(n))
const int N = 1e5 + 1;
const int K = lg(N) + 1;
vector<int> adj[N];
int anc[N][K], lvl[N];
namespace lca {
    void init(int u = 1, int p = 0, int d
        = 0) {
        lvl[u] = d;
        anc[u][0] = p;
        for (int i = 1; i < K; i++)
            anc[u][i] = anc[anc[u][i - 1]][i -
                1];
        for (auto v : adj[u])
            if (v != p)
                init(v, u, d + 1);
    }
    int getAnc(int u, int k) {

```

```

    for (int i = 0; u and i < K; i++)
        if ((k >> i) & 1)
            u = anc[u][i];
    return u;
}
int lca(int u, int v) {
    if (lvl[u] < lvl[v]) swap(u, v);
    u = getAnc(u, lvl[u] - lvl[v]);
    if (u == v) return u;
    for (int i = K - 1; ~i; i--)
        if (anc[u][i] != anc[v][i])
            u = anc[u][i], v = anc[v][i];
    return anc[u][0];
}
int dist(int u, int v) {
    return lvl[u] + lvl[v] - 2 * lvl[lca(u, v)];
}
};
struct euler_tour {
    int time = 0;
    tree &T;
    int n;
    vector<int> start, finish, level, par
        ;
    euler_tour(tree &T, int root = 0)
        : T(T), n(T.n), start(n), finish(n)
        , level(n), par(n) {
        time = 0;
        call(root);
    }
    void call(int node, int p = -1) {
        if (p != -1) level[node] = level[p]
            + 1;
        start[node] = time++;
        for (int e : T[node])
            if (e != p) call(e, node);
        par[node] = p;
        finish[node] = time++;
    }
    bool isAncestor(int node, int par) {
        return start[par] <= start[node] and
            finish[par] >= finish[node];
    }
    int subtreeSize(int node) { return
        finish[node] - start[node] + 1 >>
        1; }
};
tree virtual_tree(vector<int> &nodes,
    lca_table &table, euler_tour &tour)
{
    sort(nodes.begin(), nodes.end(),
        [&](int x, int y) { return tour.
            start[x] < tour.start[y]; });
    int n = nodes.size();
    for (int i = 0; i + 1 < n; i++)
        nodes.push_back(table.lca(nodes[i],
            nodes[i + 1]));
    sort(nodes.begin(), nodes.end());
    nodes.erase(unique(nodes.begin(),
        nodes.end()), nodes.end());
    sort(nodes.begin(), nodes.end(),
        [&](int x, int y) { return tour.
            start[x] < tour.start[y]; });
    n = nodes.size();
    stack<int> st;
    st.push(0);
    tree ans(n);
    for (int i = 1; i < n; i++) {

```

```

        while (!tour.isAncestor(nodes[i],
            nodes[st.top()])) st.pop();
        ans.addEdge(st.top(), i);
        st.push(i);
    }
    return ans;
}
set<int> getCenters(tree &T) {
    int n = T.n;
    vector<int> deg(n), q;
    set<int> s;
    for (int i = 0; i < n; i++) {
        deg[i] = T[i].size();
        if (deg[i] == 1) q.push_back(i);
        s.insert(i);
    }
    for (vector<int> t; s.size() > 2; q =
        t) {
        for (auto x : q) {
            for (auto e : T[x])
                if (--deg[e] == 1) t.push_back(e);
            s.erase(x);
        }
    }
    return s;
}

```

## 5.2 SCC

```

typedef long long LL;
const LL N = 1e6 + 7;
bool vis[N];
vector<int> adj[N], adjr[N];
vector<int> order, component;
// tp = 0 ,finding topo order, tp = 1 ,
// reverse edge traversal
void dfs(int u, int tp = 0) {
    vis[u] = true;
    if (tp) component.push_back(u);
    auto& ad = (tp ? adjr : adj);
    for (int v : ad[u])
        if (!vis[v]) dfs(v, tp);
    if (!tp) order.push_back(u);
}
int main() {
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) dfs(i);
    }
    memset(vis, 0, sizeof vis);
    reverse(order.begin(), order.end());
    for (int i : order) {
        if (!vis[i]) {
            // one component is found
            dfs(i, 1), component.clear();
        }
    }
}

```

## 5.3 Euler Tour on Edge

```

// for simplicity, G[idx] contains the
// adjacency list of a node
// while G(e) is a reference to the e-
// th edge.
const int N = 2e5 + 5;
int in[N], out[N], fwd[N], bck[N];
int t = 0;
void dfs(graph &G, int node, int par) {

```

```

    out[node] = t;
    for (int e : G[node]) {
        int v = G(e).to(node);
        if (v == par) continue;
        fwd[e] = t++;
        dfs(G, v, node);
        bck[e] = t++;
    }
    in[node] = t - 1;
}
void init(graph &G, int node) {
    t = 0;
    dfs(G, node, node);
}

```

## 5.4 LCA In O(1)

```

/* LCA in O(1)
 * depth calculates weighted distance
 * level calculates distance by number
 * of edges
 * Preprocessing in NlongN */
LL depth[N];
int level[N];

int st[N], en[N], LOG[N], par[N];
int a[N], id[N], table[L][N];

vector<PII> adj[N];
int n, root, Time, cur;

void init(int nodes, int root_) {
    n = nodes, root = root_, LOG[0] = LOG
        [1] = 0;
    for (int i = 2; i <= n; i++) LOG[i] =
        LOG[i >> 1] + 1;
    for (int i = 0; i <= n; i++) adj[i].
        clear();
}

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

int lca(int u, int v) {
    if (en[u] > en[v]) swap(u, v);
    if (st[v] <= st[u] && en[u] <= en[v])
        return v;

    int l = LOG[id[v] - id[u] + 1];
    int p1 = id[u], p2 = id[v] - (1 << l)
        + 1;
    int d1 = level[table[l][p1]], d2 =
        level[table[l][p2]];

    if (d1 < d2)
        return par[table[l][p1]];
    else
        return par[table[l][p2]];
}

```

```

LL dist(int u, int v) {
    int l = lca(u, v);
    return (depth[u] + depth[v] - (depth[
        l] * 2));
}
/* Euler tour */
void dfs(int u, int p) {

```

```

st[u] = ++Time, par[u] = p;

for (auto [v, w] : adj[u]) {
    if (v == p) continue;
    depth[v] = depth[u] + w;
    level[v] = level[u] + 1;
    dfs(v, u);
}

en[u] = ++Time;
a[++cur] = u, id[u] = cur;
}

/* RMQ */
void pre() {
    cur = Time = 0, dfs(root, root);
    for (int i = 1; i <= n; i++) table
        [0][i] = a[i];

    for (int l = 0; l < L - 1; l++) {
        for (int i = 1; i <= n; i++) {
            table[l + 1][i] = table[l][i];

            bool C1 = (1 << l) + i <= n;
            bool C2 = level[table[l][i] + (1 << l)] < level[table[l][i]];

            if (C1 && C2) table[l + 1][i] =
                table[l][i + (1 << l)];
        }
    }
}

```

## 5.5 HLD

```

const int N = 1e6 + 7;
template <typename DT>
struct Segtree {
    // write lazy segtree here
};
Segtree<int> tree(N);
vector<int> adj[N];
int depth[N], par[N], pos[N];
int head[N], heavy[N], cnt;

int dfs(int u, int p) {
    int SZ = 1, mxsz = 0, heavy;
    depth[u] = depth[p] + 1;

    for (auto v : adj[u]) {
        if (v == p) continue;
        par[v] = u;
        int subsz = dfs(v, u);
        if (subsz > mxsz) heavy[u] = v, mxsz
            = subsz;
        SZ += subsz;
    }
    return SZ;
}

void decompose(int u, int h) {
    head[u] = h, pos[u] = ++cnt;
    if (heavy[u] != -1) decompose(heavy[u], h);

    for (int v : adj[u]) {
        if (v == par[u]) continue;
        if (v != heavy[u]) decompose(v, v);
    }
}

int query(int a, int b) {

```

```

int ret = 0;
for (; head[a] != head[b]; b = par[
    head[b]]) {
    if (depth[head[a]] > depth[head[b]])
        swap(a, b);
    ret += tree.query(1, 0, cnt, pos[
        head[b]], pos[b]);
}

if (depth[a] > depth[b]) swap(a, b);
ret += tree.query(1, 0, cnt, pos[a],
    pos[b]);
return ret;
}

```

## 5.6 Centroid Tree

```

void dfs_size(int u = 1, int p = 0) {
    sz[u] = 1;
    for (auto v : adj[u])
        if (v != p) {
            dfs_size(v, u);
            sz[u] += sz[v];
        }
}

int findCentroid(int u, int p) {
    int total = sz[u];
    for (auto v : adj[u])
        if (v != p and not vis[v] and 2 * sz
            [v] > total) {
            sz[u] = total - sz[v];
            sz[v] = total;
            return findCentroid(v, u);
        }
    return u;
}

int query(int u) {
    int ans = 1e6;
    for (int i = u; i; i = par[i])
        ans = min(ans, minD[i] + lca::dist(i,
            u));
    return ans;
}

void update(int u) {
    for (int i = u; i; i = par[i])
        minD[i] = min(minD[i], lca::dist(i,
            u));
}

int decompose(int u, int p) {
    u = findCentroid(u, p);
    vis[u] = 1;
    for (auto v : adj[u])
        if (not vis[v])
            par[decompose(v, u)] = u;
    return u;
}

```

## 5.7 Dinic Max Flow

```

/**
Implementation of Dinic's algorithm
with optional scaling
Source: Chilli (https://codeforces.com/blog/entry/66006)

Complexity:  $O(ans \cdot E)$  or  $O(V^2 E)$  without
scaling,  $O(VE \log(U))$  with scaling
,
Scaling performs much better in worst
case, but has much higher constant
factor

```

```

To enable scaling, call maxFlow(true)
Everything 0-indexed
*/
namespace Dinic {
    typedef long long LL;
    const int N = 5005, K = 60; // N >
        no of nodes, K >= max bits in
        capacity
    const LL INF = 1e18;
    struct Edge { int frm, to; LL cap,
        flow; };
    int s, t, n;
    int level[N], ptr[N];
    vector<Edge> edges;
    vector<int> adj[N];
    void init(int nodes) {
        n = nodes;
        for (int i = 0; i < n; i++) adj[i].
            clear();
        edges.clear();
    }
    // For adding undirected Edge (u, v
        , c) call addEdge(u, v, c, c);
    int addEdge(int a, int b, LL cap, LL
        revcap = 0) {
        edges.push_back({a, b, cap, 0});
        edges.push_back({b, a, revcap,
            0});
        adj[a].push_back(edges.size() - 2);
        ;
        adj[b].push_back(edges.size() - 1);
        ;
        return edges.size() - 2;
    }
    bool bfs(LL lim) {
        fill(level, level + n, -1);
        level[s] = 0;
        queue<int> q;
        q.push(s);

        while (!q.empty() && level[t] ==
            -1) {
            int v = q.front();
            q.pop();
            for (int id: adj[v]) {
                Edge e = edges[id];
                if (level[e.to] == -1 &&
                    e.cap - e.flow >= lim
                ) {
                    q.push(e.to);
                    level[e.to] = level[v]
                        + 1;
                }
            }
        }
        return level[t] != -1;
    }
    LL dfs(int v, LL flow) {
        if (v == t || !flow) return
            flow;
        for (; ptr[v] < adj[v].size();
            ptr[v]++) {
            int eid = adj[v][ptr[v]];
            Edge &e = edges[eid];
            if (level[e.to] != level[v]
                + 1) continue;
            if (LL pushed = dfs(e.to,
                min(flow, e.cap - e.flow
                ))) {

```

```

        e.flow += pushed;
        edges[eid^1].flow -=
            pushed;
        return pushed;
    }
}
return 0;
}
LL maxFlow(int source, int sink,
    bool SCALING = false) {
    s = source, t = sink;

    long long flow = 0;
    for (LL lim = SCALING ? (1LL <<
        K) : 1; lim > 0; lim >>= 1)
    {
        while (bfs(lim)) {
            fill(ptr, ptr+n, 0);
            while (LL pushed = dfs(s,
                INF)) flow += pushed;
        }
    }
    return flow;
}
bool leftOfMinCut(int x) {return
    level[x] != -1;}
/// Only works for undirected graph,
/// Make sure to add UNDIRECTED
/// edges. (u, v, c, c)
/// returns n by n matrix flow, st
/// flow[i][j] = maxFlow
/// tree holds the edges of a gomory
/// -hu tree of the graph
vector<vector<LL>> allPairMaxFlow(
    vector<Edge> &tree) {
    tree.clear();
    vector<vector<LL>> flow(n,
        vector<LL> (n, INF));
    vector<int> par(n);
    for (int i=1; i<n; i++) {
        for (auto &e: edges) e.flow
            = 0;
        LL f = maxFlow(i, par[i]);
        tree.push_back({i, par[i], f
            });

        for (int j=i+1; j<n; j++)
            if (par[j] == par[i] &&
                leftOfMinCut(j)) par[
                    j] = i;

        flow[i][par[i]] = flow[par[i]
            ][i] = f;
        for (int j=0; j<i; j++)
            if (j != par[i]) flow[i
                ][j] = flow[j][i] =
                    min(f, flow[par[i]][j
                        ]);
    }
    return flow;
}
}

```

## 5.8 Min Cost Max Flow

```

mt19937 rnd(chrono::steady_clock::now()
    .time_since_epoch().count());
const LL inf = 1e9;
struct edge {

```

```

    int v, rev;
    LL cap, cost, flow;
    edge() {}
    edge(int v, int rev, LL cap, LL cost)
        : v(v), rev(rev), cap(cap), cost(
            cost), flow(0) {}
};
struct mcmf {
    int src, sink, n;
    vector<int> par, idx, Q;
    vector<bool> inq;
    vector<LL> dis;
    vector<vector<edge>> g;
    mcmf() {}
    mcmf(int src, int sink, int n)
        : src(src),
          sink(sink),
          n(n),
          par(n),
          idx(n),
          inq(n),
          dis(n),
          g(n),
          Q(10000005) {} // use Q(n) if
                          not using random
    void add_edge(int u, int v, LL cap,
        LL cost, bool directed = true) {
        edge _u = edge(v, g[v].size(), cap,
            cost);
        edge _v = edge(u, g[u].size(), 0, -
            cost);
        g[u].pb(_u);
        g[v].pb(_v);
        if (!directed) add_edge(v, u, cap,
            cost, true);
    }
    bool spfa() {
        for (int i = 0; i < n; i++) {
            dis[i] = inf, inq[i] = false;
        }
        int f = 0, l = 0;
        dis[src] = 0, par[src] = -1, Q[l++]
            = src, inq[src] = true;
        while (f < l) {
            int u = Q[f++];
            for (int i = 0; i < g[u].size(); i
                ++){
                edge &e = g[u][i];
                if (e.cap <= e.flow) continue;
                if (dis[e.v] > dis[u] + e.cost)
                {
                    dis[e.v] = dis[u] + e.cost;
                    par[e.v] = u, idx[e.v] = i;
                    if (!inq[e.v]) inq[e.v] = true
                        , Q[l++] = e.v;
                    // if (!inq[e.v]) {
                    //     inq[e.v] = true;
                    //     if (f && rnd() & 7) Q[--f]
                        = e.v;
                    //     else Q[l++] = e.v;
                    // }
                }
            }
            inq[u] = false;
        }
        return (dis[sink] != inf);
    }
    pair<LL, LL> solve() {
        LL mincost = 0, maxflow = 0;

```

```

        while (spfa()) {
            LL bottleneck = inf;
            for (int u = par[sink], v = idx[
                sink]; u != -1; v = idx[u], u
                = par[u]) {
                edge &e = g[u][v];
                bottleneck = min(bottleneck, e.
                    cap - e.flow);
            }
            for (int u = par[sink], v = idx[
                sink]; u != -1; v = idx[u], u
                = par[u]) {
                edge &e = g[u][v];
                e.flow += bottleneck;
                g[e.v][e.rev].flow -= bottleneck
                    ;
            }
            mincost += bottleneck * dis[sink],
                maxflow += bottleneck;
        }
        return make_pair(mincost, maxflow);
    }
};
// want to minimize cost and don't care
// about flow
// add edge from sink to dummy sink (
// cap = inf, cost = 0)
// add edge from source to sink (cap =
// inf, cost = 0)
// run mcmf, cost returned is the
// minimum cost

```

## 5.9 Bridge Tree

```

vector<vector<int>> components;
vector<int> depth, low;
stack<int> st;
vector<int> id;
vector<edge> bridges;
graph tree;
void find_bridges(int node, graph &G,
    int par = -1, int d = 0) {
    low[node] = depth[node] = d;
    st.push(node);
    for (int id : G[node]) {
        int to = G[id].to(node);
        if (par != to) {
            if (depth[to] == -1) {
                find_bridges(to, G, node, d + 1)
                    ;
                if (low[to] > depth[node]) {
                    bridges.emplace_back(node, to)
                        ;
                    components.push_back({});
                    for (int x = -1; x != to; x =
                        st.top(), st.pop())
                        components.back().push_back(
                            st.top());
                }
            }
            low[node] = min(low[node], low[to
                ]);
        }
    }
    if (par == -1) {
        components.push_back({});
        while (!st.empty()) components.back
            ().push_back(st.top()), st.pop
                ();
    }
}

```



```

}
graph &create_tree() {
    for (auto &comp : components) {
        int idx = tree.addNode();
        for (auto &e : comp) id[e] = idx;
    }
    for (auto &[l, r] : bridges) tree.
        addEdge(id[l], id[r]);
    return tree;
}

void init(graph &G) {
    int n = G.n;
    depth.assign(n, -1), id.assign(n, -1)
        , low.resize(n);
    for (int i = 0; i < n; i++)
        if (depth[i] == -1) find_bridges(i,
            G);
}

```

## 5.10 Tree Isomorphism

```

LL Hash(int u, int p) {
    vector<LL> childrenHash;
    for (auto v : adj[u]) if (v != p)
        childrenHash.add(Hash(v, u));
    sort(all(childrenHash));
    LL nodeHash = 0;
    for (int i = 0; i < childrenHash.size
        ()); i++)
        nodeHash = (nodeHash + childrenHash[
            i] * bigmod(SEED, i, MOD)) %
            MOD;
    return nodeHash;
}

```

## 5.11 Grundy

```

/*
single pile game-> greedy or game dp
multiple pile game and disjunctive(
    before playing, choose 1 pile) ->
    NIM game
else-> Grundy(converts n any game piles
    to n NIM piles)

```

grundy(x)->the smallest nonreachable  
grundy value

there are n pile of games and k type of  
moves.

```

if XOR(grundy(games)) == 0: losing
    state
else winning state
*/

```

```

vector<int> moves, dp;
int mex(vector<int> &a) {
    set<int> b(a.begin(), a.end());
    for (int i = 0; ; ++i)
        if (!b.count(i))
            return i;
}

```

```

int grundy(int x) {
    if (dp[x] != -1) return dp[x];
    vector<int> reachable;
    for (auto m : moves) {
        if (x - m < 0) continue;
        int val = grundy(x - m);
        reachable.push_back(val);
    }
}

```

```

return dp[x] = mex(reachable);
}

```

## 6 Math

### 6.1 Linear Sieve

```

using ULL = unsigned long long;
namespace sieve{
    const int N = 1e7;
    vector <int> primes;
    int spf[N+5], phi[N+5], NOD[N+5], cnt
        [N+5], POW[N+5];
    bool prime[N+5];
    int SOD[N+5];
    void init(){
        fill(prime+2, prime+N+1, 1);
        SOD[1] = NOD[1] = phi[1] = spf[1] =
            1;
        for(LL i=2;i<=N;i++){
            if(prime[i]) {
                primes.push_back(i), spf[i] = i;
                phi[i] = i-1;
                NOD[i] = 2, cnt[i] = 1;
                SOD[i] = i+1, POW[i] = i;
            }
            for(auto p:primes){
                if(p*i>N or p > spf[i]) break;
                prime[p*i] = false, spf[p*i] = p
                    ;
                if(i%p == 0){
                    phi[p*i]=p*phi[i];
                    NOD[p*i]=NOD[i]/(cnt[i]+1)*(
                        cnt[i]+2), cnt[p*i]=cnt[i
                            ]+1;
                    SOD[p*i]=SOD[i]/SOD[POW[i]]*(
                        SOD[POW[i]]+p*POW[i]),POW[
                            p*i]=p*POW[i];
                    break;
                } else {
                    phi[p*i]=phi[p]*phi[i];
                    NOD[p*i]=NOD[p]*NOD[i], cnt[p*
                        i]=1;
                    SOD[p*i]=SOD[p]*SOD[i], POW[p*
                        i]=p;
                }
            }
        }
    }
}

```

### 6.2 Pollard Rho

```

LL mul(LL a, LL b, LL mod) {
    return (__int128)a * b % mod;
    // LL ans = a * b - mod * (LL) (1.L /
        mod * a * b);
    // return ans + mod * (ans < 0) - mod
        * (ans >= (LL) mod);
}

LL bigmod(LL num, LL pow, LL mod) {
    LL ans = 1;
    for (; pow > 0; pow >>= 1, num = mul(
        num, num, mod))
        if (pow & 1) ans = mul(ans, num, mod
            );
    return ans;
}

bool is_prime(LL n) {
    if (n < 2 or n % 6 % 4 != 1) return (
        n | 1) == 3;
}

```

```

LL a[] = {2, 325, 9375, 28178,
    450775, 9780504, 1795265022};
LL s = __builtin_ctzll(n - 1), d = n
    >> s;
for (LL x : a) {
    LL p = bigmod(x % n, d, n), i = s;
    for (; p != 1 and p != n - 1 and x %
        n and i--; p = mul(p, p, n))
        ;
    if (p != n - 1 and i != s) return
        false;
}
return true;
}

LL get_factor(LL n) {
    auto f = [&](LL x) { return mul(x, x,
        n) + 1; };
    LL x = 0, y = 0, t = 0, prod = 2, i =
        2, q;
    for (; t++ % 40 or gcd(prod, n) == 1;
        x = f(x), y = f(f(y))) {
        (x == y) ? x = i++, y = f(x) : 0;
        prod = (q = mul(prod, max(x, y) -
            min(x, y), n)) ? q : prod;
    }
    return gcd(prod, n);
}

map<LL, int> factorize(LL n) {
    map<LL, int> res;
    if (n < 2) return res;
    LL small_primes[] = {2, 3, 5, 7, 11,
        13, 17, 19, 23, 29, 31, 37, 41,
        43, 47, 53, 59,
        61, 67, 71,
        73, 79, 83,
        89, 97};
    for (LL p : small_primes)
        for (; n % p == 0; n /= p, res[p]++)
            ;

    auto _factor = [&](LL n, auto &
        _factor) {
        if (n == 1) return;
        if (is_prime(n))
            res[n]++;
        else {
            LL x = get_factor(n);
            _factor(x, _factor);
            _factor(n / x, _factor);
        }
    };
    _factor(n, _factor);
    return res;
}

```

### 6.3 Extended Euclidean

```

int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q *
            x1);
        tie(y, y1) = make_tuple(y1, y - q *
            y1);
        tie(a1, b1) = make_tuple(b1, a1 - q
            * b1);
    }
    return a1;
}

```

## 6.4 Chinese Remainder Theorem

```
// given a, b will find solutions for
// ax + by = 1
tuple<LL, LL, LL> EGCD(LL a, LL b) {
    if (b == 0)
        return {1, 0, a};
    else {
        auto [x, y, g] = EGCD(b, a % b);
        return {y, x - a / b * y, g};
    }
}

// given modulo equations, will apply
// CRT
PLL CRT(vector<PLL> &v) {
    LL V = 0, M = 1;
    for (auto &[v, m] : v) { // value %
        mod
        auto [x, y, g] = EGCD(M, m);
        if ((v - V) % g != 0) return {-1,
            0};
        V += x * (v - V) / g % (m / g) * M,
        M *= m / g;
        V = (V % M + M) % M;
    }
    return make_pair(V, M);
}
```

## 6.5 Mobius Function

```
const int N = 1e6 + 5;
int mob[N];
void mobius() {
    memset(mob, -1, sizeof mob);
    mob[1] = 1;
    for (int i = 2; i < N; i++)
        if (mob[i]) {
            for (int j = i + i; j < N; j += i)
                mob[j] -= mob[i];
        }
}
```

## 6.6 FFT

```
using CD = complex<double>;
typedef long long LL;
const double PI = acos(-1.0L);

int N;
vector<int> perm;
vector<CD> wp[2];
void precalculate(int n) {
    assert((n & (n - 1)) == 0), N = n;
    perm = vector<int>(N, 0);
    for (int k = 1; k < N; k <= 1) {
        for (int i = 0; i < k; i++) {
            perm[i] <= 1;
            perm[i + k] = 1 + perm[i];
        }
    }
    wp[0] = wp[1] = vector<CD>(N);
    for (int i = 0; i < N; i++) {
        wp[0][i] = CD(cos(2 * PI * i / N),
            sin(2 * PI * i / N));
        wp[1][i] = CD(cos(2 * PI * i / N), -
            sin(2 * PI * i / N));
    }
}

void fft(vector<CD> &v, bool invert =
    false) {
```

```
    if (v.size() != perm.size())
        precalculate(v.size());
    for (int i = 0; i < N; i++)
        if (i < perm[i]) swap(v[i], v[perm[i]]);
    for (int len = 2; len <= N; len *= 2)
        for (int i = 0, d = N / len; i < N;
            i += len) {
            for (int j = 0, idx = 0; j < len /
                2; j++, idx += d) {
                CD x = v[i + j];
                CD y = wp[invert][idx] * v[i + j
                    + len / 2];
                v[i + j] = x + y;
                v[i + j + len / 2] = x - y;
            }
        }
    if (invert) {
        for (int i = 0; i < N; i++) v[i] /=
            N;
    }
}

void pairfft(vector<CD> &a, vector<CD>
    &b, bool invert = false) {
    int N = a.size();
    vector<CD> p(N);
    for (int i = 0; i < N; i++) p[i] = a[
        i] + b[i] * CD(0, 1);
    fft(p, invert);
    p.push_back(p[0]);
    for (int i = 0; i < N; i++) {
        if (invert) {
            a[i] = CD(p[i].real(), 0);
            b[i] = CD(p[i].imag(), 0);
        } else {
            a[i] = (p[i] + conj(p[N - i])) *
                CD(0.5, 0);
            b[i] = (p[i] - conj(p[N - i])) *
                CD(0, -0.5);
        }
    }
}

vector<LL> multiply(const vector<LL> &a
    , const vector<LL> &b) {
    int n = 1;
    while (n < a.size() + b.size()) n <=
        1;
    vector<CD> fa(a.begin(), a.end()), fb
        (b.begin(), b.end());
    fa.resize(n);
    fb.resize(n);
    // fft(fa); fft(fb);
    pairfft(fa, fb);
    for (int i = 0; i < n; i++) fa[i] =
        fa[i] * fb[i];
    fft(fa, true);
    vector<LL> ans(n);
    for (int i = 0; i < n; i++) ans[i] =
        round(fa[i].real());
    return ans;
}

const int M = 1e9 + 7, B = sqrt(M) + 1;
vector<LL> anyMod(const vector<LL> &a,
    const vector<LL> &b) {
    int n = 1;
    while (n < a.size() + b.size()) n <=
        1;
```

```
vector<CD> al(n), ar(n), bl(n), br(n)
    ;
    for (int i = 0; i < a.size(); i++) al
        [i] = a[i] % M / B, ar[i] = a[i]
        % M % B;
    for (int i = 0; i < b.size(); i++) bl
        [i] = b[i] % M / B, br[i] = b[i]
        % M % B;
    pairfft(al, ar);
    pairfft(bl, br);
    // fft(al); fft(ar); fft(bl);
    fft(br);
    for (int i = 0; i < n; i++) {
        CD ll = (al[i] * bl[i]), lr = (al[i]
            * br[i]);
        CD rl = (ar[i] * bl[i]), rr = (ar[i]
            * br[i]);
        al[i] = ll;
        ar[i] = lr;
        bl[i] = rl;
        br[i] = rr;
    }
    pairfft(al, ar, true);
    pairfft(bl, br, true);
    // fft(al, true); fft(ar, true)
    ; fft(bl, true); fft(br, true);
    vector<LL> ans(n);
    for (int i = 0; i < n; i++) {
        LL right = round(br[i].real()), left
            = round(al[i].real());
        ;
        LL mid = round(round(bl[i].real()) +
            round(ar[i].real()));
        ans[i] = ((left % M) * B * B + (mid
            % M) * B + right) % M;
    }
    return ans;
}
```

## 6.7 NTT

```
const LL N = 1 << 18;
const LL MOD = 786433;

vector<LL> P[N];
LL rev[N], w[N | 1], a[N], b[N], inv_n,
    g;
LL Pow(LL b, LL p) {
    LL ret = 1;
    while (p) {
        if (p & 1) ret = (ret * b) % MOD;
        b = (b * b) % MOD;
        p >>= 1;
    }
    return ret;
}

LL primitive_root(LL p) {
    vector<LL> factor;
    LL phi = p - 1, n = phi;
    for (LL i = 2; i * i <= n; i++) {
        if (n % i) continue;
        factor.emplace_back(i);
        while (n % i == 0) n /= i;
    }
    if (n > 1) factor.emplace_back(n);
    for (LL res = 2; res <= p; res++) {
        bool ok = true;
        for (LL i = 0; i < factor.size() &&
            ok; i++)
```

```

    ok &= Pow(res, phi / factor[i]) !=
        1;
    if (ok) return res;
}
return -1;
}

void prepare(LL n) {
    LL sz = abs(31 - __builtin_clz(n));
    LL r = Pow(g, (MOD - 1) / n);
    inv_n = Pow(n, MOD - 2);
    w[0] = w[n] = 1;
    for (LL i = 1; i < n; i++) w[i] = (w[
        i - 1] * r) % MOD;
    for (LL i = 1; i < n; i++)
        rev[i] = (rev[i >> 1] >> 1) | ((i &
            1) << (sz - 1));
}

void NTT(LL *a, LL n, LL dir = 0) {
    for (LL i = 1; i < n - 1; i++) {
        if (i < rev[i]) swap(a[i], a[rev[i]
            ]]);
    }
    for (LL m = 2; m <= n; m <= 1) {
        for (LL i = 0; i < n; i += m) {
            for (LL j = 0; j < (m >> 1); j++) {
                LL &u = a[i + j], &v = a[i + j +
                    (m >> 1)];
                LL t = v * w[dir ? n - n / m * j
                    : n / m * j] % MOD;
                v = u - t < 0 ? u - t + MOD : u
                    - t;
                u = u + t >= MOD ? u + t - MOD :
                    u + t;
            }
        }
    }
    if (dir)
        for (LL i = 0; i < n; i++) a[i] = (
            inv_n * a[i]) % MOD;
}

vector<LL> mul(vector<LL> p, vector<LL>
    q) {
    LL n = p.size(), m = q.size();
    LL t = n + m - 1, sz = 1;
    while (sz < t) sz <= 1;
    prepare(sz);

    for (LL i = 0; i < n; i++) a[i] = p[i]
        ];
    for (LL i = 0; i < m; i++) b[i] = q[i]
        ];
    for (LL i = n; i < sz; i++) a[i] = 0;
    for (LL i = m; i < sz; i++) b[i] = 0;

    NTT(a, sz);
    NTT(b, sz);
    for (LL i = 0; i < sz; i++) a[i] = (a[
        i] * b[i]) % MOD;
    NTT(a, sz, 1);

    vector<LL> c(a, a + sz);
    while (c.size() && c.back() == 0) c.
        pop_back();
    return c;
}

```

## 7 String

### 7.1 Aho Corasick

```

const int sg = 26, N = 1e3 + 9;
struct aho_corasick {
    struct node {
        node *link, *out, *par;
        bool leaf;
        LL val;
        int cnt, last, len;
        char p_ch;
        array<node*, sg> to;
        node(node* par = NULL, char p_ch = '
            $', int len = 0)
            : par(par), p_ch(p_ch), len(len)
            {}
        val = leaf = cnt = last = 0;
        link = out = NULL;
    };
    vector<node> trie;
    node* root;
    aho_corasick() {
        trie.reserve(N), trie.emplace_back()
            ;
        root = &trie[0];
        root->link = root->out = root;
    }
    inline int f(char c) { return c - 'a'
        ; }
    inline node* add_node(node* par =
        NULL, char p_ch = '$', int len =
        0) {
        trie.emplace_back(par, p_ch, len);
        return &trie.back();
    }
    void add_str(const string& s, LL val
        = 1) {
        node* now = root;
        for (char c : s) {
            int i = f(c);
            if (!now->to[i]) now->to[i] =
                add_node(now, c, now->len + 1)
                ;
            now = now->to[i];
        }
        now->leaf = true, now->val++;
    }
    void push_links() {
        queue<node*> q;
        for (q.push(root); q.empty(); q.pop
            ()) {
            node *cur = q.front(), *link = cur
                ->link;
            cur->out = link->leaf ? link :
                link->out;
            int idx = 0;
            for (auto& next : cur->to) {
                if (next != NULL) {
                    next->link = cur != root ?
                        link->to[idx++] : root;
                    q.push(next);
                } else
                    next = link->to[idx++];
            }
            cur->val += link->val;
        }
    };
};

```

### 7.2 Double hash

```
/*
```

```

* Some well known primes:
* 1949313259, 1997293877, 2091573227,
    2117566807
* Some Primes:
* 1000000007, 1000000009, 1000000861,
    1000099999 ( < 2^30 )
* 1088888881, 1111211111, 1500000001,
    1481481481 ( < 2^31 )
* 2147483647 (2^31-1),
*/
PLL base(1949313259, 1997293877);
namespace Hashing {
    using LL = long long;
    using PLL = pair<LL,LL>;
    #define ff first
    #define ss second
    const PLL M = {1e9+7, 1e9+9}; //
        /Should be large primes
    const LL base = 1259; //
        /Should be larger than alphabet
        size
    const int N = 1e6+7; //
        /Highest length of string
    PLL operator+ (const PLL& a, LL x)
        {return {a.ff + x, a.ss + x};}
    PLL operator- (const PLL& a, LL x)
        {return {a.ff - x, a.ss - x};}
    PLL operator* (const PLL& a, LL x)
        {return {a.ff * x, a.ss * x};}
    PLL operator+ (const PLL& a, PLL x)
        {return {a.ff + x.ff, a.ss + x.
            ss};}
    PLL operator- (const PLL& a, PLL x)
        {return {a.ff - x.ff, a.ss - x.
            ss};}
    PLL operator* (const PLL& a, PLL x)
        {return {a.ff * x.ff, a.ss * x.
            ss};}
    PLL operator% (const PLL& a, PLL m)
        {return {a.ff % m.ff, a.ss % m.
            ss};}
    ostream& operator<<(ostream& os, PLL
        hash) {
        return os<<"("<<hash.ff<<" , "<<
            hash.ss<<"");
    }
    PLL pb[N]; //powers of base mod
        M
    ///Call pre before everything
    void hashPre() {
        pb[0] = {1,1};
        for (int i=1; i<N; i++) pb[i] =
            (pb[i-1] * base)%M;
    }
    ///Calculates hashes of all prefixes
        of s including empty prefix
    vector<PLL> hashList(string s) {
        int n = s.size();
        vector<PLL> ans(n+1);
        ans[0] = {0,0};
        for (int i=1; i<=n; i++) ans[i]
            = (ans[i-1] * base + s[i-1])
            %M;
        return ans;
    }
    ///Calculates hash of substring s[l
        ..r] (1 indexed)
    PLL substringHash(const vector<PLL>
        &hashlist, int l, int r) {

```

```

return (hashlist[r]+(M-hashlist[
1-1])*pb[r-1+1])%M;
}
///Calculates Hash of a string
PLL Hash (string s) {
    PLL ans = {0,0};
    for (int i=0; i<s.size(); i++)
        ans=(ans*base + s[i])%M;
    return ans;
}
///Tested on https://toph.co/p/
palindromist
///appends c to string
PLL append(PLL cur, char c) {
    return (cur*base + c)%M;
}
///Tested on https://toph.co/p/
palindromist
///prepends c to string with size k
PLL prepend(PLL cur, int k, char c)
{
    return (pb[k]*c + cur)%M;
}
///Tested on https://toph.co/p/
chikongunia
///replaces the i-th (0-indexed)
character from right from a to
b;
PLL replace(PLL cur, int i, char a,
char b) {
    return cur + pb[i] * (M+b-a)%M;
}
///Erases c from front of the string
with size len
PLL pop_front(PLL hash, int len,
char c) {
    return (hash + pb[len-1]*(M-c))%
M;
}
///Tested on https://toph.co/p/
palindromist
///concatenates two strings where
length of the right is k
PLL concat(PLL left, PLL right, int
k) {
    return (left*pb[k] + right)%M;
}
PLL power (const PLL& a, LL p) {
    if (p==0) return {1,1};
    PLL ans = power(a, p/2);
    ans = (ans * ans)%M;
    if (p%2) ans = (ans*a)%M;
    return ans;
}
PLL inverse(PLL a) {
    if (M.ss == 1) return power(a, M
.ff-2);
    return power(a, (M.ff-1)*(M.ss
-1)-1);
}
///Erases c from the back of the
string
PLL invb = inverse({base, base});
PLL pop_back(PLL hash, char c) {
    return ((hash-c*M)*invb)%M;
}
///Tested on https://toph.co/p/
palindromist

```

```

///Calculates hash of string with
size len repeated cnt times
///This is O(log n). For O(1), pre-
calculate inverses
PLL repeat(PLL hash, int len, LL cnt
) {
    PLL mul = ((pb[len*cnt]-1+M) *
inverse(pb[len]-1+M))%M;
    PLL ans = (hash*mul);
    if (pb[len].ff == 1) ans.ff =
hash.ff*cnt;
    if (pb[len].ss == 1) ans.ss =
hash.ss*cnt;
    return ans%M;
}
struct pair_hash {
    inline std::size_t operator()(
const std::pair<LL,LL> & v)
const {
    return v.first*31+v.second;
}
};

```

### 7.3 KMP

```

vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        j += s[i] == s[j];
        pi[i] = j;
    }
    return pi;
}

```

### 7.4 Manacher's

```

vector<int> d1(n);
// d[i] = number of palindromes taking
s[i] as center
for (int i = 0, l = 0, r = -1; i < n; i
++) {
    int k = (i > r) ? 1 : min(d1[l + r -
i], r - i + 1);
    while (0 <= i - k && i + k < n && s[i
- k] == s[i + k]) k++;
    d1[i] = k--;
    if (i + k > r) l = i - k, r = i + k;
}
vector<int> d2(n);
// d[i] = number of palindromes taking
s[i-1] and s[i] as center
for (int i = 0, l = 0, r = -1; i < n; i
++) {
    int k = (i > r) ? 0 : min(d2[l + r -
i + 1], r - i + 1);
    while (0 <= i - k - 1 && i + k < n &&
s[i - k - 1] == s[i + k]) k++;
    d2[i] = k--;
    if (i + k > r) l = i - k - 1, r = i +
k;
}

```

### 7.5 String Match FFT

```

//find occurrences of t in s where '?'s
are automatically matched with any
character

```

```

//res[i + m - 1] = sum_j=0 to m - 1_{s[
i + j] * t[j] * (s[i + j] - t[j])}
vector<int> string_matching(string &s,
string &t) {
    int n = s.size(), m = t.size();
    vector<int> s1(n), s2(n), s3(n);
    for(int i = 0; i < n; i++) s1[i] = s[
i] == '?' ? 0 : s[i] - 'a' + 1;
    //assign any non zero number for
non '?'s
    for(int i = 0; i < n; i++) s2[i] = s1
[i] * s1[i];
    for(int i = 0; i < n; i++) s3[i] = s1
[i] * s2[i];
    vector<int> t1(m), t2(m), t3(m);
    for(int i = 0; i < m; i++) t1[i] = t[
i] == '?' ? 0 : t[i] - 'a' + 1;
    for(int i = 0; i < m; i++) t2[i] = t1
[i] * t1[i];
    for(int i = 0; i < m; i++) t3[i] = t1
[i] * t2[i];
    reverse(t1.begin(), t1.end());
    reverse(t2.begin(), t2.end());
    reverse(t3.begin(), t3.end());
    vector<int> s1t3 = multiply(s1, t3);
    vector<int> s2t2 = multiply(s2, t2);
    vector<int> s3t1 = multiply(s3, t1);
    vector<int> res(n);
    for(int i = 0; i < n; i++) res[i] =
s1t3[i] - s2t2[i] * 2 + s3t1[i];
    vector<int> oc;
    for(int i = m - 1; i < n; i++) if(res
[i] == 0) oc.push_back(i - m + 1);
    ;
    return oc;
}

```

### 7.6 Suffix Array

```

void inducedSort (const vector <int> &
vec, int val_range, vector <int> &
SA, const vector <int> &sl, const
vector <int> &lms_idx) {
    vector <int> l(val_range, 0), r(
val_range, 0);
    for (int c : vec) {
        ++r[c]; if (c + 1 < val_range) ++l[c
+ 1];
    }
    partial_sum(l.begin(), l.end(), l.
begin());
    partial_sum(r.begin(), r.end(), r.
begin());
    fill(SA.begin(), SA.end(), -1);
    for (int i = lms_idx.size() - 1; i >=
0; --i) SA[--r[vec[lms_idx[i]]]]
= lms_idx[i];
    for (int i : SA) if (i > 0 and sl[i -
1]) SA[l[vec[i - 1]]++] = i - 1;
    fill(r.begin(), r.end(), 0);
    for (int c : vec) ++r[c];
    partial_sum(r.begin(), r.end(), r.
begin());
    for (int k = SA.size() - 1, i = SA[k
]; k; --k, i = SA[k]) {
        if (i and !sl[i - 1]) SA[--r[vec[i -
1]]] = i - 1;
    }
}

```

```

vector<int> suffixArray (const vector
<int> &vec, int val_range) {
    const int n = vec.size();
    vector<int> sl(n), SA(n), lms_idx;
    for (int i = n - 2; i >= 0; --i) {
        sl[i] = vec[i] > vec[i + 1] or (vec[
            i] == vec[i + 1] and sl[i + 1])
        ;
        if (sl[i] and !sl[i + 1]) lms_idx.
            emplace_back(i + 1);
    }
    reverse(lms_idx.begin(), lms_idx.end
        ());
    inducedSort(vec, val_range, SA, sl,
        lms_idx);
    vector<int> new_lms_idx(lms_idx.size
        ()), lms_vec(lms_idx.size());
    for (int i = 0, k = 0; i < n; ++i) {
        if (SA[i] > 0 and !sl[SA[i]] and sl[
            SA[i] - 1]) new_lms_idx[k++] =
            SA[i];
    }
    int cur = 0; SA[n - 1] = 0;
    for (int k = 1; k < new_lms_idx.size
        ()); ++k) {
        int i = new_lms_idx[k - 1], j =
            new_lms_idx[k];
        if (vec[i] ^ vec[j]) {
            SA[j] = ++cur; continue;
        }
        bool flag = 0;
        for (int a = i + 1, b = j + 1; ; ++a
            , ++b) {
            if (vec[a] ^ vec[b]) {
                flag = 1; break;
            }
            if ((!sl[a] and sl[a - 1]) or (!sl
                [b] and sl[b - 1])) {
                flag = !(sl[a] and sl[a - 1]
                    and !sl[b] and sl[b - 1]);
                break;
            }
        }
        SA[j] = flag ? ++cur : cur;
    }
    for (int i = 0; i < lms_idx.size();
        ++i) lms_vec[i] = SA[lms_idx[i]];
    if (cur + 1 < lms_idx.size()) {
        auto lms_SA = suffixArray(lms_vec,
            cur + 1);
        for (int i = 0; i < lms_idx.size();
            ++i) new_lms_idx[i] = lms_idx[
                lms_SA[i]];
    }
    inducedSort(vec, val_range, SA, sl,
        new_lms_idx); return SA;
}

vector<int> getSuffixArray (const
    string &s, const int LIM = 128) {
    vector<int> vec(s.size() + 1);
    copy(begin(s), end(s), begin(vec));
    vec.back() = '$';
    auto ret = suffixArray(vec, LIM);
    ret.erase(ret.begin()); return ret;
}

// build RMQ on it to get LCP of any
    two suffix

```

```

vector<int> getLCParray (const string
    &s, const vector<int> &SA) {
    int n = s.size(), k = 0;
    vector<int> lcp(n), rank(n);
    for (int i = 0; i < n; ++i) rank[SA[i
        ]] = i;
    for (int i = 0; i < n; ++i, k ? --k :
        0) {
        if (rank[i] == n - 1) {
            k = 0; continue;
        }
        int j = SA[rank[i] + 1];
        while (i + k < n and j + k < n and s
            [i + k] == s[j + k]) ++k;
        lcp[rank[i]] = k;
    }
    lcp[n - 1] = 0; return lcp;
}

int main() {
    string s; cin >> s;
    for (const int i : getSuffixArray(s))
        printf("%d ", i);
    puts("");
    return 0;
}

7.7 Trie

template<int sz>
struct Trie {
    Trie() : id(1) {
        memset(endMark, 0, sizeof endMark);
        for_each(all(trie), [](vector<int> &
            v) { v.assign(sz, 0); });
    }

    void insert(const string &s) {
        int cur = 0;
        for (auto c : s) {
            int val = c - 'a';
            if (not trie[cur][val])
                trie[cur][val] = id++;
            cur = trie[cur][val];
        }
        endMark[cur] = true;
    }

    bool search(const string &s) {
        int cur = 0;
        for (auto c : s) {
            int val = c - 'a';
            if (not trie[cur][val])
                return false;
            cur = trie[cur][val];
        }
        return endMark[cur];
    }
};

private:
    int id, endMark[100005];
    vector<int> trie[100005];
};

```

## 7.8 Z Algo

```

vector<int> calcz(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        if (i > r) {

```

```

            l = r = i;
            while (r < n && s[r] == s[r - 1])
                r++;
            z[i] = r - l, r--;
        } else {
            int k = i - l;
            if (z[k] < r - i + 1) z[i] = z[k];
            else {
                l = i;
                while (r < n && s[r] == s[r - 1]
                    ) r++;
                z[i] = r - l, r--;
            }
        }
    }
    return z;
}

```

## 8 Extra

### 8.1 Stress Tester

```

# $2 is good code
# $3 is generator
g++ -O2 -std=c++17 "$1".cpp -o $1
g++ -O2 -std=c++17 "$2".cpp -o $2
g++ -O2 -std=c++17 "$3".cpp -o $3
for ((i = 1;; i++)); do

```

```

    echo 'Test #'$i
    timeout 5s ./ $3 $RANDOM > in
    timeout 5s ./ $1 < in > out
    timeout 5s ./ $2 < in > ans
    diff -yi ans out > diff.out
    if [ $? -ne 0 ]; then
        echo "\nInput:"
        cat in
        echo "\nDiff:"
        cat diff.out
        break
    fi
done

```

### 8.2 Sublime Build

```

{
    "shell_cmd": "g++ -O2 -std=c++17 -g -
        DLOCAL -Wall -Wextra -Wpedantic -
        Wfloat-equal -Wshift-overflow=2 -
        fsanitize=address -fsanitize=
        undefined -fno-sanitize-recover
        $file_name -o $file_base_name &&
        timeout 5s ./ $file_base_name < in
        > out",
    "working_dir": "$file_path",
    "selector": "source.cpp"
}

```

### 8.3 vimrc

```

" Auto import & Compile
:autocmd BufNewFile *.cpp Or ~/template
    .cpp
nnoremap <F4> :!xclip -o -sel clip > ~/
    cp/in <CR><CR>
inoremap <F4> <ESC>:!xclip -o -sel clip
    > ~/cp/in <CR><CR>
nnoremap <F6> :!xclip -sel clip % <CR><
    CR>
inoremap <F6> <ESC>:!xclip -sel clip %
    <CR><CR>
autocmd filetype cpp nnoremap <F9> :
    wa \! | g++ -O2 -std=c++17 % -o %:r

```



```

    && timeout 5s ./%:r < ~/cp/in> ~/cp
/out<CR>
autocmd filetype cpp inoremap <F9> <ESC
>:wa \! !g++ -O2 -std=c++17 % -o %:
r && timeout 5s ./%:r < ~/cp/in> ~/
cp/out<CR>
autocmd filetype cpp noremap <F10> :
wa \! !make %:r D=1 && ./%:r < ~/cp
/in> ~/cp/out<CR>
autocmd filetype cpp inoremap <F10> <
ESC>:wa \! !make clean && make %:r
D=1 && ./%:r < ~/cp/in > ~/cp/out<
CR>
" Auto Completion
inoremap ( (<left>
inoremap <expr> ) strpart(getline('.'),
col('.')-1, 1) == ")" ? "<Right>"
: ")"
inoremap { {<left>
inoremap <expr> } strpart(getline('.'),
col('.')-1, 1) == "]" ? "<Right>"
: "]"
inoremap [ [<left>
inoremap <expr> ] strpart(getline('.'),
col('.')-1, 1) == "]" ? "<Right>"
: "]"
inoremap <expr> " strpart(getline('.'),
col('.')-1, 1) == "\" ? "<Right>"
: "\"<left>"
inoremap <expr> ' strpart(getline('.'),
col('.')-1, 1) == "\"" ? "<Right>"
: "\"<left>"
inoremap <expr> <CR> <sid>
insert_newline()
function s:insert_newline() abort
let pair = strpart(getline('.'), col(
')-2, 2)
return stridx('{}[]', pair) % 2 ==
0 && strlen(pair) == 2 ? "<CR><
ESC>\0" : "<CR>"
endfunction
inoremap <expr> <space> <sid>
insert_space()
function s:insert_space() abort
let pair = strpart(getline('.'), col(
')-2, 2)
return stridx('{}[]', pair) % 2 ==
0 && strlen(pair) == 2 ? "<space>
>\<space><left>" : "<space>"
endfunction
inoremap <expr> <bs> <sid>rm_pair()
function s:rm_pair() abort
let pair = strpart(getline('.'), col(
')-2, 2)
return stridx('{}[]''''''''', pair) %
2 == 0 && strlen(pair) == 2 ? "<
del><bs>" : "<bs>"
endfunction
set nocompatible " be
improved, required
filetype on " required
filetype plugin on
filetype plugin indent on
syntax on
set splitright splitbelow
set mouse=a
set number
set relativenumber
set tabstop=2
set shiftwidth=2
set expandtab
set softtabstop=2
set smartindent
set smarttab
set autoindent
set cindent
set noerrorbells
set ruler
set guifont=*
set backspace=indent,eol,start
" set ignorecase
set incsearch
set nowrap
set hlsearch
" set termguicolors
set foldmethod=indent
set nofoldenable
" set cursorline
set laststatus=2
set showcmd
set wildmenu
" colorscheme torte
if !has('nvim')
set clipboard=unnamedplus
endif
if !has('nvim')
set ttymouse=xterm2
endif
nnoremap <S-j> :m .+1<CR>==
nnoremap <S-k> :m .-2<CR>==
vnoremap <S-j> :m '>+1<CR>gv==gv
vnoremap <S-k> :m '<-2<CR>gv==gv
nnoremap <A-h> <C-w>h
nnoremap <A-j> <C-w>j
nnoremap <A-k> <C-w>k
nnoremap <A-l> <C-w>l
let mapleader = ','
map <leader>cp :50 vsplit in<CR>:split
out<CR><C-w>h

```

## 9 Equations and Formulas

### 9.1 Catalan Numbers

$$C_n = \frac{1}{n+1} \binom{2n}{n} \quad C_0 = 1, C_1 = 1$$

$$1 \text{ and } C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

The number of ways to completely parenthesize  $n+1$  factors.

The number of triangulations of a convex polygon with  $n+2$  sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).

The number of ways to connect the  $2n$  points on a circle to form  $n$  disjoint i.e. non-intersecting chords.

The number of rooted full binary trees with  $n+1$  leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.

Number of permutations of  $1, \dots, n$  that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For  $n = 3$ , these permutations are 132, 213, 231, 312 and 321.

### 9.2 Stirling Numbers First Kind

The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).

$S(n, k)$  counts the number of permutations of  $n$  elements with  $k$  disjoint cycles.

$S(n, k) = (n-1) \cdot S(n-1, k) + S(n-1, k-1)$ , where,  $S(0, 0) = 1, S(n, 0) = S(0, n) = 0$

$$\sum_{k=0}^n S(n, k) = n!$$

The unsigned Stirling numbers may also be defined algebraically, as the coefficient of the rising factorial:

$$x^{\bar{n}} = x(x+1)\dots(x+n-1) = \sum_{k=0}^n S(n, k) x^k$$

Lets  $[n, k]$  be the stirling number of the first kind, then

$$\left[ \begin{matrix} n \\ k \end{matrix} \right] = \sum_{0 \leq i_1 < i_2 < \dots < i_k < n} i_1 i_2 \dots i_k.$$

### 9.3 Stirling Numbers Second Kind

Stirling number of the second kind is the number of ways to partition a set of  $n$  objects into  $k$  non-empty subsets.

$S(n, k) = k \cdot S(n-1, k) + S(n-1, k-1)$ , where  $S(0, 0) = 1, S(n, 0) = S(0, n) = 0$   
 $S(n, 2) = 2^{n-1} - 1$   $S(n, k) \cdot k! =$  number of ways to color  $n$  nodes using colors from 1 to  $k$  such that each color is used at least once.

An  $r$ -associated Stirling number of the second kind is the number of ways to partition a set of  $n$  objects into  $k$  subsets,

with each subset containing at least  $r$  elements. It is denoted by  $S_r(n, k)$  and obeys the recurrence relation.  $S_r(n+1, k) =$

$$k S_r(n, k) + \binom{n}{r-1} S_r(n-r+1, k-1)$$

Denote the  $n$  objects to partition by the integers  $1, 2, \dots, n$ . Define the reduced Stirling numbers of the second kind, denoted  $S^d(n, k)$ , to be the number of ways to partition the integers  $1, 2, \dots, n$  into  $k$  nonempty subsets such that all elements in each subset have pairwise distance at least  $d$ . That is, for any integers  $i$  and  $j$  in a given subset, it is required that  $|i - j| \geq d$ . It has been shown that these numbers satisfy,  $S^d(n, k) = S(n-d+1, k-d+1), n \geq k \geq d$

### 9.4 Other Combinatorial Identities

$$\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$$

$$\sum_{i=0}^k \binom{n+i}{i} = \sum_{i=0}^k \binom{n+i}{n} = \binom{n+k+1}{k}$$

$$n, r \in N, n > r, \sum_{i=r}^n \binom{i}{r} = \binom{n+1}{r+1}$$

$$\text{If } P(n) = \sum_{k=0}^n \binom{n}{k} \cdot Q(k), \text{ then,}$$

$$Q(n) = \sum_{k=0}^n (-1)^{n-k} \binom{n}{k} \cdot P(k)$$

$$\text{If } P(n) = \sum_{k=0}^n (-1)^k \binom{n}{k} \cdot Q(k), \text{ then,}$$

$$Q(n) = \sum_{k=0}^n (-1)^k \binom{n}{k} \cdot P(k)$$

### 9.5 Different Math Formulas

**Picks Theorem :**  $A = i + b/2 - 1$

**Derangements :**  $d(i) = (i-1) \times (d(i-1) + d(i-2))$

$$\frac{n}{ab} - \left\{ \frac{b/n}{a} \right\} - \left\{ \frac{a/n}{b} \right\} + 1$$

### 9.6 GCD and LCM

if  $m$  is any integer, then  $\gcd(a + m \cdot b, b) = \gcd(a, b)$

The gcd is a multiplicative function in the following sense: if  $a_1$  and  $a_2$  are relatively prime, then  $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$ .

$\gcd(a, \text{lcm}(b, c)) = \text{lcm}(\gcd(a, b), \gcd(a, c))$ .  
 $\text{lcm}(a, \gcd(b, c)) = \gcd(\text{lcm}(a, b), \text{lcm}(a, c))$ .

For non-negative integers  $a$  and  $b$ , where  $a$  and  $b$  are not both zero,  $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a, b)} - 1$

$$\gcd(a, b) = \sum_{k|a \text{ and } k|b} \phi(k)$$

$$\sum_{i=1}^n [\gcd(i, n) = k] = \phi\left(\frac{n}{k}\right)$$

$$\sum_{k=1}^n \gcd(k, n) = \sum_{d|n} d \cdot \phi\left(\frac{n}{d}\right)$$

$$\sum_{k=1}^n x^{\gcd(k, n)} = \sum_{d|n} x^d \cdot \phi\left(\frac{n}{d}\right)$$

$$\sum_{k=1}^n \frac{1}{\gcd(k, n)} = \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{1}{n} \sum_{d|n} d \cdot \phi(d)$$

$$\sum_{k=1}^n \frac{k}{\gcd(k, n)} = \frac{n}{2} \cdot \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$$

$$\sum_{k=1}^n \frac{n}{\gcd(k, n)} = 2 * \sum_{k=1}^n \frac{k}{\gcd(k, n)} - 1, \text{ for } n > 1$$

$$\sum_{i=1}^n \sum_{j=1}^n [\gcd(i, j) = 1] = \sum_{d=1}^n \mu(d) \left\lfloor \frac{n}{d} \right\rfloor^2$$

$$\sum_{i=1}^n \sum_{j=1}^n \gcd(i, j) = \sum_{d=1}^n \phi(d) \left\lfloor \frac{n}{d} \right\rfloor^2$$

$$\sum_{i=1}^n \sum_{j=1}^n i \cdot j [\gcd(i, j) = 1] = \sum_{i=1}^n \phi(i) i^2$$

$$F(n) = \sum_{i=1}^n \sum_{j=1}^n \text{lcm}(i, j) = \sum_{l=1}^n \left( \frac{(1 + \lfloor \frac{n}{l} \rfloor) (\lfloor \frac{n}{l} \rfloor)}{2} \right)^2 \sum_{d|l} \mu(d) l d$$