

- 1 Contest
- 2 Data structures
- 3 Numerical
- 4 Number theory
- 5 Combinatorial
- 6 Graph
- 7 Geometry
- 8 Strings
- 9 Various

## Contest (1)

template.cpp

17 lines

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
#pragma GCC optimize("Ofast,unroll-loops,trapv")
#pragma GCC target("avx,avx2,fma")
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
// cout << fixed << setprecision(25);

int main() {
    cin.tie(0)->sync_with_stdio(0);
    cin.exceptions(cin.failbit);
}
```

## Data structures (2)

OrderStatisticTree.h

**Description:** A set (not multiset!) with support for finding the  $n$ 'th element, and finding the index of an element. To get a map, change `nulltype`.

**Time:**  $\mathcal{O}(\log N)$

782797, 16 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
```

```
1 template<class T>
2 using Tree = tree<T, null_type, less<T>, rb_tree_tag,
3     tree_order_statistics_node_update>;
4
5 void example() {
6     Tree<int> t, t2; t.insert(8);
7     auto it = t.insert(10).first;
8     assert(it == t.lower_bound(9));
9     assert(t.order_of_key(10) == 1);
10    assert(t.order_of_key(11) == 2);
11    assert(*t.find_by_order(0) == 8);
12    t.join(t2); // assuming T < T2 or T > T2, merge t2
13                into t
14 }
```

17 HashMap.h

**Description:** Hash map with mostly the same API as `unordered_map`, but  $\sim 3\times$  faster. Uses  $1.5\times$  memory. Initial capacity must be a power of 2 (if provided).

d77092, 7 lines

```
23 #include <bits/extc++.h>
24 // To use most bits rather than just the lowest ones:
25 struct chash { // large odd number for C
26     const uint64_t C = 114e18 * acos(0) | 71;
27     ll operator()(ll x) const { return __builtin_bswap64(
28         x*C); }
29 };
30 __gnu_pbds::gp_hash_table<ll,int,chash> h({}, {}, {}, {}, {
31     1<<16});
```

SegmentTreeTourist.h

**Description:** .

**Time:** .

4d7093, 115 lines

```
class segtree {
public:
    struct node {
        int mn = 0;
        int mx = 0;
        int add = 0;

        void apply(int l, int r, int v) {
            mn += v;
            mx += v;
            add += v;
        }
    };

    node unite(const node &a, const node &b) const {
        node res;
        res.mn = min(a.mn, b.mn);
        res.mx = max(a.mx, b.mx);
        return res;
    }

    inline void push(int x, int l, int r) {
        int y = (l + r) >> 1;
        int z = x + ((y - 1 + 1) << 1);
        if (tree[x].add != 0) {
```

```
        tree[x + 1].apply(l, y, tree[x].add);
        tree[z].apply(y + 1, r, tree[x].add);
        tree[x].add = 0;
    }
}
```

```
inline void pull(int x, int z) {
    tree[x] = unite(tree[x + 1], tree[z]);
}
```

```
int n;
vector<node> tree;
```

```
void build(int x, int l, int r) {
    if (l == r) {
        return;
    }
    int y = (l + r) >> 1;
    int z = x + ((y - 1 + 1) << 1);
    build(x + 1, l, y);
    build(z, y + 1, r);
    pull(x, z);
}
```

```
template <typename M>
void build(int x, int l, int r, const vector<M> &v) {
    if (l == r) {
        tree[x].apply(l, r, v[l]);
        return;
    }
    int y = (l + r) >> 1;
    int z = x + ((y - 1 + 1) << 1);
    build(x + 1, l, y, v);
    build(z, y + 1, r, v);
    pull(x, z);
}
```

```
node get(int x, int l, int r, int ll, int rr) {
    if (ll <= l && r <= rr) {
        return tree[x];
    }
    int y = (l + r) >> 1;
    int z = x + ((y - 1 + 1) << 1);
    push(x, l, r);
    node res{};
    if (rr <= y) {
        res = get(x + 1, l, y, ll, rr);
    } else {
        if (ll > y) {
            res = get(z, y + 1, r, ll, rr);
        } else {
            res = unite(get(x + 1, l, y, ll, rr), get(z, y
                + 1, r, ll, rr));
        }
    }
    pull(x, z);
    return res;
}
```

<pre>template &lt;typename... M&gt; void modify(int x, int l, int r, int ll, int rr,     const M&amp;... v) {     if (ll &lt;= l &amp;&amp; r &lt;= rr) {         tree[x].apply(l, r, v...);         return;     }     int y = (l + r) &gt;&gt; 1;     int z = x + ((y - l + 1) &lt;&lt; 1);     push(x, l, r);     if (ll &lt;= y) {         modify(x + 1, l, y, ll, rr, v...);     }     if (rr &gt; y) {         modify(z, y + 1, r, ll, rr, v...);     }     pull(x, z); }  segtree(int _n) : n(_n) {     assert(n &gt; 0);     tree.resize(2 * n - 1);     build(0, 0, n - 1); }  template &lt;typename M&gt; segtree(const vector&lt;M&gt; &amp;v) {     n = v.size();     assert(n &gt; 0);     tree.resize(2 * n - 1);     build(0, 0, n - 1, v); }  };</pre>	34ecf5, 50 lines
<p>LazySegmentTree.h</p> <p><b>Description:</b> Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.</p> <p><b>Usage:</b> Node* tr = new Node(v, 0, sz(v));</p> <p><b>Time:</b> <math>\mathcal{O}(\log N)</math>.</p>	

<pre>int query(int L, int R) {     if (R &lt;= lo    hi &lt;= L) return -inf;     if (L &lt;= lo &amp;&amp; hi &lt;= R) return val;     push();     return max(l-&gt;query(L, R), r-&gt;query(L, R)); }  void set(int L, int R, int x) {     if (R &lt;= lo    hi &lt;= L) return;     if (L &lt;= lo &amp;&amp; hi &lt;= R) mset = val = x, madd = 0;     else {         push(), l-&gt;set(L, R, x), r-&gt;set(L, R, x);         val = max(l-&gt;val, r-&gt;val);     } }  void add(int L, int R, int x) {     if (R &lt;= lo    hi &lt;= L) return;     if (L &lt;= lo &amp;&amp; hi &lt;= R) {         if (mset != inf) mset += x;         else madd += x;         val += x;     }     else {         push(), l-&gt;add(L, R, x), r-&gt;add(L, R, x);         val = max(l-&gt;val, r-&gt;val);     } }  void push() {     if (!l) {         int mid = lo + (hi - lo)/2;         l = new Node(lo, mid); r = new Node(mid, hi);     }     if (mset != inf)         l-&gt;set(lo,hi,mset), r-&gt;set(lo,hi,mset), mset = inf;     else if (madd)         l-&gt;add(lo,hi,madd), r-&gt;add(lo,hi,madd), madd = 0; }  };</pre>	de4ad0, 21 lines
--	------------------

UnionFindRollback.h

**Description:** Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().

**Usage:** int t = uf.time(); ...; uf.rollback(t);

**Time:**  $\mathcal{O}(\log(N))$

<pre>struct RollbackUF {     vi e; vector&lt;pii&gt; st;     RollbackUF(int n) : e(n, -1) {}     int size(int x) { return -e[find(x)]; }     int find(int x) { return e[x] &lt; 0 ? x : find(e[x]); }     int time() { return sz(st); }     void rollback(int t) {         for (int i = time(); i --&gt; t;)             e[st[i].first] = st[i].second;         st.resize(t);     }     bool join(int a, int b) {         a = find(a), b = find(b);</pre>	
---	--

<pre>if (a == b) return false; if (e[a] &gt; e[b]) swap(a, b); st.push_back({a, e[a]}); st.push_back({b, e[b]}); e[a] += e[b]; e[b] = a; return true; } };</pre>	c59ada, 13 lines
<p>SubMatrix.h</p> <p><b>Description:</b> Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).</p> <p><b>Usage:</b> SubMatrix&lt;int&gt; m(matrix);</p> <p>m.sum(0, 0, 2, 2); // top left 4 elements</p> <p><b>Time:</b> <math>\mathcal{O}(N^2 + Q)</math></p>	

Matrix.h

**Description:** Basic operations on square matrices.

**Usage:** Matrix<int, 3> A;

A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};

vector<int> vec = {1,2,3};

vec = (A^N) \* vec;

<pre>template&lt;class T, int N&gt; struct Matrix {     typedef Matrix M;     array&lt;array&lt;T, N&gt;, N&gt; d{};     M operator*(const M&amp; m) const {         M a;         rep(i,0,N) rep(j,0,N)             rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];         return a;     }     vector&lt;T&gt; operator*(const vector&lt;T&gt;&amp; vec) const {         vector&lt;T&gt; ret(N);         rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];         return ret;     }     M operator^(ll p) const {         assert(p &gt;= 0);         M a, b(*this);         rep(i,0,N) a.d[i][i] = 1;         while (p) {             if (p&amp;1) a = a*b;             b = b*b;</pre>	c43c7d, 26 lines
---	------------------

```
p >>= 1;
}
return a;
}
};
```

LineContainer.h

**Description:** Container where you can add lines of the form  $kx+m$ , and query maximum values at points  $x$ . Useful for dynamic programming (“convex hull trick”).

**Time:**  $\mathcal{O}(\log N)$

Sec1c7, 30 lines

```
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

Treap.h

**Description:** A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.

**Time:**  $\mathcal{O}(\log N)$

9556fc, 55 lines

```
struct Node {
    Node *l = 0, *r = 0;
    int val, y, c = 1;
    Node(int val) : val(val), y(rand()) {}
    void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
```

```
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
    if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
    if (!n) return {};
    if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
        auto pa = split(n->l, k);
        n->l = pa.second;
        n->recalc();
        return {pa.first, n};
    } else {
        auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
        n->r = pa.first;
        n->recalc();
        return {n, pa.second};
    }
}

Node* merge(Node* l, Node* r) {
    if (!l) return r;
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        l->recalc();
        return l;
    } else {
        r->l = merge(l, r->l);
        r->recalc();
        return r;
    }
}
```

```
Node* ins(Node* t, Node* n, int pos) {
    auto pa = split(t, pos);
    return merge(merge(pa.first, n), pa.second);
}
```

```
// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
    Node *a, *b, *c;
    tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
    if (k <= l) t = merge(ins(a, b, k), c);
    else t = merge(a, ins(c, b, k - r));
}
```

FenwickTree.h

**Description:** Computes partial sums  $a[0] + a[1] + \dots + a[\text{pos} - 1]$ , and updates single elements  $a[i]$ , taking the difference between the old and new value.

**Time:** Both operations are  $\mathcal{O}(\log N)$ .

e62fac, 22 lines

```
struct FT {
```

```
vector<ll> s;
FT(int n) : s(n) {}
void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
}
ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
}
int lower_bound(ll sum) { // min pos st sum of [0, pos)
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
        if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
            pos += pw, sum -= s[pos-1];
    }
    return pos;
}
};
```

FenwickTree2d.h

**Description:** Computes sums  $a[i,j]$  for all  $i < I, j < J$ , and increases single elements  $a[i,j]$ . Requires that the elements to be updated are known in advance (call `fakeUpdate()` before `init()`).

**Time:**  $\mathcal{O}(\log^2 N)$ . (Use persistent segment trees for  $\mathcal{O}(\log N)$ .)

"FenwickTree.h" 157f07, 22 lines

```
struct FT2 {
    vector<vi> ys; vector<FT> ft;
    FT2(int limx) : ys(limx) {}
    void fakeUpdate(int x, int y) {
        for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
    }
    void init() {
        for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
    }
    int ind(int x, int y) {
        return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
    void update(int x, int y, ll dif) {
        for (; x < sz(ys); x |= x + 1)
            ft[x].update(ind(x, y), dif);
    }
    ll query(int x, int y) {
        ll sum = 0;
        for (; x; x &= x - 1)
            sum += ft[x-1].query(ind(x-1, y));
        return sum;
    }
};
```

RMQ.h

**Description:** Range Minimum Queries on an array. Returns  $\min(V[a], V[a + 1], \dots V[b - 1])$  in constant time.

Usage: RMQ rmq(values);  
rmq.query(inclusive, exclusive);  
Time:  $\mathcal{O}(|V|\log|V|+Q)$

510c32, 16 lines

```
template<class T>
struct RMQ {
    vector<vector<T>> jmp;
    RMQ(const vector<T>& V) : jmp(1, V) {
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
            jmp.emplace_back(sz(V) - pw * 2 + 1);
            rep(j,0,sz(jmp[k]))
                jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
        }
    }
    T query(int a, int b) {
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
};
```

MoQueries.h

Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change step to add/remove the edge (a,c) and remove the initial add call (but keep in).  
Time:  $\mathcal{O}(N\sqrt{Q})$

a12ef4, 49 lines

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer
```

```
vi mo(vector<pii> Q) {
    int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
    vi s(sz(Q)), res = s;
    #define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
    for (int qi : s) {
        pii q = Q[qi];
        while (L > q.first) add(--L, 0);
        while (R < q.second) add(R++, 1);
        while (L < q.first) del(L++, 0);
        while (R > q.second) del(--R, 1);
        res[qi] = calc();
    }
    return res;
}
```

```
vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0){
    int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
```

MoQueries Polynomial PolyRoots PolyInterpolate

```
vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N)
;
add(0, 0), in[0] = 1;
auto dfs = [&](int x, int p, int dep, auto& f) ->
    void {
        par[x] = p;
        L[x] = N;
        if (dep) I[x] = N++;
        for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
        if (!dep) I[x] = N++;
        R[x] = N;
    };
dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
iota(all(s), 0);
sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
for (int qi : s) rep(end,0,2) {
    int &a = pos[end], b = Q[qi][end], i = 0;
    #define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
        else { add(c, end); in[c] = 1; } a = c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
        I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
}
return res;
}
```

## Numerical (3)

### 3.1 Polynomials and recurrences

Polynomial.h

```
struct Poly {
    vector<double> a;
    double operator()(double x) const {
        double val = 0;
        for (int i = sz(a); i--;) (val *= x) += a[i];
        return val;
    }
    void diff() {
        rep(i,1,sz(a)) a[i-1] = i*a[i];
        a.pop_back();
    }
    void divroot(double x0) {
        double b = a.back(), c; a.back() = 0;
        for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0 +b, b=c;
```

```
        a.pop_back();
    }
};
```

PolyRoots.h

Description: Finds the real roots to a polynomial.  
Usage: polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0  
Time:  $\mathcal{O}(n^2\log(1/\epsilon))$

b00bfe, 23 lines

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
    if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
    vector<double> ret;
    Poly der = p;
    der.diff();
    auto dr = polyRoots(der, xmin, xmax);
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
    sort(all(dr));
    rep(i,0,sz(dr)-1) {
        double l = dr[i], h = dr[i+1];
        bool sign = p(l) > 0;
        if (sign ^ (p(h) > 0)) {
            rep(it,0,60) { // while (h - l > 1e-8)
                double m = (l + h) / 2, f = p(m);
                if ((f <= 0) ^ sign) l = m;
                else h = m;
            }
            ret.push_back((l + h) / 2);
        }
    }
    return ret;
}
```

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them:  $p(x) = a[0] * x^0 + \dots + a[n-1] * x^{n-1}$ . For numerical precision, pick  $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$ .  
Time:  $\mathcal{O}(n^2)$

08bf48, 13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
    vd res(n), temp(n);
    rep(k,0,n-1) rep(i,k+1,n)
        y[i] = (y[i] - y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1;
    rep(k,0,n) rep(i,0,n) {
        res[i] += y[k] * temp[i];
        swap(last, temp[i]);
        temp[i] -= last * x[k];
    }
    return res;
}
```

### LinearRecurrence.h

**Description:** Generates the  $k$ 'th term of an  $n$ -order linear recurrence  $S[i] = \sum_j S[i-j-1]tr[j]$ , given  $S[0 \dots \geq n-1]$  and  $tr[0 \dots n-1]$ . Faster than matrix multiplication. Useful together with Berlekamp–Massey.

**Usage:** linearRec({0, 1}, {1, 1}, k) //  $k$ 'th Fibonacci number

**Time:**  $\mathcal{O}(n^2 \log k)$

f4e444, 26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);

    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
        rep(i,0,n+1) rep(j,0,n+1)
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i) rep(j,0,n)
            res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]
                ) % mod;
        res.resize(n + 1);
        return res;
    };

    Poly pol(n + 1), e(pol);
    pol[0] = e[1] = 1;

    for (++k; k; k /= 2) {
        if (k % 2) pol = combine(pol, e);
        e = combine(e, e);
    }

    ll res = 0;
    rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
    return res;
}
```

## 3.2 Matrices

### Determinant.h

**Description:** Calculates determinant of a matrix. Destroys the matrix.

**Time:**  $\mathcal{O}(N^3)$

bd5cec, 15 lines

```
double det(vector<vector<double>>& a) {
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1;
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
        }
    }
    return res;
}
```

### IntDeterminant.h

**Description:** Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

**Time:**  $\mathcal{O}(N^3)$

3313dc, 18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
}
```

### SolveLinear.h

**Description:** Solves  $A * x = b$ . If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in  $A$  and  $b$  is lost.

**Time:**  $\mathcal{O}(n^2m)$

44c9ab, 38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m); iota(all(col), 0);

    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m)
            if ((v = fabs(A[r][c])) > bv)
                br = r, bc = c, bv = v;
        if (bv <= eps) {
            rep(j,i,n) if (fabs(b[j]) > eps) return -1;
            break;
        }
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) swap(A[j][i], A[j][bc]);
        bv = 1/A[i][i];
        rep(j,i+1,n) {
            double fac = A[j][i] * bv;
            b[j] -= fac * b[i];
            rep(k,i+1,m) A[j][k] -= fac*A[i][k];
        }
        rank++;
    }
}
```

```
x.assign(m, 0);
for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
}
return rank; // (multiple solutions if rank < m)
}
```

### SolveLinear2.h

**Description:** To get all uniquely determined values of  $x$  back from SolveLinear, make the following changes:

08e495, 7 lines

```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i];
fail:; }
}
```

### SolveLinearBinary.h

**Description:** Solves  $Ax = b$  over  $\mathbb{F}_2$ . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys  $A$  and  $b$ .

**Time:**  $\mathcal{O}(n^2m)$

fa2d7a, 34 lines

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if(b[j]) return -1;
            break;
        }
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) if (A[j][i] != A[j][bc]) {
            A[j].flip(i); A[j].flip(bc);
        }
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];
            A[j] ^= A[i];
        }
        rank++;
    }

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1;
    }
}
```

```
    rep(j,0,i) b[j] ^= A[j][i];
}
return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h

**Description:** Invert matrix  $A$ . Returns rank; result is stored in  $A$  unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set  $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$  where  $A^{-1}$  starts as the inverse of  $A \pmod{p}$ , and  $k$  is doubled in each step.  
**Time:**  $\mathcal{O}(n^3)$

ebfff6, 35 lines

```
int matInv(vector<vector<double>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) {
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n)
            if (fabs(A[j][k]) > fabs(A[r][c]))
                r = j, c = k;
        if (fabs(A[r][c]) < 1e-12) return i;
        A[i].swap(A[r]); tmp[i].swap(tmp[r]);
        rep(j,0,n)
            swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]
                );
        swap(col[i], col[c]);
        double v = A[i][i];
        rep(j,i+1,n) {
            double f = A[j][i] / v;
            A[j][i] = 0;
            rep(k,i+1,n) A[j][k] -= f*A[i][k];
            rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
        }
        rep(j,i+1,n) A[i][j] /= v;
        rep(j,0,n) tmp[i][j] /= v;
        A[i][i] = 1;
    }

    for (int i = n-1; i > 0; --i) rep(j,0,i) {
        double v = A[j][i];
        rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
    }

    rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
    return n;
}
```

**Description:**  $\text{fft}(a)$  computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2. Useful for convolution:  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by  $n$ , reverse(start+1, end), FFT back. Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs). Otherwise, use NTT/FFTMod.  
**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1s$  for  $N = 2^{22}$ )

00ced6, 35 lines

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1); // (^ 10% faster if
        double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i
            /2];
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-
                rolled)
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
    vd conv(const vd& a, const vd& b) {
        if (a.empty() || b.empty()) return {};
        int res(sz(a) + sz(b) - 1);
        vd L = 32 - __builtin_clz(sz(res)), n = 1 << L;
        vector<C> in(n), out(n);
        copy(all(a), begin(in));
        rep(i,0,sz(b)) in[i].imag(b[i]);
        fft(in);
        for (C& x : in) x *= x;
        rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
        fft(out);
        rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
        return res;
    }
}
```

FastFourierTransformMod.h

**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as  $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$  (in practice  $10^{16}$  or higher). Inputs must be in  $[0, \text{mod})$ .  
**Time:**  $\mathcal{O}(N \log N)$ , where  $N = |A| + |B|$  (twice as slow as NTT or FFT)

"FastFourierTransform.h"b82773, 22 lines

```
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
```

```
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt
        (M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] %
        cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] %
        cut);
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) /
            1i;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {
        ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])
            +.5);
        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5)
            ;
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

NumberTheoreticTransform.h

**Description:**  $\text{ntt}(a)$  computes  $\hat{f}(k) = \sum_x a[x]g^{xk}$  for all  $k$ , where  $g = \text{root}^{(\text{mod}-1)/N}$ .  $N$  must be a power of 2. Useful for convolution modulo specific nice primes of the form  $2^a b + 1$ , where the convolution result has size at most  $2^a$ . For arbitrary modulo, see FFTMod.  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For manual convolution: NTT the inputs, multiply pointwise, divide by  $n$ , reverse(start+1, end), NTT back. Inputs must be in  $[0, \text{mod})$ .  
**Time:**  $\mathcal{O}(N \log N)$

"../number-theory/ModPow.h"ced03d, 33 lines

```
const ll mod = (119 << 23) + 1, root = 62; // =
    998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26,
    479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i
                + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
```

```
        ai += (ai + z >= mod ? z - mod : z);
    }
}

vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
        n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod
        * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

FastSubsetTransform.h

**Description:** Transform to a basis with fast convolutions of the form  $c[z] = \sum_{z=x\oplus y} a[x] \cdot b[y]$ , where  $\oplus$  is one of AND, OR, XOR. The size of  $a$  must be a power of two.

**Time:**  $\mathcal{O}(N \log N)$

```
void FST(vi& a, bool inv) {
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j,i,i+
            step) {
                int &u = a[j], &v = a[j + step]; tie(u, v) =
                    inv ? pii(v - u, u) : pii(v, u + v); // AND
                    inv ? pii(v, u - v) : pii(u + v, u); // OR
                    pii(u + v, u - v); // XOR
            }
        }
    if (inv) for (int& x : a) x /= sz(a); // XOR only
}

vi conv(vi a, vi b) {
    FST(a, 0); FST(b, 0);
    rep(i,0,sz(a)) a[i] *= b[i];
    FST(a, 1); return a;
}
```

# Number theory (4)

## 4.1 Modular arithmetic

ModularArithmetic.h

**Description:** Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

```
"euclid.h"
const ll mod = 17; // change to something else
struct Mod {
    ll x;
    Mod(ll xx) : x(xx) {}
    Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
    Mod operator-(Mod b) { return Mod((x - b.x + mod) %
        mod); }
```

```
Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
Mod operator/(Mod b) { return *this * invert(b); }
Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) % mod);
}
Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
}
};
```

ModInverse.h

**Description:** Pre-computation of modular inverses. Assumes LIM ≤ mod and that mod is a prime.

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] %
    mod;
```

ModPow.h

**const** ll mod = 1000000007; // faster if const

```
ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;
    return ans;
}
```

ModLog.h

**Description:** Returns the smallest  $x > 0$  s.t.  $a^x = b \pmod m$ , or  $-1$  if no such  $x$  exists. modLog(a,1,m) can be used to calculate the order of  $a$ .

**Time:**  $\mathcal{O}(\sqrt{m})$

```
ll modLog(ll a, ll b, ll m) {
    ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
    unordered_map<ll, ll> A;
    while (j <= n && (e = f = e * a % m) != b % m)
        A[e * b % m] = j++;
    if (e == b % m) return j;
    if (__gcd(m, e) == __gcd(m, b))
        rep(i,2,n+2) if (A.count(e = e * f % m))
            return n * i - A[e];
    return -1;
}
```

ModSum.h

**Description:** Sums of mod’ed arithmetic progressions.

$\text{modsum}(\text{to}, c, k, m) = \sum_{i=0}^{\text{to}-1} (ki + c) \% m$ . divsum is similar but for floored division.

**Time:**  $\log(m)$ , with a large constant.

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
```

```
ull divsum(ull to, ull c, ull k, ull m) {
    ull res = k / m * sumsq(to) + c / m * to;
    k %= m; c %= m;
    if (!k) return res;
    ull to2 = (to * k + c) / m;
    return res + (to - 1) * to2 - divsum(to2, m-1 - c, m,
        k);
}
```

```
ll modsum(ull to, ll c, ll k, ll m) {
    c = ((c % m) + m) % m;
    k = ((k % m) + m) % m;
    return to * c + k * sumsq(to) - m * divsum(to, c, k,
        m);
}
```

ModMulLL.h

**Description:** Calculate  $a \cdot b \pmod c$  (or  $a^b \pmod c$ ) for  $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$ .

**Time:**  $\mathcal{O}(1)$  for modmul,  $\mathcal{O}(\log b)$  for modpow

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
```

ModSqrt.h

**Description:** Tonelli-Shanks algorithm for modular square roots. Finds  $x$  s.t.  $x^2 = a \pmod p$  ( $-x$  gives the other solution).

**Time:**  $\mathcal{O}(\log^2 p)$  worst case,  $\mathcal{O}(\log p)$  for most  $p$

```
"ModPow.h"
ll sqrt(ll a, ll p) {
    a %= p; if (a < 0) a += p;
    if (a == 0) return 0;
    assert(modpow(a, (p-1)/2, p) == 1); // else no
        solution
    if (p % 4 == 3) return modpow(a, (p+1)/4, p);
    // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8
        == 5
    ll s = p - 1, n = 2;
    int r = 0, m;
    while (s % 2 == 0)
        ++r, s /= 2;
    while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
    ll x = modpow(a, (s + 1) / 2, p);
    ll b = modpow(a, s, p), g = modpow(n, s, p);
    for (; r = m) {
        ll t = b;
        for (m = 0; m < r && t != 1; ++m)
            t = t * t % p;
```

```

    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
}
}
```

## 4.2 Primality

### FastEratosthenes.h

**Description:** Prime sieve for generating all primes smaller than LIM.  
**Time:** LIM=1e9  $\approx$  1.5s

```

6b2912, 20 lines
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
    const int S = (int)round(sqrt(LIM)), R = LIM / 2;
    vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)
        *1.1));
    vector<pii> cp;
    for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
        cp.push_back({i, i * i / 2});
        for (int j = i * i; j <= S; j += 2 * i) sieve[j] =
            1;
    }
    for (int L = 1; L <= R; L += S) {
        array<bool, S> block{};
        for (auto &[p, idx] : cp)
            for (int i=idx; i < S+L; idx = (i+=p)) block[i-L]
                = 1;
        rep(i,0,min(S, R - L))
            if (!block[i]) pr.push_back((L + i) * 2 + 1);
    }
    for (int i : pr) isPrime[i] = 1;
    return pr;
}
```

### MillerRabin.h

**Description:** Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to  $7 \cdot 10^{18}$ ; for larger numbers, use Python and extend A randomly.

**Time:** 7 times the complexity of  $a^b \bmod c$ .

```

"ModMulLL.h"
60dcd1, 12 lines
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
        1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) { // ^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n-1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
```

### Factor.h

**Description:** Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).  
**Time:**  $\mathcal{O}(n^{1/4})$ , less for numbers with small factors.

```

"ModMulLL.h", "MillerRabin.h"
a33cf6, 18 lines
ull pollard(ull n) {
    auto f = [n](ull x) { return modmul(x, x, n) + 1; };
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd
            = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}
vector<ull> factor(ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), all(r));
    return l;
}
}
```

## 4.3 Divisibility

### euclid.h

**Description:** Finds two integers  $x$  and  $y$ , such that  $ax+by = \gcd(a,b)$ . If you just need gcd, use the built in `__gcd` instead. If  $a$  and  $b$  are coprime, then  $x$  is the inverse of  $a \pmod b$ .

```

33ba8f, 5 lines
ll euclid(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = 1, y = 0, a;
    ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d;
}
```

### CRT.h

**Description:** Chinese Remainder Theorem.

`crt(a, m, b, n)` computes  $x$  such that  $x \equiv a \pmod m$ ,  $x \equiv b \pmod n$ . If  $|a| < m$  and  $|b| < n$ ,  $x$  will obey  $0 \leq x < \text{lcm}(m,n)$ . Assumes  $mn < 2^{62}$ .

**Time:**  $\log(n)$

```

"euclid.h"
04d93a, 7 lines
ll crt(ll a, ll m, ll b, ll n) {
    if (n > m) swap(a, b), swap(m, n);
    ll x, y, g = euclid(m, n, x, y);
    assert((a - b) % g == 0); // else no solution
    x = (b - a) % n * x % n / g * m + a;
    return x < 0 ? x + m*n/g : x;
}
```

### 4.3.1 Bézout’s identity

For  $a \neq, b \neq 0$ , then  $d = \gcd(a,b)$  is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If  $(x,y)$  is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

### phiFunction.h

**Description:** *Euler’s*  $\phi$  function is defined as  $\phi(n) := \#$  of positive integers  $\leq n$  that are coprime with  $n$ .  $\phi(1) = 1$ ,  $p$  prime  $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$ ,  $m,n$  coprime  $\Rightarrow \phi(mn) = \phi(m)\phi(n)$ . If  $n = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$  then  $\phi(n) = (p_1-1)p_1^{k_1-1} \dots (p_r-1)p_r^{k_r-1}$ .  $\phi(n) = n \cdot \prod_{p|n} (1-1/p)$ .  $\sum_{d|n} \phi(d) = n$ ,  $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$

**Euler’s thm:**  $a,n$  coprime  $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$ .

**Fermat’s little thm:**  $p$  prime  $\Rightarrow a^{p-1} \equiv 1 \pmod p \ \forall a$ .

```

cf7d6d, 8 lines
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
    rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
    for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
        for (int j = i; j < LIM; j += i) phi[j] -= phi[j] /
            i;
}
```

## 4.4 Fractions

### ContinuedFractions.h

**Description:** Given  $N$  and a real number  $x \geq 0$ , finds the closest rational approximation  $p/q$  with  $p,q \leq N$ . It will obey  $|p/q-x| \leq 1/qN$ . For consecutive convergents,  $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$ .  $(p_k/q_k)$  alternates between  $> x$  and  $< x$ . If  $x$  is rational,  $y$  eventually becomes  $\infty$ ; if  $x$  is the root of a degree 2 polynomial the  $a$ ’s eventually become cyclic.

**Time:**  $\mathcal{O}(\log N)$

```

dd6c5e, 21 lines
typedef double d; // for N ~ 1e7; long double for N ~ 1
e9
pair<ll, ll> approximate(d x, ll N) {
    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y
        = x;
    for (;;) {
        ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q :
            inf),
            a = (ll)floor(y), b = min(a, lim),
            NP = b*P + LP, NQ = b*Q + LQ;
        if (a > b) {
            // If b > a/2, we have a semi-convergent that
                gives us a
            // better approximation; if b = a/2, we *may*
                have one.
        }
```



```
// Return {P, Q} here for a more canonical
approximation.
return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (
d)Q)) ?
    make_pair(NP, NQ) : make_pair(P, Q);
}
if (abs(y = 1/(y - (d)a)) > 3*N) {
    return {NP, NQ};
}
LP = P; P = NP;
LQ = Q; Q = NQ;
}
}
```

FracBinarySearch.h

**Description:** Given  $f$  and  $N$ , finds the smallest fraction  $p/q \in [0,1]$  such that  $f(p/q)$  is true, and  $p,q \leq N$ . You may want to throw an exception from  $f$  if it finds an exact solution, in which case  $N$  can be removed.

**Usage:**   fracBS([](Frac f) { return f.p>=3\*f.q; }, 10);  
// {1,3}

**Time:**  $\mathcal{O}(\log(N))$

```
struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search
                             (0, N]
    if (f(lo)) return lo;
    assert(f(hi));
    while (A || B) {
        ll adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >= si) {
            adv += step;
            Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid)
                ) {
                adv -= step; si = 2;
            }
        }
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi);
        A = B; B = !!adv;
    }
    return dir ? hi : lo;
}
```

## 4.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \; b = k \cdot (2mn), \; c = k \cdot (m^2 + n^2),$$

with  $m > n > 0$ ,  $k > 0$ ,  $m \perp n$ , and either  $m$  or  $n$  even.

4.6 Primes

$p = 962592769$  is such that  $2^{21} \mid p - 1$ , which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power  $p^a$ , except for  $p = 2, a > 2$ , and there are  $\phi(\phi(p^a))$  many. For  $p = 2, a > 2$ , the group  $\mathbb{Z}_{2^a}^\times$  is instead isomorphic to  $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$ .

## 4.7 Estimates

$$\sum_{d|n} d = O(n \log \log n).$$

The number of divisors of  $n$  is at most around 100 for  $n < 5e4$ , 500 for  $n < 1e7$ , 2000 for  $n < 1e10$ , 200 000 for  $n < 1e19$ .

## 4.8 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d)g(n/d)$$

Other useful formulas/forms:

$$\sum_{d|n} \mu(d) = [n = 1] \text{ (very useful)}$$

$$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n)g(d)$$

$$g(n) = \sum_{1 \leq m \leq n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \leq m \leq n} \mu(m)g(\lfloor \frac{n}{m} \rfloor)$$

# Combinatorial (5)

## 5.1 Permutations

### 5.1.1 Factorial

$n$	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
$n$	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
$n$	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

```
IntPerm.h
Description:   Permutation -> integer conversion.   (Not order
preserving.) Integer -> permutation can use a lookup table.
Time:  $\mathcal{O}(n)$ 
044568, 6 lines

int permToInt(vi& v) {
    int use = 0, i = 0, r = 0;
    for(int x:v) r = r * ++i + __builtin_popcount(use &
        -(1<<x)),
        use |= 1 << x;
        // (note: minus,
        not ~!)
    return r;
}
```

### 5.1.2 Cycles

Let  $g_S(n)$  be the number of  $n$ -permutations whose cycle lengths all belong to the set  $S$ . Then

$$\sum_{n=0}^\infty g_S(n) \frac{x^n}{n!} = \exp \left( \sum_{n \in S} \frac{x^n}{n} \right)$$

### 5.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rfloor$$

### 5.1.4 Burnside’s lemma

Given a group  $G$  of symmetries and a set  $X$ , the number of elements of  $X$  up to symmetry equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where  $X^g$  are the elements fixed by  $g$  ( $g.x = x$ ).

If  $f(n)$  counts “configurations” (of some sort) of length  $n$ , we can ignore rotational symmetry using  $G = \mathbb{Z}_n$  to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n} \sum_{k|n} f(k)\phi(n/k).$$

## 5.2 Partitions and subsets

### 5.2.1 Partition function

Number of ways of writing  $n$  as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1, \; p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k - 1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

$n$	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	$\sim 2\text{e}5$	$\sim 2\text{e}8$

5.2.2 Lucas’ Theorem

Let  $n, m$  be non-negative integers and  $p$  a prime. Write  $n = n_kp^k + \dots + n_1p + n_0$  and  $m = m_kp^k + \dots + m_1p + m_0$ . Then  $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$ .

5.2.3 Binomials

multinomial.h

**Description:** Computes  $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1!k_2!\dots k_n!}$ . a0a312, 6 lines

```
ll multinomial(vi& v) {
    ll c = 1, m = v.empty() ? 1 : v[0];
    rep(i, 1, sz(v)) rep(j, 0, v[i])
        c = c * ++m / (j+1);
    return c;
}
```

5.3 General purpose numbers

5.3.1 Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t - 1}$  (FFT-able).  
 $B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$

Sums of powers:

$$\sum_{i=1}^n n^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m)$$
$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

5.3.2 Stirling numbers of the first kind

Number of permutations on  $n$  items with  $k$  cycles.

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

$$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$$
$$c(n, 2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$$

5.3.3 Eulerian numbers

Number of permutations  $\pi \in S_n$  in which exactly  $k$  elements are greater than the previous element.  $k$   $j$ :s s.t.  $\pi(j) > \pi(j + 1)$ ,  $k + 1$   $j$ :s s.t.  $\pi(j) \geq j$ ,  $k$   $j$ :s s.t.  $\pi(j) > j$ .

$$E(n, k) = (n - k)E(n - 1, k - 1) + (k + 1)E(n - 1, k)$$

$$E(n, 0) = E(n, n - 1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

5.3.4 Stirling numbers of the second kind

Partitions of  $n$  distinct elements into exactly  $k$  groups.

$$S(n, k) = S(n - 1, k - 1) + kS(n - 1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

5.3.5 Bell numbers

Total number of partitions of  $n$  distinct elements.  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$  For  $p$  prime,

$$B(p^m + n) \equiv mB(n) + B(n + 1) \pmod{p}$$

5.3.6 Labeled unrooted trees

# on  $n$  vertices:  $n^{n-2}$   
# on  $k$  existing trees of size  $n_i$ :  $n_1n_2 \dots n_kn^{k-2}$   
# with degrees  $d_i$ :  $(n - 2)! / ((d_1 - 1)! \dots (d_n - 1)!)$

5.3.7 Catalan numbers

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2}C_n, \quad C_{n+1} = \sum C_iC_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

- sub-diagonal monotone paths in an  $n \times n$  grid.
- strings with  $n$  pairs of parenthesis, correctly nested.
- binary trees with with  $n + 1$  leaves (0 or 2 children).

- ordered trees with  $n + 1$  vertices.
- ways a convex polygon with  $n + 2$  sides can be cut into triangles by connecting vertices with straight lines.
- permutations of  $[n]$  with no 3-term increasing subseq.

Graph (6)

6.1 Fundamentals

BellmanFord.h

**Description:** Calculates shortest paths from  $s$  in a graph that might have negative edge weights. Unreachable nodes get `dist = inf`; nodes reachable through negative-weight cycles get `dist = -inf`. Assumes  $V^2 \max |w_i| < \sim 2^{63}$ .

**Time:**  $\mathcal{O}(VE)$  830a8f, 23 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; } };
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds,
    int s) {
    nodes[s].dist = 0;
    sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

    int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
    rep(i, 0, lim) for (Ed ed : eds) {
        Node cur = nodes[ed.a], &dest = nodes[ed.b];
        if (abs(cur.dist) == inf) continue;
        ll d = cur.dist + ed.w;
        if (d < dest.dist) {
            dest.prev = ed.a;
            dest.dist = (i < lim-1 ? d : -inf);
        }
    }
    rep(i, 0, lim) for (Ed e : eds) {
        if (nodes[e.a].dist == -inf)
            nodes[e.b].dist = -inf;
    }
}
```

FloydWarshall.h

**Description:** Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix  $m$ , where  $m[i][j] = \text{inf}$  if  $i$  and  $j$  are not adjacent. As output,  $m[i][j]$  is set to the shortest distance between  $i$  and  $j$ , `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.

**Time:**  $\mathcal{O}(N^3)$  531245, 12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
    int n = sz(m);
    rep(i, 0, n) m[i][i] = min(m[i][i], 0LL);
```

```
rep(k,0,n) rep(i,0,n) rep(j,0,n)
  if (m[i][k] != inf && m[k][j] != inf) {
    auto newDist = max(m[i][k] + m[k][j], -inf);
    m[i][j] = min(m[i][j], newDist);
  }
rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
  if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -
    inf;
}
```

TopoSort.h

**Description:** Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than  $n$  – nodes reachable from cycles will not be returned.

**Time:**  $\mathcal{O}(|V| + |E|)$

66a137, 14 lines

```
vi topoSort(const vector<vi>& gr) {
  vi indeg(sz(gr)), ret;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  queue<int> q; // use priority_queue for lexic.
    largest ans.
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
  while (!q.empty()) {
    int i = q.front(); // top() for priority queue
    ret.push_back(i);
    q.pop();
    for (int x : gr[i])
      if (--indeg[x] == 0) q.push(x);
  }
  return ret;
}
```

6.2 Network flow

PushRelabel.h

**Description:** Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

**Time:**  $\mathcal{O}\left(V^2\sqrt{E}\right)$

0ae1d4, 48 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(
    n) {}

  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }
}
```

```
void addFlow(Edge& e, ll f) {
  Edge &back = g[e.dest][e.back];
  if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.
    dest);
  e.f += f; e.c -= f; ec[e.dest] += f;
  back.f -= f; back.c += f; ec[back.dest] -= f;
}
ll calc(int s, int t) {
  int v = sz(g); H[s] = v; ec[t] = 1;
  vi co(2*v); co[0] = v-1;
  rep(i,0,v) cur[i] = g[i].data();
  for (Edge& e : g[s]) addFlow(e, e.c);

  for (int hi = 0;;) {
    while (hs[hi].empty()) if (!hi--) return -ec[s];
    int u = hs[hi].back(); hs[hi].pop_back();
    while (ec[u] > 0) // discharge u
      if (cur[u] == g[u].data() + sz(g[u])) {
        H[u] = 1e9;
        for (Edge& e : g[u]) if (e.c && H[u] > H[e.
          dest]+1)
          H[u] = H[e.dest]+1, cur[u] = &e;
        if (++co[H[u]], !--co[hi] && hi < v)
          rep(i,0,v) if (hi < H[i] && H[i] < v)
            --co[H[i]], H[i] = v + 1;
        hi = H[u];
      } else if (cur[u]->c && H[u] == H[cur[u]->dest
        ]+1)
        addFlow(*cur[u], min(ec[u], cur[u]->c));
      else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlow.h

**Description:** Min-cost max-flow.  $\text{cap}[i][j] \neq \text{cap}[j][i]$  is allowed; double edges are not. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.

**Time:** Approximately  $\mathcal{O}(E^2)$

fe85cc, 81 lines

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

const ll INF = numeric_limits<ll>::max() / 4;
typedef vector<ll> VL;

struct MCMF {
  int N;
  vector<vi> ed, red;
  vector<VL> cap, flow, cost;
  vi seen;
  VL dist, pi;
  vector<pii> par;

  MCMF(int N) :
```

```
  N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost
    (cap),
  seen(N), dist(N), pi(N), par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
  }

  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({0, s});

    auto relax = [&](int i, ll cap, ll cost, int dir) {
      ll val = di - pi[i] + cost;
      if (cap && val < dist[i]) {
        dist[i] = val;
        par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.push({-dist[i]
          }, i});
        else q.modify(its[i], {-dist[i], i});
      }
    };

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (int i : ed[s]) if (!seen[i])
        relax(i, cap[s][i] - flow[s][i], cost[s][i], 1)
          ;
      for (int i : red[s]) if (!seen[i])
        relax(i, flow[i][s], -cost[i][s], 0);
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x
        = p)
        fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[
          x][p]);
      totflow += fl;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x
        = p)
        if (r) flow[p][x] += fl;
        else flow[x][p] -= fl;
    }
  }
}
```

```
    rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
}

// If some costs can be negative, call this before maxflow:
void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--){
        rep(i,0,N) if (pi[i] != INF)
            for (int to : ed[i]) if (cap[i][to])
                if ((v = pi[i] + cost[i][to]) < pi[to])
                    pi[to] = v, ch = 1;
        assert(it >= 0); // negative cost cycle
    }
};
```

EdmondsKarp.h

**Description:** Flow algorithm with guaranteed complexity  $O(VE^2)$ . To get edge flow values, compare capacities before and after, and take the positive values only.

```
template<class T> T edmondsKarp(vector<unordered_map<
    int, T>>& graph, int source, int sink) {
    assert(source != sink);
    T flow = 0;
    vi par(sz(graph)), q = par;

    for (;;) {
        fill(all(par), -1);
        par[source] = 0;
        int ptr = 1;
        q[0] = source;

        rep(i,0,ptr) {
            int x = q[i];
            for (auto e : graph[x]) {
                if (par[e.first] == -1 && e.second > 0) {
                    par[e.first] = x;
                    q[ptr++] = e.first;
                    if (e.first == sink) goto out;
                }
            }
        }
        return flow;
    out:
        T inc = numeric_limits<T>::max();
        for (int y = sink; y != source; y = par[y])
            inc = min(inc, graph[par[y]][y]);

        flow += inc;
        for (int y = sink; y != source; y = par[y]) {
            int p = par[y];
            if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
            graph[y][p] += inc;
        }
    }
};
```

```
    }
}
```

MinCut.h

**Description:** After running max-flow, the left side of a min-cut from  $s$  to  $t$  is given by all vertices reachable from  $s$ , only traversing edges with positive residual capacity.

GlobalMinCut.h

**Description:** Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.

**Time:**  $\mathcal{O}(V^3)$

```
pair<int, vi> globalMinCut(vector<vi> mat) {
    pair<int, vi> best = {INT_MAX, {}};
    int n = sz(mat);
    vector<vi> co(n);
    rep(i,0,n) co[i] = {i};
    rep(ph,1,n) {
        vi w = mat[0];
        size_t s = 0, t = 0;
        rep(it,0,n-ph) { //  $\mathcal{O}(V^2) \rightarrow \mathcal{O}(E \log V)$  with prio. queue
            w[t] = INT_MIN;
            s = t, t = max_element(all(w)) - w.begin();
            rep(i,0,n) w[i] += mat[t][i];
        }
        best = min(best, {w[t] - mat[t][t], co[t]});
        co[s].insert(co[s].end(), all(co[t]));
        rep(i,0,n) mat[s][i] += mat[t][i];
        rep(i,0,n) mat[i][s] = mat[s][i];
        mat[0][t] = INT_MIN;
    }
    return best;
}
```

GomoryHu.h

**Description:** Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.

**Time:**  $\mathcal{O}(V)$  Flow Computations

```
"PushRelabel.h"
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i,1,N) {
        PushRelabel D(N); // Dinic also works
        for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
        ;
        tree.push_back({i, par[i], D.calc(i, par[i])});
        rep(j,i+1,N)
            if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
    }
    return tree;
}
```

```
}
```

## 6.3 Matching

hopcroftKarp.h

**Description:** Fast bipartite matching algorithm. Graph  $g$  should be a list of neighbors of the left partition, and  $btoa$  should be a vector full of -1's of the same size as the right partition. Returns the size of the matching.  $btoa[i]$  will be the match for vertex  $i$  on the right side, or -1 if it's not matched.

**Usage:** vi btoa(m, -1); hopcroftKarp(g, btoa);

**Time:**  $\mathcal{O}(\sqrt{VE})$

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A,
    vi& B) {
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0);
        fill(all(B), 0);
        cur.clear();
        for (int a : btoa) if(a != -1) A[a] = -1;
        rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
        for (int lay = 1;; lay++) {
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay;
                    islast = 1;
                }
                else if (btoa[b] != a && !B[b]) {
                    B[b] = lay;
                    next.push_back(btoa[b]);
                }
            }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay;
            cur.swap(next);
        }
        rep(a,0,sz(g))
            res += dfs(a, 0, g, btoa, A, B);
    }
}
```

### DFSMatching.h

**Description:** Simple bipartite matching algorithm. Graph  $g$  should be a list of neighbors of the left partition, and  $btoa$  should be a vector full of -1's of the same size as the right partition. Returns the size of the matching.  $btoa[i]$  will be the match for vertex  $i$  on the right side, or  $-1$  if it's not matched.

**Usage:** vi btoa(m, -1); dfsMatching(g, btoa);

**Time:**  $\mathcal{O}(VE)$

522b98, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
    if (btoa[j] == -1) return 1;
    vis[j] = 1; int di = btoa[j];
    for (int e : g[di])
        if (!vis[e] && find(e, g, btoa, vis)) {
            btoa[e] = di;
            return 1;
        }
    return 0;
}

int dfsMatching(vector<vi>& g, vi& btoa) {
    vi vis;
    rep(i,0,sz(g)) {
        vis.assign(sz(btoa), 0);
        for (int j : g[i])
            if (find(j, g, btoa, vis)) {
                btoa[j] = i;
                break;
            }
    }
    return sz(btoa) - (int)count(all(btoa), -1);
}
```

### MinimumVertexCover.h

**Description:** Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

"DFSMatching.h"da4196, 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
    vi match(m, -1);
    int res = dfsMatching(g, match);
    vector<bool> lfound(n, true), seen(m);
    for (int it : match) if (it != -1) lfound[it] = false;

    vi q, cover;
    rep(i,0,n) if (lfound[i]) q.push_back(i);
    while (!q.empty()) {
        int i = q.back(); q.pop_back();
        lfound[i] = 1;
        for (int e : g[i]) if (!seen[e] && match[e] != -1)
            {
                seen[e] = true;
                q.push_back(match[e]);
            }
    }
    rep(i,0,n) if (!lfound[i]) cover.push_back(i);
    rep(i,0,m) if (seen[i]) cover.push_back(n+i);
    assert(sz(cover) == res);
    return cover;
}
```

### WeightedMatching.h

**Description:** Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes  $cost[N][M]$ , where  $cost[i][j] = cost$  for  $L[i]$  to be matched with  $R[j]$  and returns (min cost, match), where  $L[i]$  is matched with  $R[match[i]]$ . Negate costs for max cost. Requires  $N \leq M$ .

**Time:**  $\mathcal{O}(N^2M)$

1e0fe9, 31 lines

```
pair<int, vi> hungarian(const vector<vi> &a) {
    if (a.empty()) return {0, {}};
    int n = sz(a) + 1, m = sz(a[0]) + 1;
    vi u(n), v(m), p(m), ans(n - 1);
    rep(i,1,n) {
        p[0] = i;
        int j0 = 0; // add "dummy" worker 0
        vi dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true;
            int i0 = p[j0], j1, delta = INT_MAX;
            rep(j,1,m) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j;
            }
            rep(j,0,m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                else dist[j] -= delta;
            }
            j0 = j1;
        } while (p[j0]);
        while (j0) { // update alternating path
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1;
        }
    }
    rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans}; // min cost
}
```

### GeneralMatching.h

**Description:** Matching for general graphs. Fails with probability  $N/mod$ .

**Time:**  $\mathcal{O}(N^3)$

"../numerical/MatrixInverse-mod.h"cb1912, 40 lines

```
vector<pii> generalMatching(int N, vector<pii>& ed) {
    vector<vector<ll>> mat(N, vector<ll>(N)), A;
    for (pii pa : ed) {
        int a = pa.first, b = pa.second, r = rand() % mod;
        mat[a][b] = r, mat[b][a] = (mod - r) % mod;
    }

    int r = matInv(A = mat), M = 2*N - r, fi, fj;
    assert(r % 2 == 0);
}
```

```
if (M != N) do {
    mat.resize(M, vector<ll>(M));
    rep(i,0,N) {
        mat[i].resize(M);
        rep(j,N,M) {
            int r = rand() % mod;
            mat[i][j] = r, mat[j][i] = (mod - r) % mod;
        }
    }
} while (matInv(A = mat) != M);
```

```
vi has(M, 1); vector<pii> ret;
rep(it,0,M/2) {
    rep(i,0,M) if (has[i])
        rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
            fi = i; fj = j; goto done;
        } assert(0); done:
    if (fj < N) ret.emplace_back(fi, fj);
    has[fi] = has[fj] = 0;
    rep(sw,0,2) {
        ll a = modpow(A[fi][fj], mod-2);
        rep(i,0,M) if (has[i] && A[i][fj]) {
            ll b = A[i][fj] * a % mod;
            rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
        }
        swap(fi,fj);
    }
}
return ret;
}
```

### kuhn.h

**Description:** Flow algorithm with complexity  $\mathcal{O}(VE \log U)$  where  $U = \max|cap|$ .  $\mathcal{O}(\min(E^{1/2}, V^{2/3})E)$  if  $U = 1$ ;  $\mathcal{O}(\sqrt{VE})$  for bipartite matching.

30d06c, 31 lines

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    //... reading the graph ...
}
```

```
mt.assign(k, -1);
for (int v = 0; v < n; ++v) {
    used.assign(n, false);
    try_kuhn(v);
}

for (int i = 0; i < k; ++i)
    if (mt[i] != -1)
        printf("%d %d\n", mt[i] + 1, i + 1);
}
```

## 6.4 DFS algorithms

**SCC.h**  
**Description:** Finds strongly connected components in a directed graph. If vertices  $u, v$  belong to the same component, we can reach  $u$  from  $v$  and vice versa.  
**Usage:** `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.  
**Time:**  $\mathcal{O}(E + V)$

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x; z.push_back(j);
    for (auto e : g[j]) if (comp[e] < 0)
        low = min(low, val[e] ?: dfs(e,g,f));

    if (low == val[j]) {
        do {
            x = z.back(); z.pop_back();
            comp[x] = ncomps;
            cont.push_back(x);
        } while (x != j);
        f(cont); cont.clear();
        ncomps++;
    }
    return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
    int n = sz(g);
    val.assign(n, 0); comp.assign(n, -1);
    Time = ncomps = 0;
    rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

**BiconnectedComponents.h**  
**Description:** Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.

```
Usage: int eid = 0; ed.resize(N);
for each edge (a,b) {
    ed[a].emplace_back(b, eid);
    ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
Time: O(E + V)

vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, e, y, top = me;
    for (auto pa : ed[at]) if (pa.second != par) {
        tie(y, e) = pa;
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e);
        } else {
            int si = sz(st);
            int up = dfs(y, e, f);
            top = min(top, up);
            if (up == me) {
                st.push_back(e);
                f(vi(st.begin() + si, st.end()));
                st.resize(si);
            }
            else if (up < me) st.push_back(e);
            else { /* e is a bridge */ }
        }
    }
    return top;
}

template<class F>
void bicomps(F f) {
    num.assign(sz(ed), 0);
    rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

**2sat.h**  
**Description:** Calculates a valid assignment to boolean variables  $a, b, c, \dots$  to a 2-SAT problem, so that an expression of the type  $(a \vee b) \wedge (\neg a \vee c) \wedge (d \vee \neg b) \wedge \dots$  becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ( $\sim x$ ).  
**Usage:** `TwoSat ts(number of boolean variables);`  
`ts.either(0, ~3);` // Var 0 is true or var 3 is false  
`ts.setValue(2);` // Var 2 is true  
`ts.atMostOne({0,~1,2});` //  $\leq 1$  of vars 0,  $\sim 1$  and 2 are true  
`ts.solve();` // Returns true iff it is solvable  
`ts.values[0..N-1]` holds the assigned values to the vars  
**Time:**  $\mathcal{O}(N + E)$ , where  $N$  is the number of boolean variables, and  $E$  is the number of clauses.

```
struct TwoSat {
    int N;
    vector<vi> gr;
```

```
vi values; // 0 = false, 1 = true

TwoSat(int n = 0) : N(n), gr(2*n) {}

int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
}

void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
}

void setValue(int x) { either(x, x); }

void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
        int next = addVar();
        either(cur, ~li[i]);
        either(cur, next);
        either(~li[i], next);
        cur = ~next;
    }
    either(cur, ~li[1]);
}

vi val, comp, z; int time = 0;
int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
        low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back(); z.pop_back();
        comp[x] = low;
        if (values[x>>1] == -1)
            values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
}

bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
}
};
```

**Description:** Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret. **Time:**  $\mathcal{O}(V + E)$

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Eulerian paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
        if (it == end){ ret.push_back(x); s.pop_back(); continue; }
        tie(y, e) = gr[x][it++];
        if (!eu[e]) {
            D[x]--, D[y]++;
            eu[e] = 1; s.push_back(y);
        }
    }
    for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
    return {ret.rbegin(), ret.rend()};
}
```

## 6.5 Coloring

### EdgeColoring.h

**Description:** Given a simple, undirected graph with max degree  $D$ , computes a  $(D+1)$ -coloring of the edges such that no neighboring edges share a color. ( $D$ -coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.) **Time:**  $\mathcal{O}(NM)$

e210e2, 31 lines

```
vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
        }
    }
}
```

```
        adj[right][e] = -1;
        free[right] = e;
    }
    adj[u][d] = fan[i];
    adj[fan[i]][d] = u;
    for (int y : {fan[0], u, end})
        for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    rep(i,0,sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ ret[i];
    return ret;
}
```

## 6.6 Heuristics

### MaximalCliques.h

**Description:** Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique. **Time:**  $\mathcal{O}(3^{n/3})$ , much faster for sparse graphs

b0d5b1, 12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R= {}) {
    if (!P.any()) { if (!X.any()) f(R); return; }
    auto q = (P | X)._Find_first();
    auto cands = P & ~eds[q];
    rep(i,0,sz(eds)) if (cands[i]) {
        R[i] = 1;
        cliques(eds, f, P & eds[i], X & eds[i], R);
        R[i] = P[i] = 0; X[i] = 1;
    }
}
```

### MaximumClique.h

**Description:** Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph. **Time:** Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

f7c0bc, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv& r) {
        for (auto& v : r) v.d = 0;
        for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d;
    }
}
```

```
        rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
    }
    void expand(vv& R, int lev = 1) {
        S[lev] += S[lev - 1] - old[lev];
        old[lev] = S[lev - 1];
        while (sz(R)) {
            if (sz(q) + R.back().d <= sz(qmax)) return;
            q.push_back(R.back().i);
            vv T;
            for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
            if (sz(T)) {
                if (S[lev]++ / ++pk < limit) init(T);
                int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
                C[1].clear(), C[2].clear();
                for (auto v : T) {
                    int k = 1;
                    auto f = [&](int i) { return e[v.i][i]; };
                    while (any_of(all(C[k]), f)) k++;
                    if (k > mxk) mxk = k, C[mxk + 1].clear();
                    if (k < mnk) T[j++].i = v.i;
                    C[k].push_back(v.i);
                }
                if (j > 0) T[j - 1].d = 0;
                rep(k,mnk,mxk + 1) for (int i : C[k])
                    T[j].i = i, T[j++].d = k;
                expand(T, lev + 1);
            } else if (sz(q) > sz(qmax)) qmax = q;
            q.pop_back(), R.pop_back();
        }
    }
    vi maxClique() { init(V), expand(V); return qmax; }
    Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
        rep(i,0,sz(e)) V.push_back({i});
    }
};
```

### MaximumIndependentSet.h

**Description:** To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

## 6.7 Trees

### BinaryLifting.h

**Description:** Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself. **Time:** construction  $\mathcal{O}(N \log N)$ , queries  $\mathcal{O}(\log N)$

bfce85, 25 lines

```
vector<vi> treeJump(vi& P) {
    int on = 1, d = 1;
    while (on < sz(P)) on *= 2, d++;
    vector<vi> jmp(d, P);
    rep(i,1,d) rep(j,0,sz(P))
        jmp[i][j] = jmp[i-1][jmp[i-1][j]];
```

```
    return jmp;
}

int jmp(vector<vi>& tbl, int nod, int steps){
    rep(i,0,sz(tbl))
        if(steps&(1<<i)) nod = tbl[i][nod];
    return nod;
}

int lca(vector<vi>& tbl, vi& depth, int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    a = jmp(tbl, a, depth[a] - depth[b]);
    if (a == b) return a;
    for (int i = sz(tbl); i--;) {
        int c = tbl[i][a], d = tbl[i][b];
        if (c != d) a = c, b = d;
    }
    return tbl[0][a];
}
```

LCA.h

**Description:** Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.

**Time:**  $\mathcal{O}(N \log N + Q)$

"/data-structures/RMQ.h" 0f62fb, 21 lines

```
struct LCA {
    int T = 0;
    vi time, path, ret;
    RMQ<int> rmq;

    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1),
        ret)) {}
    void dfs(vector<vi>& C, int v, int par) {
        time[v] = T++;
        for (int y : C[v]) if (y != par) {
            path.push_back(v), ret.push_back(time[v]);
            dfs(C, y, v);
        }
    }

    int lca(int a, int b) {
        if (a == b) return a;
        tie(a, b) = minmax(time[a], time[b]);
        return path[rmq.query(a, b)];
    }

    //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(
        a,b)];}

};
```

CompressTree.h

**Description:** Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most  $|S| - 1$ ) pairwise LCA's and compressing edges. Returns a list of (par, orig.index) representing a tree rooted at 0. The root points to itself.

**Time:**  $\mathcal{O}(|S| \log |S|)$

"LCA.h" 9775a0, 21 lines

```
typedef vector<pair<int, int>> vpi;
```

```
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);
    int m = sz(li)-1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i,0,sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i,0,sz(li)-1) {
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
}
```

HLD.h

**Description:** Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most  $\log(n)$  light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS.EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.

**Time:**  $\mathcal{O}((\log N)^2)$

"/data-structures/LazySegmentTree.h" 6f34db, 46 lines

```
template <bool VALS_EDGES> struct HLD {
    int N, tim = 0;
    vector<vi> adj;
    vi par, siz, depth, rt, pos;
    Node *tree;
    HLD(vector<vi> adj_)
        : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
          depth(N),
          rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0);
          dfsHld(0); }
    void dfsSz(int v) {
        if (par[v] != -1) adj[v].erase(find(all(adj[v]),
            par[v]));
        for (int& u : adj[v]) {
            par[u] = v, depth[u] = depth[v] + 1;
            dfsSz(u);
            siz[v] += siz[u];
            if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
        }
    }
    void dfsHld(int v) {
        pos[v] = tim++;
        for (int u : adj[v]) {
            rt[u] = (u == adj[v][0] ? rt[v] : u);
            dfsHld(u);
        }
    }
};
```

```
template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
        if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
        op(pos[rt[v]], pos[v] + 1);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(pos[u] + VALS_EDGES, pos[v] + 1);
}

void modifyPath(int u, int v, int val) {
    process(u, v, [&](int l, int r) { tree->add(l, r,
        val); });
}

int queryPath(int u, int v) { // Modify depending on
    problem
    int res = -le9;
    process(u, v, [&](int l, int r) {
        res = max(res, tree->query(l, r));
    });
    return res;
}

int querySubtree(int v) { // modifySubtree is similar
    return tree->query(pos[v] + VALS_EDGES, pos[v] +
        siz[v]);
}
};
```

DirectedMST.h

**Description:** Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

**Time:**  $\mathcal{O}(E \log V)$

"/data-structures/UnionFindRollback.h" 39e620, 60 lines

```
struct Edge { int a, b; ll w; };
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};

Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}

void pop(Node*& a) { a->prop(); a = merge(a->l, a->r);
}

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
```



```
for (Edge e : g) heap[e.b] = merge(heap[e.b], new
    Node{e});
ll res = 0;
vi seen(n, -1), path(n), par(n);
seen[r] = r;
vector<Edge> Q(n), in(n, {-1,-1}), comp;
deque<tuple<int, int, vector<Edge>>> cycs;
rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
        if (!heap[u]) return {-1,{};};
        Edge e = heap[u]->top();
        heap[u]->delta -= e.w, pop(heap[u]);
        Q[qi] = e, path[qi++] = u, seen[u] = s;
        res += e.w, u = uf.find(e.a);
        if (seen[u] == s) {
            Node* cyc = 0;
            int end = qi, time = uf.time();
            do cyc = merge(cyc, heap[w = path[--qi]]);
            while (uf.join(u, w));
            u = uf.find(u), heap[u] = cyc, seen[u] = -1;
            cycs.push_front({u, time, {&Q[qi], &Q[end]}});
        }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
}

for (auto& [u,t,comp] : cycs) { // restore sol (
    optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
}
rep(i,0,n) par[i] = in[i].a;
return {res, par};
}
```

## 6.8 Math

### 6.8.1 Number of Spanning Trees

Create an  $N \times N$  matrix  $\text{mat}$ , and for each edge  $a \rightarrow b \in G$ , do  $\text{mat}[a][b]--$ ,  $\text{mat}[b][b]++$  (and  $\text{mat}[b][a]--$ ,  $\text{mat}[a][a]++$  if  $G$  is undirected). Remove the  $i$ th row and column and take the determinant; this yields the number of directed spanning trees rooted at  $i$  (if  $G$  is undirected, remove any row/column).

### 6.8.2 Erdős–Gallai theorem

A simple graph with node degrees  $d_1 \geq \dots \geq d_n$  exists iff  $d_1 + \dots + d_n$  is even and for every  $k = 1 \dots n$ ,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

# Geometry (7)

## 7.1 Geometric primitives

Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x, p.y); }
    bool operator==(P p) const { return tie(x,y)==tie(p.x, p.y); }
    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
    T dot(P p) const { return x*p.x + y*p.y; }
    T cross(P p) const { return x*p.y - y*p.x; }
    T cross(P a, P b) const { return (a-*this).cross(b-*this); }
    T dist2() const { return x*x + y*y; }
    double dist() const { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() const { return atan2(y, x); }
    P unit() const { return *this/dist(); } // makes dist
    ()=1
    P perp() const { return P(-y, x); } // rotates +90
    degrees
    P normal() const { return perp().unit(); }
    // returns point rotated 'a' radians ccw around the
    origin
    P rotate(double a) const {
        return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << ", " << p.y << ")"; }
};
```

lineDistance.h

Description:

Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

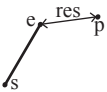
f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
```

```
return (double) (b-a).cross(p-a) / (b-a).dist();
}

SegmentDistance.h
Description:
Returns the shortest distance between point p and the line segment from point s to e.
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
"Point.h" 5c88f4, 6 lines

typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
    if (s==e) return (p-s).dist();
    auto d = (e-s).dist2(), t = min(d,max(.0, (p-s).dot(e-s)));
    return ((p-s)*d-(e-s)*t).dist()/d;
}
```



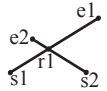
SegmentIntersection.h

Description:

If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

Usage: vector<P> inter = segInter(s1,e1,s2,e2);  
if (sz(inter)==1)  
cout << "segments intersect at " << inter[0] << endl;  
"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
    {
        auto oa = c.cross(d, a), ob = c.cross(d, b),
            oc = a.cross(b, c), od = a.cross(b, d);
        // Checks if intersection is single non-endpoint
        point.
        if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
            return {(a * ob - b * oa) / (ob - oa)};
        set<P> s;
        if (onSegment(c, d, a)) s.insert(a);
        if (onSegment(c, d, b)) s.insert(b);
        if (onSegment(a, b, c)) s.insert(c);
        if (onSegment(a, b, d)) s.insert(d);
        return {all(s)};
    }
}
```



lineIntersection.h

```

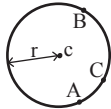
../../../../content/geometry/Point.h                                     alee63, 19 lines
typedef Point<double> P;

```

```
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p;
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.
            dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt
            (det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2;
        P u = p + d * s, v = p + d * t;
        return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2
            ;
    };
    auto sum = 0.0;
    rep(i,0,sz(ps))
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}
```

circumcircle.h

**Description:** The circumcircle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



```
Point.h
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist()/
        abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c
        )/2;
}
```

MinimumEnclosingCircle.h

**Description:** Computes the minimum circle that encloses a set of points.

```
Time: expected O(n)
circumcircle.h
pair<P, double> mec(vector<P> ps) {
    shuffle(all(ps), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0;
        rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
            o = (ps[i] + ps[j]) / 2;
            r = (o - ps[i]).dist();
            rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
                o = ccCenter(ps[i], ps[j], ps[k]);
                r = (o - ps[i]).dist();
            }
        }
    }
}
```

```

    }
}
return {o, r};
}
```

7.3 Polygons

InsidePolygon.h

**Description:** Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

```
Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
bool in = inPolygon(v, P{3, 3}, false);
Time: O(n)
Point.h, OnSegment.h, SegmentDistance.h
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
    int cnt = 0, n = sz(p);
    rep(i,0,n) {
        P q = p[(i + 1) % n];
        if (onSegment(p[i], q, a)) return !strict;
        //or: if (segDist(p[i], q, a) <= eps) return !
            strict;
        cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q
            ) > 0;
    }
    return cnt;
}
```

PolygonArea.h

**Description:** Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

```
Point.h
template<class T>
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
    return a;
}
```

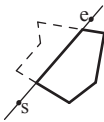
PolygonCenter.h

**Description:** Returns the center of mass for a polygon.

```
Time: O(n)
Point.h
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);
    }
    return res / A / 3;
}
```

PolygonCut.h

**Description:** Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.



```
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));
Point.h, LineIntersection.h
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
    vector<P> res;
    rep(i,0,sz(poly)) {
        P cur = poly[i], prev = i ? poly[i-1] : poly.back()
            ;
        bool side = s.cross(e, cur) < 0;
        if (side != (s.cross(e, prev) < 0))
            res.push_back(lineInter(s, e, cur, prev).second);
        if (side)
            res.push_back(cur);
    }
    return res;
}
```

ConvexHull.h

**Description:** Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.



```
Time: O(n log n)
Point.h
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
    if (sz(pts) <= 1) return pts;
    sort(all(pts));
    vector<P> h(sz(pts)+1);
    int s = 0, t = 0;
    for (int it = 2; it--; s = --t, reverse(all(pts)))
        for (P p : pts) {
            while (t >= s + 2 && h[t-2].cross(h[t-1], p) <=
                0) t--;
            h[t++] = p;
        }
    return {h.begin(), h.begin() + t - (t == 2 && h[0] ==
        h[1])};
}
```

HullDiameter.h

**Description:** Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

```
Time: O(n)
Point.h
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
    int n = sz(S), j = n < 2 ? 0 : 1;
    pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
    rep(i,0,j)
        for (; j = (j + 1) % n) {
```

```
        res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]
        }}});
    if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]
        )) >= 0)
        break;
    }
    return res.second;
}
```

PointInsideHull.h

**Description:** Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren’t included.  
**Time:**  $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h"71446b, 14 lines

```
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
    int a = 1, b = sz(l) - 1, r = !strict;
    if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
    if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
    if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p) <= -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
    }
    return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

**Description:** Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon:  $\bullet (-1, -1)$  if no collision,  $\bullet (i, -1)$  if touching the corner  $i$ ,  $\bullet (i, i)$  if along side  $(i, i + 1)$ ,  $\bullet (i, j)$  if crossing sides  $(i, i + 1)$  and  $(j, j + 1)$ . In the last case, if a corner  $i$  is crossed, this is treated as happening on side  $(i, i + 1)$ . The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.  
**Time:**  $\mathcal{O}(\log n)$

"Point.h"7cf45b, 39 lines

```
#define cmp(i, j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0

template <class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
    }
```

```
    }
    return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    rep(i, 0, 2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}
```

7.4 Misc. Point Set Problems

ClosestPair.h

**Description:** Finds the closest pair of points.  
**Time:**  $\mathcal{O}(n \log n)$

"Point.h"ac41a6, 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
    assert(sz(v) > 1);
    set<P> S;
    sort(all(v), [](P a, P b) { return a.y < b.y; });
    pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
    int j = 0;
    for (P p : v) {
        P d{1 + (ll)sqrt(ret.first), 0};
        while (v[j].y <= p.y - d.x) S.erase(v[j++]);
        auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
        for (; lo != hi; ++lo)
            ret = min(ret, {( *lo - p).dist2(), { *lo, p } });
        S.insert(p);
    }
    return ret.second;
}
```

sweepLine.h

**Description:** .  
**Time:** .

4709c6, 75 lines

```
const double EPS = 1E-9;
struct pt {
    double x, y;
};
struct seg {
    pt p, q; int id;
    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS) return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};
bool intersectld(double l1, double r1, double l2, double r2) {
    if (l1 > r1) swap(l1, r1);
    if (l2 > r2) swap(l2, r2);
    return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt& a, const pt& b, const pt& c) {
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}
bool intersect(const seg& a, const seg& b) {
    return intersectld(a.p.x, a.q.x, b.p.x, b.q.x) &&
        intersectld(a.p.y, a.q.y, b.p.y, b.q.y) &&
        vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
        vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}
bool operator<(const seg& a, const seg& b) {
    double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
    double x; int tp, id;
    event() {}
    event(double x, int tp, int id) : x(x), tp(tp), id(id) {}
    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS) return x < e.x;
        return tp > e.tp;
    }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}
pair<int, int> solve(const vector<seg>& a) {
```

```

int n = (int)a.size();
vector<event> e;
for (int i = 0; i < n; ++i) {
    e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
    e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
}
sort(e.begin(), e.end());
s.clear();
where.resize(a.size());
for (size_t i = 0; i < e.size(); ++i) {
    int id = e[i].id;
    if (e[i].tp == +1) {
        set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
        if (nxt != s.end() && intersect(*nxt, a[id]))
            return make_pair(nxt->id, id);
        if (prv != s.end() && intersect(*prv, a[id]))
            return make_pair(prv->id, id);
        where[id] = s.insert(nxt, a[id]);
    } else {
        set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
        if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
            return make_pair(prv->id, nxt->id);
        s.erase(where[id]);
    }
}
return make_pair(-1, -1);
}

```

## Strings (8)

### KMP.h

**Description:** pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.

**Time:**  $\mathcal{O}(n)$

d4375c, 16 lines

```

vi pi(const string& s) {
    vi p(sz(s));
    rep(i, 1, sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}

```

```

vi match(const string& s, const string& pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i, sz(p)-sz(s), sz(p))

```

```

        if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
        ;
    return res;
}

```

### Zfunc.h

**Description:** z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

**Time:**  $\mathcal{O}(n)$

ee09e2, 12 lines

```

vi Z(const string& S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i, 1, sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - l]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;
}

```

### Manacher.h

**Description:** For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).

**Time:**  $\mathcal{O}(N)$

c7ad79, 13 lines

```

array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi, 2> p = {vi(n+1), vi(n)};
    rep(z, 0, 2) for (int i=0, l=0, r=0; i < n; i++) {
        int t = r-i+!z;
        if (i < r) p[z][i] = min(t, p[z][l+t]);
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L >= 1 && R+1 < n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R > r) l=L, r=R;
    }
    return p;
}

```

### Hashing1.h

**Description:** Various self-explanatory methods for string hashing. Use on Codeforces, which lacks 64-bit support and where solutions can be hacked.

54a2b2, 68 lines

using hash\_t = uint64\_t;

// When  $P = 2^{32} - 13337$ , both  $P$  and  $(P - 1) / 2$  are prime.

const hash\_t HASH\_P = (unsigned) -13337;

// Avoid multiplication bases near 0 or  $P - 1$ .

uniform\_int\_distribution<hash\_t> MULT\_DIST(0.1 \* HASH\_P, 0.9 \* HASH\_P);

const hash\_t HASH\_MULT[] = {MULT\_DIST(rng), MULT\_DIST(rng)};

```

const int HASH_COUNT = 2;
vector<unsigned> hash_pow[HASH_COUNT];

```

```

struct string_hash {
    vector<unsigned> prefix_hash[HASH_COUNT];
    string result;

    string_hash() {}
}

```

```

template<typename T_string>
string_hash(const T_string &str) {
    build(str);
}

```

```

int length() const {
    return max((int) prefix_hash[0].size() - 1, 0);
}

```

```

template<typename T_string>
void build(const T_string &str) {
    for (int h = 0; h < HASH_COUNT; h++) {
        if (hash_pow[h].empty())
            hash_pow[h] = {1};

```

```

        while (hash_pow[h].size() <= str.size())
            hash_pow[h].push_back(HASH_MULT[h] *
                                   hash_pow[h].back() % HASH_P);

```

```

        prefix_hash[h].resize(str.size() + 1);
        prefix_hash[h][0] = 0;

```

```

        for (int i = 0; i < (int) str.size(); i++)
            prefix_hash[h][i + 1] = (HASH_MULT[h] *
                                       prefix_hash[h][i] + str[i]) %
                                   HASH_P;

```

```

void add_char(char c) {
    result += c;
}

```

```

for (int h = 0; h < HASH_COUNT; h++) {
    if (hash_pow[h].empty())
        hash_pow[h] = {1};

```

```

    hash_pow[h].push_back(HASH_MULT[h] *
                           hash_pow[h].back() % HASH_P);

```

```

    if (prefix_hash[h].empty())
        prefix_hash[h] = {0};

```

```

    prefix_hash[h].push_back((HASH_MULT[h] *
                               prefix_hash[h].back() + c) % HASH_P);
}

```

```

hash_t _substring_hash(int h, int start, int end)
    const {

```

```
hash_t start_hash = (hash_t) prefix_hash[h][
    start] * hash_pow[h][end - start];
return (prefix_hash[h][end] + HASH_P * HASH_P -
    start_hash) % HASH_P;
}

hash_t combined_hash(int start, int end) const {
    return _substring_hash(0, start, end) + (
        HASH_COUNT > 1 ? _substring_hash(1, start,
            end) << 32 : 0);
}
};
```

MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.  
**Usage:** rotate(v.begin(), v.begin()+minRotation(v), v.end());  
**Time:**  $\mathcal{O}(N)$

```
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1);
            break;}
        if (s[a+k] > s[b+k]) { a = b; break; }
    }
    return a;
}
```

SuffixArray.h

**Description:** Builds suffix array for a string. sa[i] is the starting index of the suffix which is *i*'th in the sorted suffix array. The returned vector is of size *n*+1, and sa[0] = *n*. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0. The input string must not contain any zero bytes.  
**Time:**  $\mathcal{O}(n \log n)$

```
struct SuffixArray {
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or
        basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2),
            lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1
                : p++;
        }
        rep(i,1,n) rank[sa[i]] = i;
```

```
for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
    for (k && k--, j = sa[rank[i] - 1];
        s[i + k] == s[j + k]; k++);
}
};
```

SuffixTree.h

**Description:** Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).  
**Time:**  $\mathcal{O}(26N)$

```
struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;

    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
                p[m++]=v; v=s[v]; q=r[v]; goto suff; }
            v=t[v][c]; q=l[v];
        }
        if (q==-1 || c==toi(a[q])) q++; else {
            l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
            p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
            l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;
            v=s[p[m]]; q=l[m];
            while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]
                ]; }
            if (q==r[m]) s[m]=v; else s[m]=m+2;
            q=r[v]-(q-r[m]); m+=2; goto suff;
        }
    }

    SuffixTree(string a) : a(a) {
        fill(r,r+N,sz(a));
        memset(s, 0, sizeof s);
        memset(t, -1, sizeof t);
        fill(t[1],t[1]+ALPHA,0);
        s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p
            [1] = 0;
        rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
    }

    // example: find longest common substring (uses ALPHA
        = 28)
    pii best;
    int lcs(int node, int i1, int i2, int olen) {
        if (l[node] <= i1 && i1 < r[node]) return 1;
        if (l[node] <= i2 && i2 < r[node]) return 2;
        int mask = 0, len = node ? olen + (r[node] - l[node]
            ) : 0;
        rep(c,0,ALPHA) if (t[node][c] != -1)
```

```
mask |= lcs(t[node][c], i1, i2, len);
if (mask == 3)
    best = max(best, {len, r[node] - len});
return mask;
}

static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z'
        + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
}
};
```

automaton.h

**Description:** .  
**Time:** .

```
void compute_automaton(string s, vector<vector<int>>&
    aut) {
    s += '#';
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(26));
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c != s[i])
                aut[i][c] = aut[pi[i-1]][c];
            else
                aut[i][c] = i + ('a' + c == s[i]);
        }
    }
}
```

AhoCorasick.h

**Description:** Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(–, word) finds all words (up to  $N\sqrt{N}$  many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.  
**Time:** construction takes  $\mathcal{O}(26N)$ , where *N* = sum of length of patterns. find(*x*) is  $\mathcal{O}(N)$ , where *N* = length of *x*. findAll is  $\mathcal{O}(NM)$ .

```
struct AhoCorasick {
    enum {alpha = 26, first = 'A'}; // change this!
    struct Node {
        // (nmatches is optional)
        int back, next[alpha], start = -1, end = -1,
            nmatches = 0;
        Node(int v) { memset(next, v, sizeof(next)); }
    };
    vector<Node> N;
    vi backp;
    void insert(string& s, int j) {
        assert(!s.empty());
        int n = 0;
```

```
for (char c : s) {
    int& m = N[n].next[c - first];
    if (m == -1) { n = m = sz(N); N.emplace_back(-1);
    }
    else n = m;
}
if (N[n].end == -1) N[n].start = j;
backp.push_back(N[n].end);
N[n].end = j;
N[n].nmatches++;
}

AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
        int n = q.front(), prev = N[n].back;
        rep(i,0,alpha) {
            int &ed = N[n].next[i], y = N[prev].next[i];
            if (ed == -1) ed = y;
            else {
                N[ed].back = y;
                (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
                = N[y].end;
                N[ed].nmatches += N[y].nmatches;
                q.push(ed);
            }
        }
    }
}

vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    for (char c : word) {
        n = N[n].next[c - first];
        res.push_back(N[n].end);
        // count += N[n].nmatches;
    }
    return res;
}

vector<vi> findAll(vector<string>& pat, string word)
{
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
        int ind = r[i];
        while (ind != -1) {
            res[i - sz(pat[ind]) + 1].push_back(ind);
            ind = backp[ind];
        }
    }
    return res;
}
};
```

# Various (9)

## 9.1 Intervals

### IntervalContainer.h

**Description:** Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).  
**Time:**  $\mathcal{O}(\log N)$

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
    R) {
        if (L == R) return is.end();
        auto it = is.lower_bound({L, R}), before = it;
        while (it != is.end() && it->first <= R) {
            R = max(R, it->second);
            before = it = is.erase(it);
        }
        if (it != is.begin() && (--it)->second >= L) {
            L = min(L, it->first);
            R = max(R, it->second);
            is.erase(it);
        }
        return is.insert(before, {L,R});
    }

void removeInterval(set<pii>& is, int L, int R) {
    if (L == R) return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L) is.erase(it);
    else (int&)it->second = L;
    if (R != r2) is.emplace(R, r2);
}
```

### IntervalCover.h

**Description:** Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add || R.empty(). Returns empty set on failure (or if G is empty).  
**Time:**  $\mathcal{O}(N \log N)$

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
    vi S(sz(I)), R;
    iota(all(S), 0);
    sort(all(S), [&](int a, int b) { return I[a] < I[b];
    });
    T cur = G.first;
    int at = 0;
    while (cur < G.second) { // (A)
        pair<T, int> mx = make_pair(cur, -1);
        while (at < sz(I) && I[S[at]].first <= cur) {
            mx = max(mx, make_pair(I[S[at]].second, S[at]));
            at++;
        }
        if (mx.second == -1) return {};
        cur = mx.first;
    }
```

```
        R.push_back(mx.second);
    }
    return R;
}
```

### ConstantIntervals.h

**Description:** Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.  
**Usage:** constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});  
**Time:**  $\mathcal{O}(k \log \frac{n}{k})$

```
template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
    q) {
        if (p == q) return;
        if (from == to) {
            g(i, to, p);
            i = to; p = q;
        } else {
            int mid = (from + to) >> 1;
            rec(from, mid, f, g, i, p, f(mid));
            rec(mid+1, to, f, g, i, p, q);
        }
    }
}

template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
    if (to <= from) return;
    int i = from; auto p = f(i), q = f(to-1);
    rec(from, to-1, f, g, i, p, q);
    g(i, to, q);
}
```

## 9.2 Misc. algorithms

### TernarySearch.h

**Description:** Find the smallest i in [a,b] that maximizes f(i), assuming that f(a) < ... < f(i) ≥ ... ≥ f(b). To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).  
**Usage:** int ind = ternSearch(0,n-1,[&](int i){return a[i];});  
**Time:**  $\mathcal{O}(\log(b-a))$

```
template<class F>
int ternSearch(int a, int b, F f) {
    assert(a <= b);
    while (b - a >= 5) {
        int mid = (a + b) / 2;
        if (f(mid) < f(mid+1)) a = mid; // (A)
        else b = mid+1;
    }
    rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
    return a;
}
```

LIS.h

**Description:** Compute indices for the longest increasing subsequence.

**Time:**  $\mathcal{O}(N \log N)$

2932a0, 17 lines

```
template<class I> vi lis(const vector<I>& S) {
    if (S.empty()) return {};
    vi prev(sz(S));
    typedef pair<I, int> p;
    vector<p> res;
    rep(i,0,sz(S)) {
        // change 0 -> i for longest non-decreasing
        // subsequence
        auto it = lower_bound(all(res), p{S[i], 0});
        if (it == res.end()) res.emplace_back(), it = res.
            end()-1;
        *it = {S[i], i};
        prev[i] = it == res.begin() ? 0 : (it-1)->second;
    }
    int L = sz(res), cur = res.back().second;
    vi ans(L);
    while (L--) ans[L] = cur, cur = prev[cur];
    return ans;
}
```

FastKnapsack.h

**Description:** Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.

**Time:**  $\mathcal{O}(N \max(w_i))$

b20ccc, 16 lines

```
int knapsack(vi w, int t) {
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t) a += w[b++];
    if (b == sz(w)) return a;
    int m = *max_element(all(w));
    vi u, v(2*m, -1);
    v[a+m-t] = b;
    rep(i,b,sz(w)) {
        u = v;
        rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
        for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
            v[x-w[j]] = max(v[x-w[j]], j);
    }
    for (a = t; v[a+m-t] < 0; a--);
    return a;
}
```

## 9.3 Dynamic programming

### KnuthDP.h

**Description:** When doing DP on intervals:  $a[i][j] = \min_{i < k < j} (a[i][k] + a[k][j]) + f(i, j)$ , where the (minimal) optimal  $k$  increases with both  $i$  and  $j$ , one can solve intervals in increasing order of length, and search  $k = p[i][j]$  for  $a[i][j]$  only between  $p[i][j - 1]$  and  $p[i + 1][j]$ . This is known as Knuth DP. Sufficient criteria for this are if  $f(b, c) \leq f(a, d)$  and  $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$  for all  $a \leq b \leq c \leq d$ . Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

**Time:**  $\mathcal{O}(N^2)$

DivideAndConquerDP.h

**Description:** Given  $a[i] = \min_{lo(i) \leq k < hi(i)} (f(i, k))$  where the (minimal) optimal  $k$  increases with  $i$ , computes  $a[i]$  for  $i = L..R - 1$ .

**Time:**  $\mathcal{O}((N + (hi - lo)) \log N)$

d38d2b, 18 lines

```
struct DP { // Modify at will:
    int lo(int ind) { return 0; }
    int hi(int ind) { return ind; }
    ll f(int ind, int k) { return dp[ind][k]; }
    void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

    void rec(int L, int R, int LO, int HI) {
        if (L >= R) return;
        int mid = (L + R) >> 1;
        pair<ll, int> best(LLONG_MAX, LO);
        rep(k, max(LO, lo(mid)), min(HI, hi(mid)))
            best = min(best, make_pair(f(mid, k), k));
        store(mid, best.second, best.first);
        rec(L, mid, LO, best.second+1);
        rec(mid+1, R, best.second, HI);
    }
    void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

## 9.4 Optimization tricks

\_\_builtin\_ia32\_ldmxcsr(40896); disables denormals (which make floats 20x slower near their minimum value).

### 9.4.1 Bit hacks

- $x \ \& \ -x$  is the least bit in  $x$ .
- ```
for (int x = m; x; ) { --x &= m; ... }
```

 loops over all subset masks of  $m$  (except  $m$  itself).
- $c = x\&-x$ ,  $r = x+c$ ;  $((r\^x) >> 2)/c$  |  $r$  is the next number after  $x$  with the same number of bits set.
- ```
rep(b,0,K) rep(i,0,(1 << K))
    if (i & 1 << b) D[i] += D[i^(1 << b)];
```

 computes all sums of subsets.

FastMod.h

**Description:** Compute  $a\%b$  about 5 times faster than usual, where  $b$  is constant but not known at compile time. Returns a value congruent to  $a \pmod b$  in the range  $[0, 2b)$ .

751a02, 8 lines

```
typedef unsigned long long ull;
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m(-1ULL / b) {}
    ull reduce(ull a) { // a % b + (0 or b)
        return a - (ull)((__uint128_t(m) * a) >> 64) * b;
    }
};
```

FastInput.h

**Description:** Read an integer from stdin. Usage requires your program to pipe in input from file.

**Usage:** ./a.out < input.txt

**Time:** About 5x as fast as cin/scanf.

7b3c70, 17 lines

```
inline char gc() { // like getchar()
    static char buf[1 << 16];
    static size_t bc, be;
    if (bc >= be) {
        buf[0] = 0, bc = 0;
        be = fread(buf, 1, sizeof(buf), stdin);
    }
    return buf[bc++]; // returns 0 on EOF
}

int readInt() {
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 48;
    return a - 48;
}
```

BumpAllocator.h

**Description:** When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

745db2, 8 lines

```
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
    static size_t i = sizeof buf;
    assert(s < i);
    return (void*)&buf[i -= s];
}
void operator delete(void*) {}
```

SmallPtr.h

**Description:** A 32-bit pointer that points into BumpAllocator memory.

2dd6c9, 10 lines

```
template<class T> struct ptr {
    unsigned ind;
    ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0)
    {
```



```
    assert(ind < sizeof buf);
}
T& operator*() const { return *(T*)(buf + ind); }
T* operator->() const { return &*this; }
T& operator[](int a) const { return (&this)[a]; }
explicit operator bool() const { return ind; }
};
```

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.

Usage: vector<vector<int, small<int>>>> ed(N);

b666d4, 14 lines

```
char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;

template<class T> struct small {
    typedef T value_type;
    small() {}
    template<class U> small(const U&) {}
    T* allocate(size_t n) {
        buf_ind -= n * sizeof(T);
        buf_ind &= 0 - alignof(T);
        return (T*)(buf + buf_ind);
    }
    void deallocate(T*, size_t) {}
};
```