University of Science, VNU-HCM

# HCMUS-DewyWayLouis

Hieu Vu Tran Minh, Bao Quoc Doan, Thinh Duc Hoang

# Mathematics (1)

## 1.1 Equations

$$ax^2+bx+c=0 \Rightarrow x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

The extremum is given by $x=-b/2a$.

## 1.2 Trigonometry

$$\sin(v+w)=\sin v\cos w+\cos v\sin w$$
$$\cos(v+w)=\cos v\cos w-\sin v\sin w$$

$$\tan(v+w)=\frac{\tan v+\tan w}{1-\tan v\tan w}$$
$$\sin v+\sin w=2\sin\frac{v+w}{2}\cos\frac{v-w}{2}$$
$$\cos v+\cos w=2\cos\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$a\cos x+b\sin x=r\cos(x-\phi)$$
$$a\sin x+b\cos x=r\sin(x+\phi)$$

where $r=\sqrt{a^2+b^2}, \phi=\operatorname{atan2}(b,a)$.

## 1.3 Geometry

Side lengths: $a,b,c$

Semiperimeter: $p=\dfrac{a+b+c}{2}$

Area: $A=\sqrt{p(p-a)(p-b)(p-c)}$

Circumradius (r ngoai tiep): $R=\dfrac{abc}{4A}$

Inradius (r noi tiep): $r=\dfrac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a=\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$

Law of sines: $\dfrac{\sin\alpha}{a}=\dfrac{\sin\beta}{b}=\dfrac{\sin\gamma}{c}=\dfrac{1}{2R}$

Law of cosines: $a^2=b^2+c^2-2bc\cos\alpha$

Law of tangents: $\dfrac{a+b}{a-b}=\dfrac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

## 1.4 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x=\frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x=-\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x=1+\tan^2 x \qquad \frac{d}{dx}\arctan x=\frac{1}{1+x^2}$$
$$\int\tan ax=-\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax=\frac{\sin ax-ax\cos ax}{a^2}$$
$$\int e^{-x^2}=\frac{\sqrt{\pi}}{2}\operatorname{erf}(x) \qquad \int xe^{ax}dx=\frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx=[F(x)g(x)]_a^b-\int_a^b F(x)g'(x)dx$$

## 1.5 Sums

$$c^a+c^{a+1}+\cdots+c^b=\frac{c^{b+1}-c^a}{c-1}, c\neq 1$$

$$1+2+3+\cdots+n=\frac{n(n+1)}{2}$$
$$1^2+2^2+3^2+\cdots+n^2=\frac{n(2n+1)(n+1)}{6}$$
$$1^3+2^3+3^3+\cdots+n^3=\frac{n^2(n+1)^2}{4}$$
$$1^4+2^4+3^4+\cdots+n^4=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

## 1.6 Series

$$e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots, (-\infty<x<\infty)$$
$$\ln(1+x)=x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots, (-1<x\leq 1)$$
$$\sqrt{1+x}=1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\ldots, (-1\leq x\leq 1)$$
$$\sin x=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots, (-\infty<x<\infty)$$
$$\cos x=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots, (-\infty<x<\infty)$$
$$\ln(2)=\frac{1}{1*2}+\frac{1}{3*4}+\frac{1}{5*6}+\ldots$$
$$\ln(2)=\sum_{k=1}^{\infty}\frac{1}{2^k k}$$
$$\ln(\frac{n}{n-1})=\sum_{k=1}^{\infty}\frac{1}{n^k k}$$
$$\frac{\pi}{4}=\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\ldots$$
$$e=\frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\ldots$$

## 1.7 Sum of divisors

Given $n=p_1^{m_1}p_2^{m_2}\ldots p_k^{m_k}$ with $p_i$ is prime factor
$\sigma(n)=(p_1^0+p_1^1+\cdots+p_1^{m_1})(p_2^0+p_2^1+\cdots+p_2^{m_2})(p_k^0+p_k^1+\cdots+p_k^{m_k})$
or $\sigma(n)=\prod_{i=1}^{k}\frac{p_i^{m_i+1}-1}{p_i-1}$

# Data structures (2)

OrderedSet.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 `null_type`.
**Time:** $\mathcal{O}(\log N)$

782797, 16 lines

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

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

HashMap.h

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

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

## SegmentTree.h

**Description:** Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.
**Time:** $\mathcal{O}(\log N)$

0f4bdb, 19 lines

```cpp
struct Tree {
  typedef int T;
  static constexpr T unit = INT_MIN;
  T f(T a, T b) { return max(a, b); } // (any associative fn)
  vector<T> s; int n;
  Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {}
  void update(int pos, T val) {
    for (s[pos += n] = val; pos /= 2;)
      s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
  }
  T query(int b, int e) { // query [b, e)
    T ra = unit, rb = unit;
    for (b += n, e += n; b < e; b /= 2, e /= 2) {
      if (b % 2) ra = f(ra, s[b++]);
      if (e % 2) rb = f(s[--e], rb);
    }
    return f(ra, rb);
  }
};
```

## LazySegmentTree.h

**Description:** 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.
**Usage:** `Node* tr = new Node(v, 0, sz(v));`
**Time:** $\mathcal{O}(\log N)$.

"../various/BumpAllocator.h" 34ecf5, 50 lines

```cpp
const int inf = 1e9;
struct Node {
  Node *l = 0, *r = 0;
  int lo, hi, mset = inf, madd = 0, val = -inf;
  Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
  Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
    if (lo + 1 < hi) {
      int mid = lo + (hi - lo)/2;
      l = new Node(v, lo, mid); r = new Node(v, mid, hi);
      val = max(l->val, r->val);
    }
    else val = v[lo];
  }
  int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val;
    push();
    return max(l->query(L, R), r->query(L, R));
  }
  void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
    else {
      push(), l->set(L, R, x), r->set(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
      if (mset != inf) mset += x;
      else madd += x;
      val += x;
    }
    else {
      push(), l->add(L, R, x), r->add(L, R, x);
      val = max(l->val, r->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->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
    else if (madd)
      l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
  }
};
```

## DSURollback.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:** $\mathcal{O}(\log(N))$

de4ad0, 21 lines

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

## PrefixSum2D.h

**Description:** Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
**Usage:** `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
**Time:** $\mathcal{O}\left(N^2 + Q\right)$

c59ada, 13 lines

```cpp
template<class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u][l];
  }
};
```

## 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;`

c43c7d, 26 lines

```cpp
template<class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& 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<T> operator*(const vector<T>& vec) const {
    vector<T> 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 >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      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:** $\mathcal{O}(\log N)$

8ec1c7, 30 lines

```cpp
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] + ... + a[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] >= sum
    // 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], ... 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)]);
  }
};
```

## RRRangeSet.h

**Description:** MergeAdjSeg is true = merge 2 consecutive segments, e.g. [1, 10] and [11, 20] = [1, 20] ca1905, 84 lines

```
template<typename T, bool MergeAdjSeg = true>
struct RangeSet {
    T n_elements() const { return sz; }
    T n_ranges() const { return ranges.size(); }

    bool contains(T x) const {
        auto it = ranges.upper_bound(x);
        return it != ranges.begin() && x <= prev(it)->second;
    }

    // Find range containing x, i.e. l <= x <= r
    auto find_range(T x) const {
        auto it = ranges.upper_bound(x);
        return it != ranges.begin() && x <= prev(it)->second ?
            prev(it) : ranges.end();
    }

    T insert(T l, T r) {  // Insert [l, r]
        assert(l <= r);
        auto it = ranges.upper_bound(l);
        if (it != ranges.begin() && is_mergeable(prev(it)->
            second, l)) {
            it = prev(it);
            l = min(l, it->first);
        }
        T inserted = 0;
        for (; it != ranges.end() && is_mergeable(r, it->first)
            ; it = ranges.erase(it)) {
            auto [cl, cr] = *it;
            r = max(r, cr);
            inserted -= cr - cl + 1;
        }

        inserted += r - l + 1;
        ranges[l] = r;
        sz += inserted;
        return inserted;
    }

    T erase(T l, T r) {   // Erase [l, r]
        assert(l <= r);
        T tl = l, tr = r;
        auto it = ranges.upper_bound(l);
        if (it != ranges.begin() && l <= prev(it)->second) {
            it = prev(it);
            tl = it->first;
        }

        T erased = 0;
        for (; it != ranges.end() && it->first <= r; it =
            ranges.erase(it)) {
            auto [cl, cr] = *it;
            tr = cr;
            erased += cr - cl + 1;
        }
        if (tl < l) {
            ranges[tl] = l-1;
            erased -= l - tl;
        }
        if (r < tr) {
            ranges[r + 1] = tr;
        erased -= tr - r;
    }
    sz -= erased;
    return erased;
}

// Find min x: x >= lower && x NOT in this set
T minimum_excluded(T lower) const {
    static_assert(MergeAdjSeg);
    auto it = find_range(lower);
    return it == ranges.end() ? lower : it->second + 1;
}

// Find max x: x <= upper && x NOT in this set
T maximum_excluded(T upper) const {
    static_assert(MergeAdjSeg);
    auto it = find_range(upper);
    return it == ranges.end() ? upper : it->first - 1;
}

T sz = 0;
map<T, T> ranges;

bool is_mergeable(T cur_r, T next_l) {
    return next_l <= cur_r + MergeAdjSeg;
}
};
```

## PersistentArray.h

**Description:** PersistentArray

550250, 42 lines

```
template<typename T>
struct PersistentArray {
    static const int LOG = 4;
    static const int FULL_MASK = (1<<LOG) - 1;

    struct Node {
        T x;
        array<Node*, 1<<LOG> child;
        Node(const T& _x) : x(_x) {
            fill(child.begin(), child.end(), nullptr);
        }
        Node(const Node& n) : x(n.x) {
            copy(n.child.begin(), n.child.end(), child.begin())
                ;
        }
    };

    // get p-th element in `t` version
    static T get(Node* t, int p) {
        if (t == NULL) return 0;
        return p ? get(t->child[p & FULL_MASK], p >> LOG) : t->
             x;
    }

    // set p-th element in `t` version to x, and return new
         version
    static Node* set(Node* t, int p, const T& x) {
        t = (t == NULL) ? new Node(0) : new Node(*t);
        if (p == 0) t->x = x;
        else {
            auto ptr = set(t->child[p & FULL_MASK], p >> LOG, x
                );
            t->child[p & FULL_MASK] = ptr;
        }
        return t;
    }

    // init a persistent array and return root node
    Node* build(const vector<T>& v) {
        Node* root = NULL;
        for (int i = 0; i < (int) v.size(); i++) {
            root = set(root, i, v[i]);
        }
        return root;
    }
};
```

## LiChao.h

**Description:** li-chao tree for vuhieu

1fe529, 119 lines

```
template<
typename T,  // for segment & coordinates data types, e.g. long
      long
typename TM>  // for intermediate computations, e.g. __int128_t
struct LiChao {
    LiChao(const vector<T>& _xs) : xs(_xs) {
        sort(xs.begin(), xs.end());
        xs.erase(unique(xs.begin(), xs.end()), xs.end());

        n = xs.size();
        head = 1;
        while (head < n) head <<= 1;

        lines.assign(head * 2, {0, 0, -1, false});
        xyz.resize(head * 2);
        for (int i = 0; i < n; i++) {
            xyz[head + i] = {xs[i], xs[i], xs[i]};
        }
        for (int i = head - 1; i; i--) {
            int l = i*2, r = i*2 + 1;
            xyz[i] = {
                get<0>(xyz[l]),
                get<0>(xyz[r]),
                get<2>(xyz[r]),
            };
        }
    }

    void add_line(T a, T b, int idx = -1) {
        ql = 0, qr = n;
        if (ql >= qr) return;
        rec(1, 0, head, Line{a, b, idx, true});
    }

    void add_segment(T left, T right, T a, T b, int idx = -1) {
        ql = lower_bound(xs.begin(), xs.end(), left) - xs.begin
            ();
        qr = lower_bound(xs.begin(), xs.end(), right) - xs.
            begin();
        if (ql >= qr) return;
        rec(1, 0, head, Line{a, b, idx, true});
    }

    struct Result {
        T line_a, line_b;
        int line_id;
        bool is_valid;  // if false -> result is INFINITY
        TM minval;
    };

    Result get(T x) {
        int i = lower_bound(xs.begin(), xs.end(), x) - xs.begin
            ();
        assert(i < n && xs[i] == x);
        return get(i, x);
    }

// private:
    int n, head;
    vector<T> xs;  // coordinates of all get queries

    struct Line {
        T a, b;  // a*x + b
        int id;
        bool is_valid;
        TM f(T x) const { return TM(a) * x + b; }
    };
    vector<Line> lines;

    vector<tuple<T, T, T>> xyz;  // <left, mid, right>

    int ql, qr;
    void rec(int i, int l, int r, Line new_line) {
        const int mid = (l + r) / 2;

        if (l >= qr || r <= ql) {
            return;
        } else if (ql <= l && r <= qr) {
            if (!lines[i].is_valid) {
                lines[i] = new_line;
                return;
            }

            auto [x, y, z] = xyz[i];
            bool upd_x = lines[i].f(x) > new_line.f(x);
            bool upd_y = lines[i].f(y) > new_line.f(y);
            bool upd_z = lines[i].f(z) > new_line.f(z);

            if (upd_x && upd_y && upd_z) {
                lines[i] = new_line;
                return;
            }
            if (upd_y && upd_z) {
                swap(lines[i], new_line);
                rec(i*2, l, mid, new_line);
            } else if (upd_x && upd_y) {
                swap(lines[i], new_line);
                rec(i*2 + 1, mid, r, new_line);
            } else if (upd_x) {
                rec(i*2, l, mid, new_line);
            } else if (upd_z) {
                rec(i*2+1, mid, r, new_line);
            } else {
                return;
            }
        } else {
            if (ql < mid) rec(i*2, l, mid, new_line);
            if (qr > mid) rec(i*2+1, mid, r, new_line);
        }
    }

    Result get(int i, T x) {
        i += head;
        Line res = lines[i];
        TM val = res.is_valid ? res.f(x) : 0;
        for (i /= 2; i; i /= 2) {
            if (!lines[i].is_valid) continue;
            TM tmp = lines[i].f(x);
            if (!res.is_valid || tmp < val) res = lines[i], val
                 = tmp;
        }
        return {res.a, res.b, res.id, res.is_valid, val};
    }
};
```

# Numerical (3)

## 3.1 Optimization

### Integrate.h

**Description:** Simple integration of a function over an interval using Simpson's rule. The error should be proportional to $h^4$, although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

4756fc, 7 lines

```
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
  double h = (b - a) / 2 / n, v = f(a) + f(b);
  rep(i,1,n*2)
    v += f(a + i*h) * (i&1 ? 4 : 2);
  return v * h / 3;
}
```

## 3.2 Matrices

### RRMatrix.h

**Description:** Matrix, which works for both double and int

61647f, 198 lines

```
template<typename T>
struct Matrix {
    int n_row, n_col;
    vector<T> x;

    // access
    typename vector<T>::iterator operator [] (int r) { return x
        .begin() + r * n_col; }
    inline T get(int i, int j) const { return x[i * n_col + j];
        }
    vector<T> at(int r) const {
        return vector<T> { x.begin() + r * n_col, x.begin() + (
            r+1) * n_col };
    }

    // constructors
    Matrix() = default;
    Matrix(int _n_row, int _n_col) : n_row(_n_row), n_col(
        _n_col), x(n_row * n_col) {}
    Matrix(const vector<vector<T>>& d) : n_row(d.size()), n_col
        (d.size() ? d[0].size() : 0) {
        for (auto& row : d) std::copy(row.begin(), row.end(),
            std::back_inserter(x));
    }

    // convert to 2d vec
    vector<vector<T>> vecvec() const {
        vector<vector<T>> ret(n_row);
        for (int i = 0; i < n_row; i++) {
            std::copy(x.begin() + i*n_col,
                    x.begin() + (i+1)*n_col,
                    std::back_inserter(ret[i]));
        }
        return ret;
    }
    operator vector<vector<T>>() const { return vecvec(); }

    static Matrix identity(int n) {
        Matrix res(n, n);
        for (int i = 0; i < n; i++) {
            res[i][i] = 1;
        }
        return res;
    }

    Matrix transpose() const {
        Matrix res(n_col, n_row);
        for (int i = 0; i < n_row; i++) {
            for (int j = 0; j < n_col; j++) {
                res[j][i] = this->get(i, j);
            }
        }
        return res;
    }

    Matrix& operator *= (const Matrix& r) { return *this = *
        this * r; }
    Matrix operator * (const Matrix& r) const {
        assert(n_col == r.n_row);
        Matrix res(n_row, r.n_col);

        for (int i = 0; i < n_row; i++) {
            for (int k = 0; k < n_col; k++) {
                for (int j = 0; j < r.n_col; j++) {
                    res[i][j] += this->get(i, k) * r.get(k, j);
                }
            }
        }
        return res;
    }

    Matrix pow(long long n) const {
        assert(n_row == n_col);
        Matrix res = identity(n_row);
        if (n == 0) return res;

        bool res_is_id = true;
        for (int i = 63 - __builtin_clzll(n); i >= 0; i--) {
            if (!res_is_id) res *= res;
            if ((n >> i) & 1) res *= (*this), res_is_id = false
                ;
        }
        return res;
    }

    // Gauss
    template <typename T2, typename std::enable_if<std::
        is_floating_point<T2>::value>::type * = nullptr>
    static int choose_pivot(const Matrix<T2> &mtr, int h, int c
        ) noexcept {
        int piv = -1;
        for (int j = h; j < mtr.n_row; j++) {
            if (mtr.get(j, c) and (piv < 0 or std::abs(mtr.get(
                j, c)) > std::abs(mtr.get(piv, c)))) piv = j;
        }
        return piv;
    }
    template <typename T2, typename std::enable_if<!std::
        is_floating_point<T2>::value>::type * = nullptr>
    static int choose_pivot(const Matrix<T2> &mtr, int h, int c
        ) noexcept {
        for (int j = h; j < mtr.n_row; j++) {
            if (mtr.get(j, c) != T(0)) return j;
        }
        return -1;
    }

    // return upper triangle matrix
    [[nodiscard]] Matrix gauss() const {
        int c = 0;
        Matrix mtr(*this);
        vector<int> ws;
        ws.reserve(n_col);

        for (int h = 0; h < n_row; h++) {
            if (c == n_col) break;
            int piv = choose_pivot(mtr, h, c);
            if (piv == -1) {
                c++;
                h--;
                continue;
            }
            if (h != piv) {
                for (int w = 0; w < n_col; w++) {
                    swap(mtr[piv][w], mtr[h][w]);
                    mtr[piv][w] *= -1; // for determinant
                }
            }
            ws.clear();
            for (int w = c; w < n_col; w++) {
                if (mtr[h][w] != 0) ws.emplace_back(w);
            }
            const T hcinv = T(1) / mtr[h][c];
            for (int hh = 0; hh < n_row; hh++) {
                if (hh != h) {
                    const T coeff = mtr[hh][c] * hcinv;
                    for (auto w : ws) mtr[hh][w] -= mtr[h][w] *
                        coeff;
                    mtr[hh][c] = 0;
                }
            }
            c++;
        }
        return mtr;
    }

    // For upper triangle matrix
    T det() const {
        T ret = 1;
        for (int i = 0; i < n_row; i++) {
            ret *= get(i, i);
        }
        return ret;
    }

    // return rank of inverse matrix. If rank < n -> not
        invertible
    int inverse() {
        assert(n_row == n_col);
        vector<vector<T>> ret = identity(n_row), tmp = *this;
        int rank = 0;

        for (int i = 0; i < n_row; i++) {
            int ti = i;
            while (ti < n_row && tmp[ti][i] == 0) ++ti;
            if (ti == n_row) continue;
            else ++rank;

            ret[i].swap(ret[ti]);
            tmp[i].swap(tmp[ti]);

            T inv = T(1) / tmp[i][i];
            for (int j = 0; j < n_col; j++) ret[i][j] *= inv;
            for (int j = i+1; j < n_col; j++) tmp[i][j] *= inv;

            for (int h = 0; h < n_row; h++) {
                if (i == h) continue;
                const T c = -tmp[h][i];
                for (int j = 0; j < n_col; j++) ret[h][j] +=
                    ret[i][j] * c;
                for (int j = i+1; j < n_col; j++) tmp[h][j] +=
                    tmp[i][j] * c;
            }
        }
        *this = ret;
        return rank;
    }

    // sum of all elements in this matrix
    T sum_all() {
        return submatrix_sum(0, 0, n_row, n_col);
    }

    // sum of [r1, r2) x [c1, c2)
    T submatrix_sum(int r1, int c1, int r2, int c2) {
        T res {0};
        for (int r = r1; r < r2; ++r) {
            res += std::accumulate(
                    x.begin() + r * n_col + c1,
                    x.begin() + r * n_col + c2,
                    T{0});
        }
        return res;
    }
};
template<typename T>
ostream& operator << (ostream& cout, const Matrix<T>& m) {
    cout << m.n_row << ' ' << m.n_col << endl;
    for (int i = 0; i < m.n_row; ++i) {
        cout << "row [" << i << "] = " << m.at(i) << endl;
    }
    return cout;
}
// }}}
```

# 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" 35bfea, 18 lines

```
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 $\leq$ mod and that mod is a prime.

6f684f, 3 lines

```
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

b83e45, 8 lines

```
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:** $\mathcal{O}\left(\sqrt{m}\right)$

c040b8, 11 lines

```
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;
}
```

### ModMulLL.h

**Description:** Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) 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

bbbd8f, 11 lines

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

"ModPow.h" 19a793, 24 lines

```
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;
}
```

### SieveOnRange.h

**Description:** sieve on range
**Time:** ask vuhieu

aa6ca8, 15 lines

```
const int MAX_RANGE = (int) 1e6 + 15;
bool isPrime[MAX_RANGE];
pair<int, vector<long long>> sieve_on_range(long long L, long
     long R) {
    memset(isPrime, true, sizeof isPrime);
    for (long long i = 2; i * i <= R; ++i) {
        for (long long j = max(i * i, (L + i - 1) / i * i); j
             <= R; j += i) {
            assert(0 <= j - L && j - L < MAX_RANGE);
            isPrime[j - L] = false;
        }
    }
    if (L <= 1) isPrime[1 - L] = false;
    vector<long long> prime;
    for (long long i = L; i <= R; ++i) if (isPrime[i - L])
        prime.push_back(i);
    return make_pair((int) prime.size(), prime);
}
```

### 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:** $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.
"ModMulLL.h", "MillerRabin.h" d8d98d, 18 lines

```
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, n) + i; };
  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 $0 \le x < \mathrm{lcm}(m, n)$. 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;
}
```

## 4.4 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.

# 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 Derangements

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

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

## 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 | ∼2e5 | ∼2e8 |

### 5.2.2 Lucas' Theorem

Let $n, m$ be non-negative integers and $p$ a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.

### 5.2.3 Binomials

### multinomial.h

**Description:** Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$.
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 Stirling numbers of the first kind

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

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

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

### 5.3.2 Stirling numbers of the second kind

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

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

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

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

### 5.3.3 Catalan numbers

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

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

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

- 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 $m[i][j] =$ `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:** $\mathcal{O}\left(N^3\right)$

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

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 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:** $\mathcal{O}\left(E + V\right)$

76b5c9, 24 lines

```
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);
}
```

### BridgeArticulation.h

**Description:** Finding bridges and articulation points Assume already have undirected graph vector< vector<int> > G with V vertices Vertex index from 0
**Usage:** `UndirectedDfs tree;`
`Then you can use tree.bridges and tree.articulation_points`

6d3f6c, 49 lines

```
struct UndirectedDfs {
    vector<vector<int>> g;
    int n;
    vector<int> low, num, parent;
    vector<bool> is_articulation;
    int counter, root, children;

    vector< pair<int,int> > bridges;
    vector<int> articulation_points;
    map<pair<int,int>, int> cnt_edges;

    UndirectedDfs(const vector<vector<int>>& _g) : g(_g), n(g.
         size()),
            low(n, 0), num(n, -1), parent(n, 0),
                 is_articulation(n, false),
            counter(0), children(0) {
        for (int u = 0; u < n; u++) {
            for (int v : g[u]) {
                cnt_edges[{u, v}] += 1;
            }
        }
        for(int i = 0; i < n; ++i) if (num[i] == -1) {
            root = i; children = 0;
            dfs(i);
            is_articulation[root] = (children > 1);
        }
        for(int i = 0; i < n; ++i)
            if (is_articulation[i]) articulation_points.
                 push_back(i);
    }
private:
    void dfs(int u) {
        low[u] = num[u] = counter++;
        for (int v : g[u]) {
            if (num[v] == -1) {
                parent[v] = u;
                if (u == root) children++;
                dfs(v);
                if (low[v] >= num[u])
                    is_articulation[u] = true;
                if (low[v] > num[u]) {
                    if (cnt_edges[{u, v}] == 1) {
                        bridges.push_back(make_pair(u, v));
                    }
                }
                low[u] = min(low[u], low[v]);
            } else if (v != parent[u])
                low[u] = min(low[u], num[v]);
        }
    }
};
// }}}
```

### EulerWalk.h

**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:** $\mathcal{O}\left(V + E\right)$

cdbd50, 15 lines

```
vector<int> eulerWalk(vector<vector<pii>>& gr, int nedges, int
     src=0) {
  int n = sz(gr);
  vector<int> D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler 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()};
}
```

## 6.3 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 $\mathcal{O}\left(N \log N\right)$, queries $\mathcal{O}\left(\log N\right)$

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)];}
};
```

## LCAConstant.h

**Description:** O(1) LCA and RMQ
**Time:** $\mathcal{O}(1)$ query, $\mathcal{O}(n \log n)$ precompute

da86a6, 107 lines

```
template <class T>
struct RMQ { // 0-based
  vector<vector<T>> rmq;
  T kInf = numeric_limits<T>::max();
  void build(const vector<T>& V) {
    int n = V.size(), on = 1, dep = 1;
    while (on < n) on *= 2, ++dep;
    rmq.assign(dep, V);

    for (int i = 0; i < dep - 1; ++i)
      for (int j = 0; j < n; ++j) {
        rmq[i + 1][j] = min(rmq[i][j], rmq[i][min(n - 1, j + (1
              << i))]);
      }
  }
  T query(int a, int b) { // [a, b)
    if (b <= a) return kInf;
    int dep = 31 - __builtin_clz(b - a); // log(b - a)
    return min(rmq[dep][a], rmq[dep][b - (1 << dep)]);
  }
};

struct LCA { // 0-based
  int LG, N;
  vector<int> enter, depth, exxit;
  vector<vector<int>> G, par;
  vector<pair<int, int>> linear;
  RMQ<pair<int, int>> rmq;
  int timer = 0;
  LCA() {}
  LCA(int n) {
    N = n;
    LG = __lg(N) + 1;
    enter.assign(n + 1, -1);
    exxit.assign(n + 1, -1);
    depth.resize(n + 1);
    G.resize(n + 1);
    par.assign(n + 1, vector<int>(LG + 1, -1));
    linear.resize(2 * n + 1);
  }
  void init(int n) {
    N = n;
    LG = __lg(N) + 1;
    enter.assign(n + 1, -1);
    exxit.assign(n + 1, -1);
    depth.resize(n + 1);
    G.resize(n + 1);
    par.assign(n + 1, vector<int>(LG + 1, -1));
    linear.resize(2 * n + 1);
  }
  void dfs(int node, int dep) {
    linear[timer] = {dep, node};
    enter[node] = timer++;
    depth[node] = dep;
    sz[node] = 1;
    for (auto vec : G[node])
    if (enter[vec] == -1) {
      par[vec][0] = node;
      dfs(vec, dep + 1);
      linear[timer++] = {dep, node};
      sz[node] += sz[vec];
    }
    exxit[node] = timer;
  }
  void add_edge(int a, int b) {
    G[a].push_back(b);
    G[b].push_back(a);
  }
  void build(int root) {
    dfs(root, 0);
    rmq.build(linear);
  }
  void build_par(void) {
    for (int j = 1; j < LG; ++j)
      for (int i = 1; i <= N; ++i) if (par[i][j - 1] != -1) {
        par[i][j] = par[par[i][j - 1]][j - 1];
    }
  }
  int kth(int node, int k) {
    for (int i = k; i > 0; i ^= (i & -i)) {
      int ind = __builtin_ctz(i);
      if (par[node][ind] != -1) node = par[node][ind];
    }
    return node;
  }
  int tin(int u) { return enter[u] + 1; }
  int tout(int u) { return exxit[u] + 1; }
  int query(int a, int b) {
    if (a == 0 || b == 0) return max(a, b);
    a = enter[a], b = enter[b];
    return rmq.query(min(a, b), max(a, b) + 1).second;
  }
  int queryLOG(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    for (int i = LG - 1; i >= 0; --i) if (depth[par[u][i]] >=
          depth[v]) {
      u = par[u][i];
    }
    if (u == v) return u;
    for (int i = LG - 1; i >= 0; --i) if (par[u][i] != par[v][i
        ]) {
      u = par[u][i];
      v = par[v][i];
    }
    return par[u][0];
  }
  int dist(int a, int b) {
    return depth[a] + depth[b] - 2 * depth[query(a, b)];
  }
};
```

## 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:** $\mathcal{O}\left((\log N)^2\right)$

"../data-structures/LazySegmentTree.h" 03139d, 46 lines

```
template <bool VALS_EDGES> struct HLD {
  int N, tim = 0;
  vector<vi> adj;
  vi par, siz, rt, pos;
  Node *tree;
  HLD(vector<vi> adj_)
    : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
      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;
      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 (pos[rt[u]] > pos[rt[v]]) swap(u, v);
      op(pos[rt[v]], pos[v] + 1);
    }
    if (pos[u] > pos[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 = -1e9;
    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};
}
```

# Geometry (7)

## 7.1 Geometric primitives

### Point.h

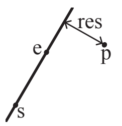
**Description:** Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

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.

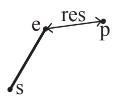


"Point.h" f6bf6b, 4 lines

```
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.



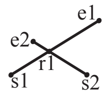
**Usage:** Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;

"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.



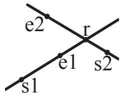
**Usage:** vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at "<< inter[0] << endl;

"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

**Description:**
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. 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 ll.



**Usage:** auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at "<< res.second << endl;

"Point.h" a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

### sideOf.h

**Description:** Returns where $p$ is as seen from $s$ towards $e$. 1/0/-1 ⇔ left/on line/right. If the optional argument *eps* is given 0 is returned if $p$ is within distance *eps* from the line. P is supposed to be Point<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.

**Usage:** bool left = sideOf(p1,p2,q)==1;

"Point.h" f5b12f, 9 lines

```
template<class P>
int sideOf(P startline, P endline, P p) { return sgn(startline.
    cross(endline, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

### OnSegment.h

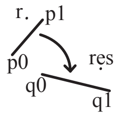
**Description:** Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

### linearTransformation.h

**Description:**



Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

"Point.h" 03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

### Angle.h

**Description:** A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

**Usage:** vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

## 7.2 Circles

### CircleIntersection.h

**Description:** Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

### CircleLine.h

**Description:** Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

"Point.h" e0cfba, 9 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

### CirclePolygonIntersection.h

**Description:** Returns the area of the intersection of a circle with a ccw polygon.

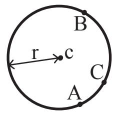
**Time:** $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 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 circumcirle of a triangle (hinh tron ngoai tiep tam giac) 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" 1caa3a, 9 lines

```
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 $\mathcal{O}(n)$

"circumcircle.h" 09dd0a, 17 lines

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

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
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!

"Point.h" f12300, 6 lines

```
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

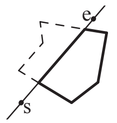
**Description:** Returns the center of mass for a polygon.

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

"Point.h" 9706dc, 9 lines

```
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

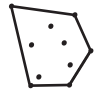


**Description:**
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
**Usage:** `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```
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:** $\mathcal{O}(n \log n)$

"Point.h" 310954, 13 lines

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

"Point.h" c571b8, 12 lines

```
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;
}
```

## 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;
}
```

# 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[i] 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:** $\mathcal{O}(N)$

e7ad79, 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;
}
```

### MinRotation.h

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

d07a42, 8 lines

```
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:** $\mathcal{O}(n \log n)$

769289, 23 lines

```
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)), y(n), ws(max(n, lim)), rank(n);
    x.push_back(0), 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++);
  }
};
```

## Hashing.h

**Description:** Self-explanatory methods for string hashing.

2d2a67, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
typedef uint64_t ull;
struct H {
  ull x; H(ull x=0) : x(x) {}
  H operator+(H o) { return x + o.x + (x + o.x < x); }
  H operator-(H o) { return *this + ~o.x; }
  H operator*(H o) { auto m = (__uint128_t)x * o.x;
    return H((ull)m) + (ull)(m >> 64); }
  ull get() const { return x + !~x; }
  bool operator==(H o) const { return get() == o.get(); }
  bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)

struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};

vector<H> getHashes(string& str, int length) {
  if (sz(str) < length) return {};
  H h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}

H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

## Hashing2d.h

**Description:** String hashing 2d for what?

165853, 35 lines

```
struct Hashing2D {
  vector<vector<int>> hs;
  vector<int> PWX, PWY;
  int n, m;
  static const int PX = 3731,  PY = 2999, mod = 998244353;
  Hashing2D() {}
  Hashing2D(vector<string>& s) {
    n = (int)s.size(), m = (int)s[0].size();
    hs.assign(n + 1, vector<int>(m + 1, 0));
    PWX.assign(n + 1, 1);
    PWY.assign(m + 1, 1);
    for (int i = 0; i < n; i++) PWX[i + 1] = 1LL * PWX[i] * PX
        % mod;
    for (int i = 0; i < m; i++) PWY[i + 1] = 1LL * PWY[i] * PY
        % mod;
    for (int i = 0; i < n; i++)
      for (int j = 0; j < m; j++)
        hs[i + 1][j + 1] = s[i][j] - 'a' + 1;
    for (int i = 0; i <= n; i++)
    for (int j = 0; j < m; j++)
      hs[i][j + 1] = (hs[i][j + 1] + 1LL * hs[i][j] * PY % mod)
          % mod;
    for (int i = 0; i < n; i++)
    for (int j = 0; j <= m; j++)
      hs[i + 1][j] = (hs[i + 1][j] + 1LL * hs[i][j] * PX % mod)
          % mod;
  }
  int get_hash(int x1, int y1, int x2, int y2) { // 1-indexed
    assert(1 <= x1 && x1 <= x2 && x2 <= n);
    assert(1 <= y1 && y1 <= y2 && y2 <= m);
    x1--; y1--;
    int dx = x2 - x1, dy = y2 - y1;
    return (1LL * (hs[x2][y2] - 1LL * hs[x2][y1] * PWY[dy] %
        mod + mod) % mod -
      1LL * (hs[x1][y2] - 1LL * hs[x1][y1] * PWY[dy] % mod +
          mod) % mod * PWX[dx] % mod + mod) % mod;
  }
  int get_hash() {
    return get_hash(1, 1, n, m);
  }
};
```

# Various (9)

## TernarySearch.h

**Description:** Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. 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:** $\mathcal{O}(\log(b-a))$

9155b4, 11 lines

```
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;
}
```

## vuhieuComb.h

**Description:** Calculate combinatorial prepare $O(n)$, query $O(n/ln(n))$, whatever modulo. Để có thể tính được với một modulo bất kì, ta không được sử dụng các biểu thức có phép chia. Vì vậy, ta sẽ phân tích số C(k, n) ra thừa số nguyên tố để được một biểu thức chỉ toàn phép nhân. Áp dụng công thức: $C(k,n) = n!/k!/(n-k)!$ Ta thấy C(k, n) là một phép nhân/chia của các giai thừa. Do đó, ta cần phân tích n! ra TSNT, và phần tích k!, (n-k)! theo cách hoàn toàn tương tự, từ đó ta suy ra được phân tích của C(k, n) Vấn đề bây giờ là làm thế nào để phân tích n! ra thừa số nguyên tố. Như đã biết, với mọi số nguyên tố p, n! chia hết cho p khi và chỉ khi n >= p. Như thế, n! có các ước nguyên tố là mọi số nguyên tố từ 1 đến n. Do đó, với mỗi số nguyên tố p <= n, ta cần tìm số mũ của nó, tức tìm số k lớn nhất để n! chia hết cho $p^k$ Kết quả: số mũ của p trong phân tích ra TSNT của n! là $n/p + n/(p^2) + n/(p^3) + \dots$ Phép chia là phép chia làm tròn xuống, vì vậy, sẽ đến một lúc mẫu số > n, khi đó, phân số có giá trị = 0. Do đó, tổng trên không vô hạn.

b8dc06, 22 lines

```
#define MAX 1000100
bool notPrime[MAX];
vector<int> primes;

//prime sieve
notPrime[0] = notPrime[1] = true;
for (int i = 2; 1LL * i * i < MAX; i++) if (!notPrime[i])
  for (int j = i * i; j < MAX; j += i) notPrime[j] = true;
for (int i = 2; i < MAX; i++) if (!notPrime[i]) primes.
    push_back(i);

int comb(int k, int n) {
  if (k > n) return 0;
  int res = 1;
  for (int p : primes) {
    if (p > n) break;
    int exp = 0; //calcuate p exponentation
    for (long long tmp = p; tmp <= n; tmp *= p)
      exp += n / tmp - k / tmp - (n - k) / tmp;
    res = 1LL * res * pw(p, exp) % MOD;
    }
  return res;
}
```

## 9.1 Dynamic programming

## 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;
}
```

## EditDistance.h

**Description:** Minimum number of operations to transform string a => string b

**Time:** idk

b288d3, 21 lines

```
int edit_distance(string a, string b) {
    int la = a.size();
    int lb = b.size();
    a = " " + a + " ";
    b = " " + b + " ";
    vector<vector<int>> f(la + 1, vector<int> (lb + 1, la + lb)
        );

    for (int j = 0; j <= lb; ++j) f[0][j] = j;
    for (int i = 0; i <= la; ++i) f[i][0] = i;

    for (int i = 1; i <= la; ++i) {
        for (int j = 1; j <= lb; ++j) {
            if (a[i] == b[j]) f[i][j] = f[i-1][j-1];
            else f[i][j] = 1 + min({
                    f[i-1][j-1],  // modify
                    f[i][j-1],    // remove b[j]
                    f[i-1][j]});    // remove a[i]
        }
    }
    return f.back().back();
}
```

### 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;
}
```

### UnboundKnapsack.h

**Description:** Select subset of items, such that sum(weights) <= capacity and sum(values) is maximum. An item can be selected unlimited number of times
**Time:** idk

bf22b7, 11 lines

```
int knapsack_unbounded(int capacity, vector<int> weights,
     vector<int> values) {
    int n = weights.size();
    vector<int> f(capacity + 1, 0);
    for (int i = 0; i < n; ++i) {
        for (int j = weights[i]; j <= capacity; ++j) {
            f[j] = max(f[j], f[j-weights[i]] + values[i]);
        }
    }

    return *max_element(f.begin(), f.end());
}
```

### BoundedKnapsack.h

**Description:** ps-profits, ws-weights, ms-maximum limit of each element W-maximum weight
**Time:** $\mathcal{O}(n * W)$

ecb4aa, 27 lines

```
int boundedKnapsack(vector<int> ps, vector<int> ws, vector<int>
      ms, int W) {
  int n = ps.size();
  vector<vector<int>> dp(n + 1, vector<int>(W + 1));
  for (int i = 0; i < n; ++i) {
    for (int s = 0; s < ws[i]; ++s) {
      int alpha = 0;
      queue<int> que;
      deque<int> peek;
      for (int w = s; w <= W; w += ws[i]) {
        alpha += ps[i];
        int a = dp[i][w] - alpha;
        que.push(a);
        while (!peek.empty() && peek.back() < a) peek.pop_back
            ();
        peek.push_back(a);
        while (que.size() > ms[i] + 1) {
          if (que.front() == peek.front()) peek.pop_front();
          que.pop();
        }
        dp[i + 1][w] = peek.front() + alpha;
      }
    }
  }
  int ans = 0;
  for (int w = 0; w <= W; ++w)
    ans = max(ans, dp[n][w]);
  return ans;
}
```

### DivideAndConquerDP.h

**Description:** Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i,k))$ 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); }
};
```

### DPSOS.h

**Description:** DP SOS

a28187, 9 lines

```
#define MASK(n) (1LL << (n))
#define BIT(x, i) (((x) >> (i)) & 1)
int a[MASK(20) + 2], sum[MASK(20) + 2];

memset(sum, 0, sizeof sum);
for (int mask = 0; mask < MASK(n); mask++) sum[mask] = a[mask];

for (int i = 0; i < n; i++) for (int mask = 0; mask < MASK(n);
     mask++) if (BIT(mask, i))
sum[mask] += sum[mask - MASK(i)];
```

## 9.2 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (`1`), 0-divs (`4`), infinities (`8`) and denormals (`16`).

## 9.3 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

### 9.3.1 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.

### 9.3.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

### 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 - 480;
  return a - 48;
}
```

## Int128Helper.h

**Description:** i128 helper function

712410, 48 lines

```
using i128 = __int128_t;
i128 str2i128(string str) {
    i128 ret = 0;
    bool minus = false;
    for (auto c : str) {
        if (c == '-') minus = true;
        else ret = ret * 10 + c - '0';
    }
    return minus ? -ret : ret;
}
istream &operator>>(istream &is, i128 &x) {
    string s;
    return is >> s, x = str2i128(s), is;
}
ostream &operator<<(ostream &os, const i128 &x) {
    i128 tmp = x;
    if (tmp == 0) return os << 0;
    vector<int> ds;
    if (tmp < 0) {
        os << '-';
        while (tmp) {
            int d = tmp % 10;
            if (d > 0) d -= 10;
            ds.emplace_back(-d), tmp = (tmp - d) / 10;
        }
    } else {
        while (tmp) ds.emplace_back(tmp % 10), tmp /= 10;
    }
    reverse(ds.begin(), ds.end());
    for (auto i : ds) os << i;
    return os;
}
i128 my_abs(i128 n) {
    if (n < 0) return -n;
    return n;
}
i128 gcd(i128 a, i128 b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
int ctz128(i128 n) {    // Count trailing zeroes
    if (!n) return 128;
    if (!static_cast<uint64_t>(n)) {
        return __builtin_ctzll(static_cast<uint64_t>(n >> 64))
            + 64;
    } else {
        return __builtin_ctzll(static_cast<uint64_t>(n));
    }
}
```

## IncreaseStackSize.h

**Description:** tang stack hack

2c806a, 21 lines

```
void main_() {
    // implement your solution here
}
static void run_with_stack_size(void (*func)(void), size_t
     stsize) {
    char *stack, *send;
    stack = (char *)malloc(stsize);
    send = stack + stsize - 16;
    send = (char *)((uintptr_t)send / 16 * 16);
    asm volatile(
        "mov %%rsp, (%0)\n"
        "mov %0, %%rsp\n"
        :
        : "r"(send));
    func();
    asm volatile("mov (%0), %%rsp\n" : : "r"(send));
    free(stack);
}
int main() {
    run_with_stack_size(main_, 1024 * 1024 * 1024); // run with
         a 1 GiB stack
    return 0;
}
```

## SIMD.h

**Description:** Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern "_mm(256)?_name_(si(128|256)|epi(8|16|32|64)|pd|ps)". Not all are described here; grep for _mm_ in /usr/lib/gcc/*/4.9/include/ for more. If AVX is unsupported, try 128-bit operations, "emmintrin.h"and #define __SSE__ and __MMX__ before including it. For aligned memory use _mm_malloc(size, 32) or int buf[N] alignas(32), but prefer loadu/storeu.

551b82, 43 lines

```
#pragma GCC target ("avx2") // or sse4.1
#include "immintrin.h"

typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))

// High-level/specific methods:
// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc
// blendv_(epi8|ps|pd) (z?y:x), movemask_epi8 (hibits of bytes)
// i32gather_epi32(addr, x, 4): map addr[] over 32-b parts of x
// sad_epu8: sum of absolute differences of u8, outputs 4xi64
// maddubs_epi16: dot product of unsigned i7's, outputs 16xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtsi128_si32 (128->lo32)
// permute2f128_si256(x,x,1) swaps 128-bit lanes
// shuffle_epi32(x, 3*64+2*16+1*4+0) == x for each lane
// shuffle_epi8(x, y) takes a vector instead of an imm

// Methods that work with most data types (append e.g. _epi32):
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,
// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|hi)

int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
  int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }

ll example_filteredDotProduct(int n, short* a, short* b) {
  int i = 0; ll r = 0;
  mi zero = _mm256_setzero_si256(), acc = zero;
  while (i + 16 <= n) {
    mi va = L(a[i]), vb = L(b[i]); i += 16;
    va = _mm256_and_si256(_mm256_cmpgt_epi16(vb, va), va);
    mi vp = _mm256_madd_epi16(va, vb);
    acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
      _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)));
  }
  union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[i];
  for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <- equiv
  return r;
}
```