Chinese University of Hong Kong, Shenzhen

???

adapted from KTH ACM Contest Template Library

Contest (1)

base.hpp

<bits/stdc++.h> 25 lines

```
using std::abs, std::sin, std::cos, std::tan, std::asin,
    ↪std::acos, std::atan2;
using std::min, std::max, std::swap;
using std::pair, std::tuple;
using std::set, std::map, std::multiset;
using std::tie;
using std::vector, std::array, std::string;

template <class T> using Vec = vector<T>;
template <class T> using Opt = std::optional<T>;

using i8 = int8_t;
using u8 = uint8_t;
using i32 = int32_t;
using i64 = int64_t;
using u32 = uint32_t;
using u64 = uint64_t;
using i128 = __int128_t;
using u128 = __uint128_t;

inline std::mt19937_64 mt(
  std::chrono::steady_clock::now().time_since_epoch().count
    ↪());

template <class T> T rand_int(T l, T r) {
  return std::uniform_int_distribution<T>(l, r)(mt);
} // hash-cpp-all = ad2a9a13becc0025e4b88cd15efc960b
```

bashrc

5 lines

```
setxkbmap -option caps:escape
alias e='vim'
alias cls='clear -x'
alias mv='mv -i'
alias cp='cp -i'
```

hash-cpp.sh

1 lines

```
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum
```

Makefile

3 lines

```
CXXFLAGS = -O2 -std=gnu++20 -Wall -Wextra -Wno-unused-
    ↪result -pedantic -Wshadow -Wformat=2 -Wfloat-equal -
    ↪Wconversion -Wlogical-op -Wshift-overflow=2 -
    ↪Wduplicated-cond -Wcast-qual -Wcast-align
DEBUGFLAGS = -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC -
    ↪fsanitize=address -fsanitize=undefined -fno-sanitize-
    ↪recover=all -fstack-protector -D_FORTIFY_SOURCE=2
CXXFLAGS += $(DEBUGFLAGS) # flags with speed penalty
```

yc.hpp

<functional> 17 lines

```
namespace std {

template <class F> struct y_combinator_result {
  F f;
  template <class T>
  explicit y_combinator_result(T&& f_) : f(std::forward<T>(
    ↪f_)) {}
  template <class... A> decltype(auto) operator()(A&&... a)
    ↪ {
    return f(std::ref(*this), std::forward<A>(a)...);
  }
};

template <class F> decltype(auto) y_combinator(F&& f) {
  return y_combinator_result<std::decay_t<F>>(std::forward<
    ↪F>(f));
}

}  // namespace std
// hash-cpp-all = 98db09b175ca877a9ee82c609d4312a6
```

fast-input.hpp

Description: Fast scanner implementation based on fread

59 lines

```
namespace fast_input {

struct Scanner {
  FILE* f;
  Scanner(FILE* f_ = stdin) : f(f_) {}

  char get() { // hash-cpp-1
    static array<char, 1 << 16> buf;
    static size_t s = 0, e = 0;
    if (s >= e) {
      buf[0] = 0;
      s = 0;
      e = fread(data(buf), 1, sizeof(buf), f);
    }
    return buf[s++];
  } // hash-cpp-1 = dbac8c21422ef521045397b89e192021

  using Self = Scanner;

  char skip_whitespaces() {
    char c;
    while ((c = get()) <= ' ') {
    }
    return c;
  }

  template <class T> Self& operator>>(T& x) {
    char c = skip_whitespaces();
    bool neg = false;
    if (c == '-') {
      neg = true;
      c = get();
    }
    x = 0;
    do {
      x = 10 * x + (c & 15);
    } while ((c = get()) >= '0');
    if (neg) x = -x;
    return *this;
  }

  Self& operator>>(string& x) {
    char c = skip_whitespaces();
    x = {};
    do {
      x += c;
    } while ((c = get()) > ' ');
    return *this;
  }

  Self& operator>>(double& x) {
    string z;
    *this >> z;
    x = stod(z);
    return *this;
  }
};

}  // namespace fast_input
```

Data Structure (2)

hash-map.hpp

Description: Faster and safer hash map. For some key type K other than uint64_t, define a custom hash function that maps K to uint64_t

<ext/pb_ds/assoc_container.hpp> 14 lines

```
struct CustomHash { // hash-cpp-1
  u64 operator()(u64 x) const {
    static const u64
      z = std::chrono::steady_clock::now().time_since_epoch
        ↪().count(),
      c = u64(4e18 * acos(0)) + 71;
    return u64(__builtin_bswap64((x ^ z) * c));
  }
}; // hash-cpp-1 = 6ca7abe4a2d489e8dcdf809aad4c93c1

template <class K, class V, class Hash = CustomHash>
using HashMap = __gnu_pbds::gp_hash_table<K, V, Hash>;

template <class K, class Hash = CustomHash>
using HashSet = HashMap<K, __gnu_pbds::null_type, Hash>;
```

binary-indexed-tree.hpp

Description: Supports computing partial sum $a_0 + \ldots + a_{i-1}$ and incrementing some a_i by v

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

45 lines

```
template <class T> struct BIT {
  Vec<T> x;
  int s, w;
  BIT(int n) { build(n); }
  BIT(const Vec<T>& a) { build(a); }

  void build(int n) { // hash-cpp-1
    x.clear();
    x.resize(s = n);
    w = std::bit_width<u32>(s) - 1;
  } // hash-cpp-1 = d609ae73bb14759f097e750981a47c31

  void build(const Vec<T>& a) { // hash-cpp-2
    build(int(a.size()));
    copy(a.begin(), a.end(), x.begin());
    for (int i = 0; i < s; i++) {
      int j = i | (i + 1);
      if (j < s) x[j] += x[i];
    }
  } // hash-cpp-2 = 40280f94a7097b2d70d078828d1ba56d

  void add(int i, T v) { // hash-cpp-3
    for (; i < s; i |= i + 1) x[i] += v;
  }
  T sum(int i) {
    T res = 0;
    for (; i; i &= i - 1) res += x[i - 1];
    return res;
  } // hash-cpp-3 = e7fbe70df2a7ecfa13485bb1c017438a
```

```
  // Slightly tested; requires s >= 1
  int kth(T k) { // hash-cpp-4
    int cur = 0;
    for (int i = w; i >= 0; i--) {
      int nxt = cur + (1 << i);
      if (nxt <= s && x[nxt - 1] <= k) {
        k -= x[nxt - 1];
        cur = nxt;
      }
    }
    return cur;
  } // hash-cpp-4 = 788c41fbea7c5755e3df0caae1249411

  int kth_helper(T k, int i = 0) { return kth(k + sum(i));
    ↪}
};
```

lazy-segtree.hpp

Description: Lazy segtree abstraction

167 lines

```
template <class M> struct LazySegtree {
  using S = M::S;
  using F = M::F;
  M m;
  Vec<S> d;
  Vec<F> lz;
  int n, h, sz;
  LazySegtree(M m_) : m(m_), n(0), h(0), sz(0) {}
  template <class A> LazySegtree(int n_, A a, M m_) : m(m_)
    ↪ { build(n_, a); }
  template <class A> void build(int n_, A a) { // hash-cpp
    ↪-1
    n = n_;
    sz = std::bit_ceil<uint32_t>(n);
    h = std::countr_zero<uint32_t>(sz);
    d.resize(2 * sz);
    lz.assign(sz, m.id());
    for (int i = 0; i < n; i++) d[sz + i] = a(i);
    for (int i = n; i < sz; i++) d[sz + i] = m.e();
    for (int i = sz - 1; i >= 1; i--) update(i);
  } // hash-cpp-1 = 3daff936b4ff25e69bacb710b05a4914

  void update(int i) { // hash-cpp-2
    d[i] = m.op(d[2 * i], d[2 * i + 1]);
  } // hash-cpp-2 = 353f7580bfd321bdccddd446692b7f8b

  void apply(int i, F f) { // hash-cpp-3
    d[i] = m.mapping(f, d[i]);
    if (i < sz) lz[i] = m.composition(lz[i], f);
  } // hash-cpp-3 = 066198e6507bd0fb1d8f62457b912fee

  void downdate(int i) { // hash-cpp-4
    apply(2 * i, lz[i]);
    apply(2 * i + 1, lz[i]);
    lz[i] = m.id();
  } // hash-cpp-4 = 46a017e02b26c704289940242c450305

  void downdate_range(int l, int r) { // hash-cpp-5
    l += sz, r += sz;
    for (int i = h; i >= 1; i--) {
      if (((l >> i) << i) != l) downdate(l >> i);
      if (((r >> i) << i) != r) downdate((r - 1) >> i);
    }
  } // hash-cpp-5 = 740eb7bc3b5128e2958ac01b4a1b1814

  S prod(int l, int r) { // hash-cpp-6
    assert(0 <= l && l <= r && r <= n);
    if (l == r) return m.e();
    downdate_range(l, r);
    S sl = m.e(), sr = m.e();
    for (int a = l + sz, b = r + sz; a < b; a /= 2, b /= 2)
      ↪ {
      if (a & 1) sl = m.op(sl, d[a++]);
      if (b & 1) sr = m.op(d[--b], sr);
    }
    return m.op(sl, sr);
  } // hash-cpp-6 = a59327a4ea4e2789d70fbf683619e523

  void apply(int l, int r, F f) { // hash-cpp-7
    assert(0 <= l && l <= r && r <= n);
    if (l == r) return;
    downdate_range(l, r);
    l += sz, r += sz;
    for (int a = l, b = r; a < b; a /= 2, b /= 2) {
      if (a & 1) apply(a++, f);
      if (b & 1) apply(--b, f);
    }
    for (int i = 1; i <= h; i++) {
      if (((l >> i) << i) != l) update(l >> i);
      if (((r >> i) << i) != r) update((r - 1) >> i);
    }
  } // hash-cpp-7 = 655465247dd934e37768c858108371fc

  // You can use this to query stuff,
  // which is sometimes more efficient than using prod
  template <class G> void enumerate(int l, int r, G g) { //
    ↪ hash-cpp-8
    assert(0 <= l && l <= r && r <= n);
    if (l == r) return;
    downdate_range(l, r);
    for (int a = l + sz, b = r + sz; a < b; a /= 2, b /= 2)
      ↪ {
      if (a & 1) g(d[a++]);
      if (b & 1) g(d[--b]);
    }
  } // hash-cpp-8 = 516415088e3e5ad3a49dbc0c0935faab

  // Enumerating in some sequential order
  template <bool l_to_r = true, class G>
  void enumerate_in_order(int l, int r, G g) {
    assert(0 <= l && l <= r && r <= n);
    if (l == r) return; // hash-cpp-9
    downdate_range(l, r);
    static Vec<int> ls, rs;
    ls.clear(), rs.clear();
    for (int a = l + sz, b = r + sz; a < b; a /= 2, b /= 2)
      ↪ {
      if (a & 1) ls.push_back(a++);
      if (b & 1) rs.push_back(--b);
    } // hash-cpp-9 = 2481fb42166bf39d0da2499c3e727a6d
    if constexpr (l_to_r) {
      for (int i : ls) g(d[i]);
      for (int z = int(rs.size()) - 1; z >= 0; z--) g(d[rs[
        ↪z]]);
    } else {
      for (int i : rs) g(d[i]);
      for (int z = int(ls.size()) - 1; z >= 0; z--) g(d[ls[
        ↪z]]);
    }
  }

  const S& all_prod() const { return d[1]; }

  template <class P> pair<int, S> max_right(int l, P p) {
    ↪// hash-cpp-10
    assert(0 <= l && l <= n);
    if (l == n) return {n, m.e()};
    l += sz;
    for (int i = h; i >= 1; i--) downdate(l >> i);
    S s = m.e();
    assert(p(s));
    do {
      while (l % 2 == 0) l /= 2;
      if (!p(m.op(s, d[l]))) {
        while (l < sz) {
          downdate(l);
          l = 2 * l;
          S t = m.op(s, d[l]);
          if (p(t)) {
            s = t;
            l++;
          }
        }
        return {l - sz, s};
      }
      s = m.op(s, d[l]);
      l++;
    } while ((l & -l) != l);
    return {n, s};
  } // hash-cpp-10 = 659b16e053dcfd226edd2f7354d3c75c

  template <class P> pair<int, S> min_left(int r, P p) { //
    ↪ hash-cpp-11
    assert(0 <= r && r <= n);
    if (r == 0) return {0, m.e()};
    r += sz;
    for (int i = h; i >= 1; i--) downdate((r - 1) >> i);
    S s = m.e();
    assert(p(s));
    do {
      r--;
      while (r > 1 && r % 2) r /= 2;
      if (!p(m.op(d[r], s))) {
        while (r < sz) {
          downdate(r);
          r = 2 * r + 1;
          S t = m.op(d[r], s);
          if (p(t)) {
            s = t;
            r--;
          }
        }
        return {r + 1 - sz, s};
      }
      s = m.op(d[r], s);
    } while ((r & -r) != r);
    return {0, s};
  } // hash-cpp-11 = 679cc146eea81abf054b473f1e991349

  void set(int p, S s) { // hash-cpp-12
    assert(0 <= p && p < n);
    p += sz;
    for (int i = h; i >= 1; i--) downdate(p >> i);
    d[p] = s;
    for (int i = 1; i <= h; i++) update(p >> i);
  } // hash-cpp-12 = eee80c946397620fdc779230722e1655
};
```

static-range.hpp

Description: Static range composition. You need to specify a compositition function f and an identity element e

Time: $\mathcal{O}(N \log N)$ building and $\mathcal{O}(1)$ querying

34 lines

```
template <class T, class F> struct StaticRange {
  Vec<Vec<T>> d; // hash-cpp-1
  const F f;
  const T e;
  StaticRange(const Vec<T>& a, F f_, T e_) : f(f_), e(e_) {
    int n = int(size(a));
    int h = 0;
    while ((2 << h) < n) h++;
    d.resize(h + 1);
    d[0] = a;
    for (int k = 0; k < h; k++) {
      d[k + 1].resize(n, e);
      int s = 1 << (k + 1);
      for (int i = s; i < n; i += 2 * s) {
        T x = e;
        for (int j = i - 1; j >= i - s; j--) {
          d[k + 1][j] = x = f(a[j], x);
        }
        x = e;
        for (int j = i; j < i + s && j < n; j++) {
          d[k + 1][j] = x = f(x, a[j]);
        }
      }
    }
  } // hash-cpp-1 = 6a493be3848c7679ff694dbec308c49d

  T prod(int l, int r) const { // hash-cpp-2
    if (l >= r) return e;
    r--;
    if (l == r) return d[0][l];
    int k = std::bit_width<u32>(l ^ r) - 1;
    return f(d[k][l], d[k][r]);
  } // hash-cpp-2 = d9562651cbcb96c76a7b3ab940158907
};
```

treap.hpp

Description: Randomized Treap with split/merge support. `nodes.size() < nodes.capacity()` must be maintained. One strategy to save space is to refactor everything when the size of `nodes` is approximating its capacity

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

202 lines

```
template <class M, bool persistent = false> struct
  ↪TreapManager {
  using S = M::S;
  using F = M::F;

  TreapManager(M m_, int alloc = 0) : m(m_) {
    if (alloc > 0) {
      nodes.reserve(alloc);
    } else {
      // make sure to understand what you're doing
      assert(!persistent);
    }

    for (int z = 0; z < 2; z++) {
      states[z] = u32(mt());
    }
  }

  using Tree = int;

  Tree make_empty() { return Tree(null); }

  Tree make_single(S s) { // hash-cpp-1
    int i = int(nodes.size());
    nodes.push_back(Node{null, null, 1, false, false, s, s,
      ↪ m.id()});
    return i;
  } // hash-cpp-1 = 6c4d20b86ebfc6f60d88165b76573a67

  Tree make_copy(Tree o) { return _make_copy(o); }

  int size(const Tree t) { return _size(t); }
  int reverse(Tree t) { return _reverse(t); }
  int apply(Tree t, F f) { return _apply(t, f); }
  S prod(const Tree& t) { return _prod(t); }

  Tree split_k(Tree& t, int k) { // hash-cpp-2
    Tree o;
    tie(t, o) = _split_k(t, k);
    return o;
  } // hash-cpp-2 = c70f87700806d15a4c4ec662572f17ff

  Tree merge(Tree a, Tree b) { return _merge(a, b); }

  Tree build(const Vec<S>& a) { // hash-cpp-3
    if (a.empty()) return make_empty();
    return _build(a, 0, int(a.size()));
  } // hash-cpp-3 = d5774c15e3b5b571de7d737f390da619

  Vec<S> to_array(const Tree& t) { // hash-cpp-4
    Vec<S> buf;
    buf.reserve(size(t));
    _to_array(t, buf);
    return buf;
  } // hash-cpp-4 = 7367030dad11dcd4f5db83533a4b3d26

private:
  static constexpr int null = -42;
  M m;

  struct Node { // hash-cpp-5
    int li, ri, sz;
    bool rev, app;
    S a, s;
    F f;
  };
  Vec<Node> nodes;
  Node& node(int i) { return nodes[i]; }
  int _size(int i) { return i == null ? 0 : node(i).sz; }
    ↪// hash-cpp-5 = 7ff1fec7f9265acee7e49866a73a5d75

  int _make_copy(int o) { // hash-cpp-6
    if constexpr (!persistent) return o;

    if (o == null) return null;
    assert(nodes.size() < nodes.capacity());
    int i = int(nodes.size());
    nodes.push_back(node(o));
    return i;
  } // hash-cpp-6 = 26a70edec35d6f656b6f85d49ceb2fc6

  int _build(const Vec<S>& a, int l, int r) { // hash-cpp-7
    if (r - l == 1) {
      return make_single(a[l]);
    }
    int md = (l + r) / 2;
    return _merge(_build(a, l, md), _build(a, md, r));
  } // hash-cpp-7 = 5b1df26f9cad8f5588e7f963e3252ea4

  void _update(int i) { // hash-cpp-8
    auto& n = node(i);
    n.s = m.op(_prod(n.li), m.op(n.a, _prod(n.ri)));
    n.sz = size(n.li) + size(n.ri) + 1;
  } // hash-cpp-8 = c5fb7048740c35c2a720845684e4ff19

  int _reverse(int i) { // hash-cpp-9
    if (i == null) return i;
    i = _make_copy(i);
    auto& n = node(i);
    n.rev = !n.rev;
    swap(n.li, n.ri);
    return i;
  } // hash-cpp-9 = 266d7203b1c04371492ea0bd85cb281d

  S _prod(int i) { return i == null ? m.e() : node(i).s; }

  int _apply(int i, F f) { // hash-cpp-10
    if (i == null) return i;
    i = _make_copy(i);
    auto& n = node(i);
    n.s = m.mapping_sz(f, n.s, n.sz);
    n.a = m.mapping_sz(f, n.a, 1);
    n.f = m.composition(f, n.f);
    n.app = true;
    return i;
  } // hash-cpp-10 = c1044aa4c9dbe3605f7e255c9ef1131b

  int downdate(int i) { // hash-cpp-11
    assert(i != null);
    i = _make_copy(i);
    auto& n = node(i);
    if (n.rev) {
      n.li = _reverse(n.li);
      n.ri = _reverse(n.ri);
      n.rev = false;
    }
    if (n.app) {
      n.li = _apply(n.li, n.f);
      n.ri = _apply(n.ri, n.f);
      n.f = m.id();
      n.app = false;
    }
    return i;
  } // hash-cpp-11 = de62225a6441397fe26f3bdae0f19423

  template <class F> pair<int, int> _split(int i, F go_left
    ↪) { // hash-cpp-12
    if (i == null) return {null, null};
    i = downdate(i);
    auto& n = node(i);
    int li = n.li, ri = n.ri;
    int x, y;
    if (go_left(li, ri)) {
      y = i;
      tie(x, n.li) = _split(n.li, go_left);
    } else {
      x = i;
      tie(n.ri, y) = _split(n.ri, go_left);
    }
    _update(i);
    return {x, y};
  } // hash-cpp-12 = 3162351f3f2db4155104ab28b68b8e49

  pair<int, int> _split_k(int i, int k) { // hash-cpp-13
    return _split(i, [&](int li, int) -> bool {
      int lsz = size(li);
      if (k <= lsz) {
        return true;
      } else {
```

```
      k -= lsz + 1;
      return false;
    }
  });
} // hash-cpp-13 = 21661461b27eeb90e1e770dacc49c006

// Use std::mt19937_64 if performance is not an issue
// https://prng.di.unimi.it/xoroshiro64star.c
inline u32 rotl(const u32 x, int k) { // hash-cpp-14
  return (x << k) | (x >> (32 - k));
}
u32 states[2];
u32 rng() {
  const u32 s0 = states[0];
  u32 s1 = states[1];
  const u32 res = s0 * 0x9E3779BB;
  s1 ^= s0;
  states[0] = rotl(s0, 26) ^ s1 ^ (s1 << 9);
  states[1] = rotl(s1, 13);
  return res;
} // hash-cpp-14 = e7808fea1f575341ec66945f5eb60d5a

int _merge(int a, int b) { // hash-cpp-15
  if (a == null) return b;
  if (b == null) return a;
  int r;
  u32 sa = size(a), sb = size(b);
  if (rng() % (sa + sb) < sa) {
    r = downdate(a);
    node(r).ri = _merge(node(r).ri, b);
  } else {
    r = downdate(b);
    node(r).li = _merge(a, node(r).li);
  }
  _update(r);
  return r;
} // hash-cpp-15 = 5e3944c92c44935fc0a83a6a0cdeb76f

void _to_array(int i, Vec<S>& buf) { // hash-cpp-16
  if (i == null) return;
  downdate(i);
  auto& n = node(i);
  _to_array(n.li, buf);
  buf.push_back(n.a);
  _to_array(n.ri, buf);
} // hash-cpp-16 = f2ee73067be10b96ad2b205b24626251
};
```

queue-aggregation.hpp

Description: A queue that supports querying the compositition of all elements

39 lines

```
template <class T, class F> struct QueueAggregation {
  const F f; // hash-cpp-1
  const T e;
  Vec<T> as, bs, ae, be;
  T vs, ve;
  QueueAggregation(F f_, T e_) : f(f_), e(e_), vs(e), ve(e)
    ↪ {} // hash-cpp-1 = aa12ea64acbdf59b8b481d300dcebc03

  void push_s(const T& x) { // hash-cpp-2
    as.push_back(x), bs.push_back(vs = f(x, vs));
  }
  void push_e(const T& x) { ae.push_back(x), be.push_back(
    ↪ve = f(ve, x)); }
  void reduce() {
    while (!ae.empty()) {
      push_s(ae.back()), ae.pop_back();
    }
    be.clear();
    ve = e;
  } // hash-cpp-2 = 8fa4388f714c1fcf480662f94acb94d7

  bool empty() const { // hash-cpp-3
    return as.empty() && ae.empty();
  }
  int size() const { return int(as.size() + ae.size()); }
    ↪// hash-cpp-3 = b5166973f8a1e060551da48002d67335

  void push(const T& x) { // hash-cpp-4
    if (as.empty()) {
      push_s(x), reduce();
    } else {
      push_e(x);
    }
  }
  void pop() {
    assert(!empty());
    if (as.empty()) reduce();
    as.pop_back(), bs.pop_back();
    vs = (bs.empty() ? e : bs.back());
  }
  T prod() const { return f(vs, ve); } // hash-cpp-4 = 0
    ↪b46cd5fba53f4c166094224da58ee1c
};
```

line-container.hpp

Description: Container where you can add lines of the form $y = kx+m$, and query maximum values at given points. Useful for dynamic programming ("convex hull trick")
Time: $\mathcal{O}(\log N)$ with a large constant factor

42 lines

```
namespace line_container {

struct Line { // hash-cpp-1
  mutable i64 k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(i64 x) const { return p < x; }
}; // hash-cpp-1 = fe34d12ba12e83886abda0a6086b3ea0

struct LineContainer : multiset<Line, std::less<>> {
  using I = iterator; // hash-cpp-2
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const i64 inf = std::numeric_limits<i64>::max();
  static i64 div(i64 a, i64 b) {
    return a / b - ((a ^ b) < 0 && a % b);
  } // hash-cpp-2 = 916c6b8fae9c3a6ff292036f8a529685
  bool isect(I x, I y) { // hash-cpp-3
    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;
  } // hash-cpp-3 = dec9ff4585adbee96b3f9592b3614988
  void add(i64 k, i64 m) { // hash-cpp-4
    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));
    }
  } // hash-cpp-4 = 78c5a4da92215ce013230b8b18572988
  i64 query(i64 x) { // hash-cpp-5
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  } // hash-cpp-5 = 368705d894929cc338d6d2732483f777
};

}  // namespace line_container
```

persistent-array.hpp

Description: Persistent array based on persistent segtrees

69 lines

```
template <class D> struct PersistentArray {
  union N { // hash-cpp-1
    D v;
    array<int, 2> c;
    N(const D& a) : v(a) {}
    N(int a, int b) : c{a, b} {}
  };
  Vec<N> x;
  int s, h;
  // Modify this so that it can reserve memory for x
  PersistentArray() {} // hash-cpp-1 = 1
    ↪ff3a53ab6ec6894dd8830d2abad7b10

  // All arrays share the same layout (length)
  int build(int n) { // hash-cpp-2
    x.clear();
    s = 1, h = 0;
    while (s < n) {
      s *= 2;
      h++;
    }
    int rt = make_leaf(D());
    for (int l = 0; l < h; l++) {
      rt = make_node(rt, rt);
    }
    return rt;
  } // hash-cpp-2 = 07caee6062571a915772221c203141f3

  int make_leaf(const D& a) { // hash-cpp-3
    x.emplace_back(a);
    return int(x.size()) - 1;
  }
  int make_node(int a, int b) {
    x.emplace_back(a, b);
    return int(x.size()) - 1;
  } // hash-cpp-3 = 1fee63ccaf8114c5295fe73f218cc786

  int set(int rt, int i, const D& a) { // hash-cpp-4
    static int buf[40];
    for (int l = 0; l < h; l++) {
      buf[l] = rt;
      if ((i >> (h - 1 - l)) & 1) {
        rt = x[rt].c[1];
      } else {
        rt = x[rt].c[0];
      }
    }
    int res = make_leaf(a);
    for (int l = h - 1; l >= 0; l--) {
      int j = buf[l];
      if ((i >> (h - 1 - l)) & 1) {
        res = make_node(x[j].c[0], res);
      } else {
```

```
      res = make_node(res, x[j].c[1]);
    }
  }
  return res;
} // hash-cpp-4 = ce571ab8758dbbaf6d393f0545a71302

D get(int rt, int i) { // hash-cpp-5
  for (int l = h - 1; l >= 0; l--) {
    if (i & (1 << l)) {
      rt = x[rt].c[1];
    } else {
      rt = x[rt].c[0];
    }
  }
  return x[rt].v;
} // hash-cpp-5 = 3a880dd33ae85a7becf12470a5ee22d6
};
```

fast-set.hpp

Description: A set for insertion, removal and querying the predecessor/successor of some element

85 lines

```
struct FastSet {
  static constexpr int B = 64;

  int n, h;
  Vec<Vec<u64>> x;
  FastSet(int n_ = 0) : n(n_) {
    int m = (n ? n : 1);
    do {
      x.push_back(Vec<u64>((m + B - 1) / B));
      m = (m + B - 1) / B;
    } while (m > 1);
    h = int(size(x));
  } // hash-cpp-0 = d41d8cd98f00b204e9800998ecf8427e

  bool empty() const { // hash-cpp-1
    return !x[h - 1][0];
  }
  bool operator[](int i) const {
    return (x[0][i / B] >> (i % B) & 1) != 0;
  } // hash-cpp-1 = 757daa5e083e34270c6abb210a3bcdae

  void set(int i) { // hash-cpp-2
    for (int d = 0; d < h; d++) {
      int q = i / B, r = i % B;
      x[d][q] |= u64(1) << r;
      i = q;
    }
  } // hash-cpp-2 = f800fd0cfa9df69d8679dc495e0432fd

  void reset(int i) { // hash-cpp-3
    for (int d = 0; d < h; d++) {
      int q = i / B, r = i % B;
      x[d][q] &= ~(u64(1) << r);
      if (x[d][q]) break;
      i = q;
    }
  } // hash-cpp-3 = 964743f280b87681157a390bd7fd5449

  // min active j s.t. j >= i
  int next(int i) const { // hash-cpp-4
    if (i >= n) return n;
    i = max(i, 0);
    for (int d = 0; d < h; d++) {
      int q = i / B, r = i % B;
      if (q >= int(size(x[d]))) break;
      u64 up = x[d][q] >> r;
      if (up) {
        i += std::countr_zero(up);
        for (int e = d - 1; e >= 0; e--) {
          i = i * B + std::countr_zero(x[e][i]);
        }
        return i;
      }
      i = q + 1;
    }
    return n;
  } // hash-cpp-4 = 744dbbecee0c3e9dd0bac58360a1875c

  // max active j s.t. j <= i
  int prev(int i) const { // hash-cpp-5
    if (i < 0) return -1;
    i = min(i, n - 1);
    for (int d = 0; d < h; d++) {
      if (i < 0) break;
      int q = i / B, r = i % B;
      u64 lo = x[d][q] << (B - 1 - r);
      if (lo) {
        i -= std::countl_zero(lo);
        for (int e = d - 1; e >= 0; e--) {
          i = i * B + B - 1 - std::countl_zero(x[e][i]);
        }
        return i;
      }
      i = q - 1;
    }
    return -1;
  } // hash-cpp-5 = 37e66d1d428168f7250d08686528b97b

  // not tested
  template <class F> void enumerate(int l, int r, F f) {
    for (int p = next(l); p < r; p = next(p + 1)) {
      f(p);
    }
  }
};
```

Ad Hoc (3)

monotone-minima.hpp

Description: Given an $N \times M$ matrix A, returns $m_i = \operatorname{argmin}_j A_{i,j}$ given that $m_0, \ldots, m_{N-1}$ is non-decreasing

17 lines

```
// f(i, j, k) := [A_{i, j} <= A_{i, k}], given j < k
template <class F> Vec<int> monotone_minima(int n, int m, F
    ↪ f) {
  auto res = Vec<int>(n);
  auto inner = [&](auto self, int s, int e, int l, int r) {
    if (s == e) return;
    int i = (s + e) / 2;
    int b = l;
    for (int k = l + 1; k < r; k++) {
      if (!f(i, b, k)) b = k;
    }
    res[i] = b;
    self(self, s, i, l, b + 1);
    self(self, i + 1, e, b, r);
  };
  inner(inner, 0, n, 0, m);
  return res;
} // hash-cpp-all = 74852d91f028814bde26cc235dcac6bb
```

min-plus-convex.hpp

Description: Given $a_0, \ldots, a_{N-1}$ and $b_0, \ldots, b_{M-1}$ such that $a_{i+1} - a_i \le a_{i+2} - a_{i+1}$, returns $c_0, \ldots, c_{(N-1)+(M-1)}$ such that $c_k = \min_{i+j=k} a_i + b_j$

"ad-hoc/monotone-minima.hpp" 15 lines

```
// a convex and b arbitrary
template <class T> Vec<T> min_plus_convex(const Vec<T>& a,
    ↪const Vec<T>& b) {
  int n = int(size(a)), m = int(size(b));
  if (!n || !m) return {};
  auto x = monotone_minima(n + m - 1, m, [&](int i, int j,
      ↪int k) -> bool {
    if (i < k) return true;
    if (i - j >= n) return false;
    return a[i - j] + b[j] <= a[i - k] + b[k];
  });
  auto res = Vec<T>(n + m - 1);
  for (int i = 0; i < n + m - 1; i++) {
    res[i] = a[i - x[i]] + b[x[i]];
  }
  return res;
} // hash-cpp-all = 61c18c03ecb8ff250898af56d7c09e07
```

floor-ceil-range.hpp

28 lines

```
inline void floor_range(i64 n, auto f) {
  int rt = int(sqrt(double(n))); // hash-cpp-1
  int num = (i64(rt) * rt + rt <= n ? rt : rt - 1);
  i64 prv = n + 1;
  for (int q = 1; q <= num; q++) {
    i64 x = i64(double(n) / (q + 1)) + 1;
    f(q, x, prv);
    prv = x;
  }
  for (int l = rt; l >= 1; l--) {
    f(i64(double(n) / l), l, l + 1);
  } // hash-cpp-1 = 93b579b8e33ad19ecbdae71c9d87828d
}

inline void ceil_range(i64 n, auto f) {
  int rt = int(sqrt(double(n))); // hash-cpp-2
  i64 prv = std::numeric_limits<i64>::max();
  for (int q = 1; q <= rt; ++q) {
    i64 x = i64(double(n + q - 1) / q);
    f(q, x, prv);
    prv = x;
  }
  int num = (n <= i64(rt) * rt + rt ? rt : rt + 1);
  if (n == rt * rt) --num;
  for (int l = num; l >= 1; --l) {
    f(i64(double(n + l - 1) / l), l, l + 1);
  } // hash-cpp-2 = fc1cdafe17e28a72208134fdc874de4c
}
```

palindromic-decomp-dp.hpp

Description: CF932G DP

"string/eertree.hpp" 58 lines

```
// dp[j] := sum_{i s.t. [i, j) is palindromic} {dp[i] * x}
template <class S, int sigma, bool even = false>
Vec<S> palindromic_decomp_dp(const Vec<int>& a,
                             auto add,
                             S add_e,
                             auto mul_x,
                             S mul_e) {
  int n = int(a.size()); // hash-cpp-1
  Vec<int> locs(n);
```

```
  Eertree<sigma> et(n);
  for (int i = 0; i < n; i++) {
    assert(0 <= a[i] && a[i] < sigma);
    locs[i] = et.append(a[i]);
  } // hash-cpp-1 = a13e04432b9972a781423e207b1ae08d

  int nnodes = et.size();
  Vec<int> nxt(nnodes);
  nxt[0] = -1;
  if constexpr (even) {
    assert(n % 2 == 0);
    for (int v = 1; v < nnodes; v++) {
      nxt[v] = (et[v].len() % 2 == 0 ? v : nxt[et[v].fail])
        ↪;
    }
  } else {
    iota(nxt.begin() + 1, nxt.end(), 1);
  }

  Vec<int> diff(nnodes, 1e9); // hash-cpp-2
  Vec<pair<int, int>> top(nnodes);
  for (int v = 2; v < nnodes; v++) {
    int w = nxt[et[v].fail];
    int d = et[v].len() - et[w].len();
    diff[v] = d;
    top[v] = (diff[v] == diff[w] ? top[w] : pair<int, int>(
      ↪w, 0));
    top[v].second++;
  } // hash-cpp-2 = 904fb97daaf4a91bd6da446a3dceea9c

  Vec<S> dp(n + 1, add_e), gdp = dp; // hash-cpp-3
  dp[0] = mul_e;
  for (int j = 0; j < n; j++) {
    int v = nxt[locs[j]];
    int i = (j + 1) - et[v].len();
    while (v >= 2) {
      int d = diff[v];
      auto [p, s] = top[v];
      if (s == 1) {
        gdp[i] = dp[i];
      } else {
        gdp[i] = add(gdp[i], dp[i + d * (s - 1)]);
      }
      dp[j + 1] = add(dp[j + 1], mul_x(gdp[i]));
      i += d * s;
      v = p;
    }
  } // hash-cpp-3 = 770718f9348189ea652a30650d5b66bf

  return dp;
}
```

Algebra (4)

modint.hpp

Description: Frees you from writing % mod stuff. This only works with prime modulo numbers that are determined during compile-time

61 lines

```
template <class T> T pow(T a, i64 b) {
  assert(b >= 0);
  T r = 1;
  while (b) {
    if (b & 1) r *= a;
    a *= a;
    b >>= 1;
  }
  return r;
}

template <u32 mod> struct ModInt {
  using mint = ModInt;

  static constexpr u32 m = mod; // hash-cpp-1
  u32 v;
  constexpr ModInt() : v(0) {}
  template <class T> constexpr ModInt(T a) { s(u32(a % m +
    ↪m)); }
  constexpr mint& s(u32 a) {
    v = a < m ? a : a - m;
    return *this;
  }
  friend mint inv(const mint& n) { return pow(n, m - 2); }
    ↪// hash-cpp-1 = 4dece1675e6b05bf2630f4e3f6e64fb3

  mint operator-() const { // hash-cpp-2
    mint res;
    res.v = v ? m - v : 0;
    return res;
  } // hash-cpp-2 = 682e0bd616a7a1b4efedf0025fd9946a

  friend bool operator==(const mint& a, const mint& b) {
    return a.v == b.v;
  } // hash-cpp-3
  friend bool operator!=(const mint& a, const mint& b) {
    return !(a == b);
  } // hash-cpp-3 = b054e0d8dd8962f442fc0071aae9094a

  mint& operator+=(const mint& o) { return s(v + o.v); } //
    ↪ hash-cpp-4
  mint& operator-=(const mint& o) { return s(v + m - o.v);
    ↪}
  mint& operator*=(const mint& o) {
    v = u32(u64(v) * o.v % m);
    return *this;
  }
  mint& operator/=(const mint& o) { return *this *= inv(o);
    ↪ } // hash-cpp-4 = 5f038b9c2be1f65c54a372c65ee72c5b

  friend mint operator+(const mint& a, const mint& b) {
    return mint(a) += b;
  } // hash-cpp-5
  friend mint operator-(const mint& a, const mint& b) {
    ↪return mint(a) -= b; }
  friend mint operator*(const mint& a, const mint& b) {
    ↪return mint(a) *= b; }
  friend mint operator/(const mint& a, const mint& b) {
    return mint(a) /= b;
  } // hash-cpp-5 = e5581458153784a125647e1cbb9747eb

  static constexpr u32 get_mod() { return m; }
  static constexpr mint get_root() {
    if (m == 998244353) return 3;
    if (m == 1053818881) return 2789;
    assert(false);
  }
};
```

nft.hpp

59 lines

```
template <class T> void nft(Vec<T>& a, int n) {
  static Vec<int> rev = {0, 1}; // hash-cpp-1
  static Vec<T> rt(2, 1);
  if (ssize(rt) < n) {
    rev.resize(n);
    for (int i = 0; i < n; i++) {
      rev[i] = (rev[i >> 1] | ((i & 1) * n)) >> 1;
    }
    rt.reserve(n);
    for (int k = int(size(rt)); k < n; k *= 2) {
      rt.resize(2 * k);
      T z = pow(T::get_root(), (T::get_mod() - 1) / (2 * k)
        ↪);
      for (int i = k / 2; i < k; i++) {
        rt[2 * i] = rt[i];
        rt[2 * i + 1] = rt[i] * z;
      }
    }
  } // hash-cpp-1 = cba95331cf1ba99f75ee2fafa229bb40
  int s = std::countr_zero(u32(size(rev)) / n); // hash-cpp
    ↪-2
  for (int i = 0; i < n; i++) {
    int j = rev[i] >> s;
    if (i < j) swap(a[i], a[j]);
  }
  for (int k = 1; k < n; k *= 2) {
    for (int i = 0; i < n; i += 2 * k) {
      auto it1 = begin(a) + i;
      auto it2 = it1 + k;
      for (int j = 0; j < k; j++, ++it1, ++it2) {
        T t = rt[j + k] * *it2;
        *it2 = *it1 - t;
        *it1 += t;
      }
    }
  } // hash-cpp-2 = 1b6c673b5ee9b617060d250f010a7ec4
}

template <class T> void inft(Vec<T>& a, int n) { // hash-
  ↪cpp-3
  T d = inv(T(n));
  for (int i = 0; i < n; i++) a[i] *= d;
  reverse(begin(a) + 1, end(a));
  nft(a, n);
} // hash-cpp-3 = e89b7ab3e8c68c4b0bc3cc4883fe743d

template <class T> Vec<T> multiply(Vec<T> a, Vec<T> b) { //
  ↪ hash-cpp-4
  int n = int(size(a));
  int m = int(size(b));
  if (!n || !m) return {};
  int s = std::bit_ceil<u32>(n + m - 1);
  a.resize(s), nft(a, s);
  b.resize(s), nft(b, s);
  T is = inv(T(s));
  for (int i = 0; i < s; i++) {
    a[i] *= b[i] * is;
  }
  reverse(begin(a) + 1, end(a));
  nft(a, s);
  a.resize(n + m - 1);
  return a;
} // hash-cpp-4 = 4dba4cf3b97f05245a59534493d49529
```

matrix.hpp

Description: Gaussian elimination and stuff. `solve_linearеq` returns the pair (some particular solution, a basis of the null space). Note that this is meant to be a quick reference for linear algebra operations; some ad-hoc modifications are needed to cope with numerical issues.

"algebra/modint.hpp" 108 lines

```
namespace matrix {
```

```cpp
template <bool rref = false, class T>
pair<int, T> sweep(Vec<Vec<T>>& a, int c = -1) {
  int h = int(a.size());
  if (!h) return {0, 0};
  int w = int(a[0].size());

  if (c == -1) c = w; // hash-cpp-1
  int r = 0;
  T det = 1;
  for (int j = 0; j < c; j++) {
    int p = -1;
    for (int i = r; i < h; i++) {
      if (a[i][j] != T(0)) {
        p = i;
        break;
      }
    }
    if (p == -1) {
      det = 0;
      continue;
    }
    if (r != p) {
      det = -det;
      swap(a[r], a[p]);
    }
    auto& ar = a[r];
    det *= ar[j]; // hash-cpp-1 =
      ↪f28a2f4f866a3ba8944ea81aaacdae01

    int is; // hash-cpp-2
    T d = T(1) / ar[j];
    if constexpr (rref) {
      for (int k = j; k < w; k++) {
        ar[k] *= d;
      }
      d = 1;
      is = 0;
    } else {
      is = r + 1;
    } // hash-cpp-2 = 2e7107ced9297d66963c63feb0f864a8

    for (int i = is; i < h; i++) { // hash-cpp-3
      if (i == r) continue;
      auto& ai = a[i];
      if (ai[j] != T(0)) {
        T e = ai[j] * d;
        for (int k = j; k < w; k++) {
          ai[k] -= ar[k] * e;
        }
      }
    }
    r++;
  } // hash-cpp-3 = 3742e1cbeef59a39cf9ae7fdbc754997
  return {r, det};
}

template <class T>
Opt<pair<Vec<T>, Vec<Vec<T>>>> solve_lineareq(Vec<Vec<T>> a
  ↪, Vec<T> b) {
  int h = int(a.size());
  assert(h);
  int w = int(a[0].size());
  for (int i = 0; i < h; i++) a[i].push_back(b[i]); // hash
    ↪-cpp-4
  int r = sweep<true>(a, w).first;
  for (int i = r; i < h; i++) {
    if (a[i][w] != T(0)) return std::nullopt;
  }
  Vec<T> x(w);
  Vec<int> pivot(w, -1);
  int z = 0;
  for (int i = 0; i < r; i++) {
    while (a[i][z] == T(0)) z++;
    x[z] = a[i][w], pivot[z] = i;
  } // hash-cpp-4 = 87f9743ba6a85ce99922eb413906672b
  Vec<Vec<T>> ker; // hash-cpp-5
  for (int j = 0; j < w; j++) {
    if (pivot[j] == -1) {
      Vec<T> v(w);
      v[j] = 1;
      for (int k = 0; k < j; k++) {
        if (pivot[k] != -1) v[k] = -a[pivot[k]][j];
      }
      ker.push_back(v);
    }
  } // hash-cpp-5 = 39e8c67d53dbc75c4490fa63713b3358
  return make_pair(x, ker);
}

template <class T> Vec<Vec<T>> mat_inv(Vec<Vec<T>> a) {
  int n = int(a.size()); // hash-cpp-6
  Vec<Vec<T>> m(n, Vec<T>(2 * n));
  for (int i = 0; i < n; i++) {
    copy(begin(a[i]), end(a[i]), begin(m[i]));
    m[i][n + i] = 1;
  }
  if (sweep<true>(m, n).first != n) return {};
  Vec<Vec<T>> b(n);
  for (int i = 0; i < n; i++) {
    copy(begin(m[i]) + n, end(m[i]), back_inserter(b[i]));
  }
  return b; // hash-cpp-6 = 28
    ↪ff5dbef99143041dbdce2cbdf92b1e
}

template <class T> int mat_rank(Vec<Vec<T>> a) { return
  ↪sweep<true>(a).first; }

template <class T> T mat_det(Vec<Vec<T>> a) { return sweep<
  ↪false>(a).second; }

}  // namespace matrix
```

barrett.hpp

11 lines

```cpp
struct Barrett {
  static constexpr int k = 96;
  const u32 m;
  const u128 bm;
  Barrett(u32 m_) : m(m_), bm(((u128(1) << k) - 1) / m + 1)
    ↪ {}

  u32 reduce(u64 a) const {
    u64 q = u64((a * bm) >> k);
    return u32(a - q * m);
  }
}; // hash-cpp-all = fc333a19a1bc91f7fbba43f8b156f36e
```

Tree (5)

cartesian-tree.hpp

27 lines

```cpp
template <class T> struct CartesianTree {
  int n, root;
  Vec<int> p;
  Vec<array<int, 2>> c;
  CartesianTree(const Vec<T>& a)
    : n(int(size(a))), root(0), p(n, -1), c(n, {-1, -1}) {
    auto stk = Vec<int>{0};
    stk.reserve(n);
    for (int i = 1; i < n; i++) {
      if (a[stk.back()] > a[i]) {
        while (size(stk) >= 2 && a[stk.end()[-2]] > a[i]) {
          stk.pop_back();
        }
        if (size(stk) == 1) {
          root = p[c[i][0] = stk.back()] = i;
        } else {
          p[c[i][0] = stk.back()] = i;
          c[p[i] = stk.end()[-2]][1] = i;
        }
        stk.back() = i;
      } else {
        c[p[i] = stk.back()][1] = i;
        stk.push_back(i);
      }
    }
  }
}; // hash-cpp-all = e127c42059822fa5cfd16e249d98fad5
```

hld.hpp

"data-structure/flatten-vector.hpp" 182 lines

```cpp
struct HLD {
  int n;
  Vec<int> ord, iord, sz;
  Vec<int> depth;
  Vec<pair<int, int>> path;
  Vec<pair<int, int>> edges;
  HLD(int n_) : n(n_), iord(n), sz(n, 1), depth(n), path(n)
    ↪ {}

  void add_edge(int a, int b) {
    edges.emplace_back(a, b);
    edges.emplace_back(b, a);
  }

  void build(int r = 0) {
    auto tr = FlattenVector<int>(n, edges);

    auto par = Vec<int>(n, -1); // hash-cpp-1
    auto topo = Vec<int>{r};
    topo.reserve(n);
    for (int z = 0; z < n; z++) {
      int v = topo[z];
      for (int w : tr[v]) {
        if (w == par[v]) continue;
        par[w] = v;
        depth[w] = depth[v] + 1;
        topo.push_back(w);
      }
    } // hash-cpp-1 = 91fa439a841e76f46edeebc764b068d4

    auto max_ch = Vec<int>(n, -1); // hash-cpp-2
    for (int v : topo | std::views::drop(1) | std::views::
      ↪reverse) {
      int p = par[v];
      sz[p] += sz[v];
```

```
    if (max_ch[p] == -1 || sz[max_ch[p]] < sz[v]) {
      max_ch[p] = v;
    }
  } // hash-cpp-2 = 017206f47c6fb0fce7cecf88537ec4c1

  auto stk = Vec<pair<int, bool>>{{r, true}}; // hash-cpp
    ↪-3
  stk.reserve(n);
  while (!stk.empty()) {
    auto [v, ir] = stk.back();
    stk.pop_back();
    int i = int(size(ord));
    ord.push_back(v);
    iord[v] = i;
    if (ir) {
      path[i] = {par[v] == -1 ? -1 : iord[par[v]], 1};
    } else {
      path[i] = {path[i - 1].first, path[i - 1].second +
        ↪1};
    }
    if (max_ch[v] == -1) continue;
    for (int w : tr[v]) {
      if (w == par[v] || w == max_ch[v]) continue;
      stk.emplace_back(w, true);
    }
    stk.emplace_back(max_ch[v], false);
  } // hash-cpp-3 = 098e7b2bf4bc733606489799d82a6d09
}

bool in_subtree(int a, int v) const {
  return iord[a] <= iord[v] && iord[v] < iord[a] + sz[a];
}

Opt<int> get_ancestor(int a, int k) const { // hash-cpp-4
  assert(k >= 0);
  a = iord[a];
  while (a != -1 && k) {
    if (k >= path[a].second) {
      k -= path[a].second;
      a = path[a].first;
    } else {
      a -= k;
      k = 0;
    }
  }
  if (a != -1) {
    return ord[a];
  } else {
    return std::nullopt;
  }
} // hash-cpp-4 = e2a19fffa4a8f39d85ba61c16889a45c

int lca(int a, int b) const { // hash-cpp-5
  a = iord[a], b = iord[b];
  while (true) {
    if (a > b) swap(a, b);
    if (a > b - path[b].second) {
      return ord[a];
    }
    b = path[b].first;
  }
} // hash-cpp-5 = 55dbd4c94db1271544da38e0e05015c1

Opt<int> jump(int s, int t, int d) const { // hash-cpp-6
  int w = lca(s, t);
  if (d <= depth[s] - depth[w]) {
    return get_ancestor(s, d);
  } else {
    d = (depth[s] + depth[t] - 2 * depth[w]) - d;
    if (d >= 0) {
      return get_ancestor(t, d);
    } else {
      return std::nullopt;
    }
  }
} // hash-cpp-6 = 656007c3e4cc94b03fc9827135d52ee6

Vec<pair<int, int>> extract(int s, int t) { // hash-cpp-7
  static Vec<pair<int, int>> res;
  res.clear();
  s = iord[s], t = iord[t];
  while (true) {
    if (t > s - path[s].second) {
      res.emplace_back(s, t + 1);
      break;
    }
    res.emplace_back(s, s - path[s].second + 1);
    s = path[s].first;
  }
  return res;
} // hash-cpp-7 = cfe7a03a44193fc1c00e3f6a5f40b1c8

template <bool vertex = true, class F>
void apply(int s, int t, F f) { // hash-cpp-8
  int a = lca(s, t);
  for (auto&& [x, y] : extract(s, a)) {
    f(x + 1, y);
  }
  if constexpr (vertex) {
    f(iord[a], iord[a] + 1);
  }
  auto des = extract(t, a);
  for (auto&& [x, y] : des | std::views::reverse) {
    f(y, x + 1);
  }
} // hash-cpp-8 = 9f6536b32da8351e82174844f1be0f09

template <class F> int get_lowest(int a, F f) const { //
  ↪hash-cpp-9
  a = iord[a];
  while (a != -1) {
    int t = a - path[a].second + 1;
    if (!f(ord[t])) {
      a = path[a].first;
      continue;
    }
    int mi = t, ma = a + 1;
    while (ma - mi > 1) {
      int md = (mi + ma) / 2;
      if (f(ord[md]))
        mi = md;
      else
        ma = md;
    }
    return ord[mi];
  }
  return -1;
} // hash-cpp-9 = 0d2776498d957db35fa731b99c42c002

Vec<int> inds;
pair<Vec<int>, Vec<int>> compress(Vec<int> vs) { // hash-
  ↪cpp-10
  inds.resize(n, -1);
  auto cmp = [&](int a, int b) { return iord[a] < iord[b
    ↪]; };
  std::ranges::sort(vs, cmp);
  vs.erase(unique(begin(vs), end(vs)), end(vs));
  int num = int(size(vs));
  assert(num >= 1);
  for (int z = 1; z < num; z++) {
    vs.push_back(lca(vs[z - 1], vs[z]));
  }

  std::ranges::sort(vs, cmp);
  vs.erase(unique(begin(vs), end(vs)), end(vs));
  num = int(size(vs));
  for (int z = 0; z < num; z++) inds[vs[z]] = z;
  Vec<int> par(num, -1);
  for (int z = 1; z < num; z++) {
    par[z] = inds[lca(vs[z - 1], vs[z])];
  }
  return {vs, par};
} // hash-cpp-10 = 9529325c9d44175758bcc26856fb927d
};
```

tree-dp.hpp

Description: All-direction tree DP blackbox

75 lines

```
using std::views::reverse;

template <class S> struct TreeDP {
  template <class G, class RF, class CF> struct Inner {
    Vec<S> low, high;
    Vec<int> edges, par;
    const RF rake;
    const CF compress;
    Inner(const G& g, auto make, RF rake_, CF compress_)
      : rake(rake_), compress(compress_) {
      int n = int(g.size());
      auto single = Vec<S>(n);
      edges.resize(n - 1);
      for (int v = 0; v < n; v++) {
        single[v] = make(v);
        for (int e : g[v]) edges[e] ^= v;
      }

      auto bfs = Vec<int>{0};
      bfs.reserve(n);
      par.assign(n, -1);
      for (size_t z = 0; z < size(bfs); z++) {
        int v = bfs[z];
        for (int e : g[v]) {
          if (par[v] == e) continue;
          int w = v ^ edges[e];
          par[w] = e;
          bfs.push_back(w);
        }
      }

      low = single;
      auto up = Vec<S>(n);
      auto pref = Vec<S>(n);
      for (int v : bfs | reverse) {
        for (int e : g[v]) {
          if (par[v] == e) continue;
          int w = v ^ edges[e];
          pref[w] = low[v];
          up[w] = compress(low[w], e, v);
          low[v] = rake(low[v], up[w], v);
        }
      }

      high.resize(n);
```

```
    auto f = Opt<S>();
    for (int v : bfs) {
      if (v != 0) [[likely]] {
        f = compress(high[v], par[v], v);
      }
      for (int e : g[v] | reverse) {
        if (par[v] == e) continue;
        int w = v ^ edges[e];
        if (f.has_value()) [[likely]] {
          high[w] = rake(pref[w], *f, v);
          f = rake(up[w], *f, v);
        } else {
          high[w] = pref[w];
          f = up[w];
        }
      }
    }
  }

  S get_vertex(int v) const {
    if (v == 0) return low[v];
    return rake(low[v], compress(high[v], par[v], v), v);
  }
};

template <class G, class RF, class CF>
static auto solve(const G& g, auto make, RF rake, CF
  ↪compress) {
  return Inner(g, make, rake, compress);
}
}; // hash-cpp-all = 9a03ab6861288f0a6b879150bdeada6e
```

Graph (6)

eulerian-trail.hpp

Description: Eulerian undirected/directed trail algorithm. Returns a list of (vertex, edge)'s in the trail with `src` at the start, or `std::nullopt` if there is no trail. Note that choosing the starting vertex can be somewhat ad-hoc :)

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

87 lines

```
namespace eulerian_trail {

// (vertex, edge)
// For the returned list, edge is the preceding edge of
  ↪that vertex
using E = pair<int, int>;
template <bool cyc_only = false>
Opt<Vec<E>> go(int nv, const Vec<Vec<E>>& g, int ne, int
  ↪src = 0) {
  assert(nv == int(size(g)));
  assert(0 <= src && src < nv);

  Vec<Vec<E>::const_iterator> its(nv); // hash-cpp-1
  for (int i = 0; i < nv; i++) its[i] = begin(g[i]);
  Vec<int> state(nv);
  if constexpr (!cyc_only) state[src]++;
  Vec<bool> seen(ne);
  Vec<E> res, stk = {E(src, -1)}; // hash-cpp-1 = 2
    ↪ce68639be60f45b59e1e28cbea708c0

  while (!stk.empty()) { // hash-cpp-2
    auto [i, p] = stk.back();
    auto& it = its[i];
    if (it == end(g[i])) {
      res.emplace_back(i, p);
      stk.pop_back();
      continue;
    }
    auto [j, e] = *(it++);
    if (!seen[e]) {
      state[i]--, state[j]++;
      stk.emplace_back(j, e);
      seen[e] = true;
    }
  }
  if (int(size(res)) != ne + 1) {
    return {};
  }
  for (int s : state) {
    if (s < 0) return {};
  }
  return Vec<E>(rbegin(res), rend(res)); // hash-cpp-2 =
    ↪334b4d0a9ab464b25cfa1dd7a1b0714b
}

template <bool cyc_only = false>
Opt<Vec<E>> trail_undirected(int nv, const Vec<pair<int,
  ↪int>>& edges) {
  assert(nv > 0);

  Vec<Vec<E>> g(nv);
  int e = 0;
  for (auto [a, b] : edges) {
    g[a].emplace_back(b, e);
    g[b].emplace_back(a, e);
    e++;
  }

  int src = 0; // hash-cpp-3
  for (int i = 0; i < nv; i++) {
    if (!g[i].empty()) src = i;
  }
  for (int i = 0; i < nv; i++) {
    if (size(g[i]) % 2 == 1) src = i;
  } // hash-cpp-3 = 80724ceaee254adebb9b8f246229e6d6
  return go<cyc_only>(nv, g, int(size(edges)), src);
}

template <bool cyc_only = false>
Opt<Vec<E>> trail_directed(int nv, const Vec<pair<int, int
  ↪>>& edges) {
  assert(nv > 0);

  Vec<Vec<E>> g(nv);
  Vec<int> indeg(nv);
  int e = 0;
  for (auto [a, b] : edges) {
    g[a].emplace_back(b, e);
    indeg[b]++;
    e++;
  }

  int src = 0; // hash-cpp-4
  for (int i = 0; i < nv; i++) {
    if (!g[i].empty()) src = i;
  }
  for (int i = 0; i < nv; i++) {
    if (indeg[i] < int(size(g[i]))) src = i;
  } // hash-cpp-4 = a6820e1aab49fceff350c7c4747a3e7c
  return go<cyc_only>(nv, g, int(size(edges)), src);
}

}  // namespace eulerian_trail
```

bipartite.hpp

Description: Hopcroft–Karp algorithm that gives a maximum bipartite matching. `edges` should be a sequence of edges (a_i, b_i) such that $a_i \in [n_l]$ and $b_i \in [n_r]$

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

93 lines

```
struct Bipartite {
  int nl, nr;
  Vec<Vec<int>> g;
  Vec<int> mtl, mtr, lvl;
  Vec<bool> seen;
  Bipartite(int nl_, int nr_)
    : nl(nl_),
      nr(nr_),
      g(nl),
      mtl(nl, -1),
      mtr(nr, -1),
      lvl(nl),
      seen(nr) {}

  void add_edge(int a, int b) { g[a].push_back(b); }

  void run() {
    Vec<int> q;
    q.reserve(nl);
    while (true) {
      q.clear(); // hash-cpp-1
      for (int i = 0; i < nl; i++) {
        if (mtl[i] == -1) {
          lvl[i] = 0;
          q.push_back(i);
        } else {
          lvl[i] = -1;
        }
      }
      // If there is an alternating path that
      // leads to some unmatched left-side vertex
      bool f = false;
      for (int z = 0; z < int(q.size()); z++) {
        int i = q[z];
        for (int j : g[i]) {
          int o = mtr[j];
          if (o == -1) {
            f = true;
          } else if (lvl[o] == -1) {
            lvl[o] = lvl[i] + 1;
            q.push_back(o);
          }
        }
      }
      if (!f) {
        for (int i : q) {
          for (int j : g[i]) seen[j] = true;
        }
        break;
      } // hash-cpp-1 = 7810f15a14a1c2f2460b4a75dc158b26

      Vec<bool> done(nl); // hash-cpp-2
      for (int s = 0; s < nl; s++) {
        if (mtl[s] != -1) continue;

        auto dfs = [&](auto self, int i) -> bool {
          if (done[i]) return false;
          done[i] = true;
          for (int j : g[i]) {
            int o = mtr[j];
            if (o == -1 ||
```

```
            (lvl[i] + 1 == lvl[o] && self(self, o))) {
            mtl[i] = j, mtr[j] = i;
            return true;
          }
        }
        return false;
      };
      dfs(dfs, s);
    } // hash-cpp-2 = 815ba2fd9b6cbd3873d6b1685e348d6d
  }

  Vec<pair<int, int>> matching() { // hash-cpp-3
    Vec<pair<int, int>> res;
    for (int i = 0; i < nl; i++) {
      int j = mtl[i];
      if (j != -1) res.emplace_back(i, j);
    }
    return res;
  } // hash-cpp-3 = 99b9b84954bc198aa01b8e0472d9bc57

  pair<Vec<int>, Vec<int>> vertex_cover() { // hash-cpp-4
    Vec<int> lvs, rvs;
    for (int i = 0; i < nl; i++) {
      if (lvl[i] == -1) lvs.push_back(i);
    }
    for (int j = 0; j < nr; j++) {
      if (seen[j]) rvs.push_back(j);
    }
    return {lvs, rvs};
  } // hash-cpp-4 = eefb9beeb3ba02086a05cd06bd677af7
};
```

enumerate-triangles.hpp

Description: Enumerates all triangles (x, y, z) in an undirected graph
Time: TODO

"data-structure/flatten-vector.hpp" 27 lines

```
template <class F> void triangles(int n, Vec<pair<int, int
    ↪>> edges, F f) {
  auto deg = Vec<int>(n);
  for (auto [a, b] : edges) {
    deg[a]++;
    deg[b]++;
  }
  for (auto& [a, b] : edges) {
    if (tie(deg[a], a) > tie(deg[b], b)) {
      swap(a, b);
    }
  }
  auto adj = FlattenVector<int>(n, edges);

  Vec<int> ind(n);
  int i = 0;
  for (int x = 0; x < n; x++) {
    ++i;
    for (int y : adj[x]) ind[y] = i;
    for (int y : adj[x]) {
      for (int z : adj[y]) {
        if (ind[z] == i) {
          f(x, y, z);
        }
      }
    }
  }
} // hash-cpp-all = ea79f127a7db1c040e491740b59d70c6
```

dfs-tree.hpp

46 lines

```
struct DFSTree {
  struct Inner {
    Vec<int> ord;
    Vec<int> par;

    template <class G> Inner(const G& g) {
      int n = int(g.size());
      ord.reserve(n);
      par = Vec<int>(n, -2);
      auto its = Vec<decltype(g[0].begin())>(n);
      for (int v = 0; v < n; v++) {
        its[v] = g[v].begin();
      }
      for (int r = 0; r < n; r++) {
        if (par[r] != -2) continue;
        par[r] = -1;
        ord.push_back(r);
        int v = r;
        while (v >= 0) {
          auto& it = its[v];
          if (it == g[v].end()) {
            v = par[v];
            continue;
          }
          if (int w = *it; par[w] == -2) {
            par[w] = v;
            v = w;
            ord.push_back(v);
          }
          it = std::next(it);
        }
      }
    }

    Vec<int> get_iord() const {
      int n = int(par.size());
      auto iord = Vec<int>(n);
      for (int i = 0; i < n; i++) {
        iord[ord[i]] = i;
      }
      return iord;
    }
  };

  template <class G> static auto make(const G& g) { return
      ↪Inner(g); }
}; // hash-cpp-all = 7b238aa6c08eb0a736bb52143c2460da
```

block-cut.hpp

"graph/dfs-tree.hpp" 36 lines

```
template <class G, class F> int block_cut_tree(const G& g,
    ↪F f) {
  int n = int(g.size());
  auto dfs = DFSTree::make(g);
  auto iord = dfs.get_iord();

  auto low = iord;
  for (int v = 0; v < n; v++) {
    for (int w : g[v]) {
      low[v] = min(low[v], iord[w]);
    }
  }
  for (int v : dfs.ord | std::views::reverse) {
    if (int p = dfs.par[v]; p != -1) {
      low[p] = min(low[p], low[v]);
    }
  }

  int nb = 0;
  for (int v : dfs.ord) {
    if (int p = dfs.par[v]; p != -1) {
      if (low[v] < iord[p]) {
        low[v] = low[p];
        f(low[v], v);
      } else {
        low[v] = nb++;
        f(low[v], p);
        f(low[v], v);
      }
    }
  }
  for (int v = 0; v < n; v++) {
    if (g[v].empty()) f(nb++, v);
  }

  return nb;
} // hash-cpp-all = 8744004b9b0ea7ec498fab64d5372217
```

two-edge-cc.hpp

"graph/dfs-tree.hpp" 35 lines

```
template <class G, class F> int two_edge_cc(const G& g, F f
    ↪) {
  int n = int(g.size());
  auto dfs = DFSTree::make(g);
  auto iord = dfs.get_iord();

  auto low = iord;
  auto seen_par = Vec<u8>(n);
  for (int v = 0; v < n; v++) {
    for (int w : g[v]) {
      if (dfs.par[w] == v && !seen_par[w]) {
        seen_par[w] = 1;
      } else {
        low[w] = min(low[w], iord[v]);
      }
    }
  }
  for (int v : dfs.ord | std::views::reverse) {
    if (int p = dfs.par[v]; p != -1) {
      low[p] = min(low[p], low[v]);
    }
  }

  int n2 = 0;
  auto idx = Vec<int>(n);
  for (int v : dfs.ord) {
    if (low[v] == iord[v]) {
      idx[v] = n2++;
    } else {
      idx[v] = idx[dfs.par[v]];
    }
    f(idx[v], v);
  }

  return n2;
} // hash-cpp-all = 8327f365da334218553abed60432ad7d
```

Number Theory (7)

factor.hpp

Description: Returns prime factors in ascending order (e.g. 2299 -> {11, 11, 19})

Time: $\mathcal{O}\left(n^{1/4}\right)$

73 lines

```cpp
namespace factor {

template <class T> T pow_mod(T a, u64 b, T m) { // hash-cpp-1
  T r = 1;
  while (b) {
    if (b & 1) r = r * a % m;
    a = a * a % m;
    b >>= 1;
  }
  return r;
} // hash-cpp-1 = 8153dd104c95c28bf40b51ccfa359c28

template <class T> bool is_prime(T n) { // hash-cpp-2
  if (n <= 1 || n % 2 == 0) return (n == 2);
  int s = __builtin_ctzll(n - 1);
  T d = (n - 1) >> s;
  for (u128 a : {2, 325, 9375, 28178, 450775, 9780504,
      1795265022}) {
    a %= n;
    if (a == 0) continue;
    a = pow_mod<u128>(a, d, n);
    if (T(a) == 1 || T(a) == n - 1) continue;
    for (int i = 0; i < s - 1; i++) {
      a = a * a % n;
      if (T(a) == n - 1) break;
    }
    if (T(a) != n - 1) return false;
  }
  return true;
} // hash-cpp-2 = 91830792ecc62049005cfc63ebf602cb

template <class T> T pollard(T n) { // hash-cpp-3
  T x = 0, y = 0, t = 30, p = 2, it = 1;
  auto f = [&](T a) { return T(u128(a) * a % n) + it; };
  while (t++ % 40 || std::gcd(p, n) == 1) {
    if (x == y) {
      x = ++it, y = f(x);
    }
    T d = max(x, y) - min(x, y);
    if (T q = T(u128(p) * d % n); q) {
      p = q;
    }
    x = f(x), y = f(f(y));
  }
  return std::gcd(p, n);
} // hash-cpp-3 = 750e917ce8d6b979f4af39351f6fedc1

// Returns prime factors in ascending order
template <class T> Vec<T> factorize(T n) { // hash-cpp-4
  if (n == 1) return {};
  if (is_prime(n)) return {n};
  T f = pollard(n);
  auto a = factorize(f), b = factorize(n / f);
  Vec<T> c(a.size() + b.size());
  merge(begin(a), end(a), begin(b), end(b), begin(c));
  return c;
} // hash-cpp-4 = 33d26dfcca56fce967c8610a56b9f578

template <class T> T primitive_root(T p) {
  assert(is_prime(p));
  auto f = factorize(p - 1);
  T c;
  while (true) {
    c = rand_int<T>(1, p - 1);
    if (!std::ranges::any_of(f, [&](T d) {
        return pow_mod<u128>(c, (p - 1) / d, p) == 1;
      })) {
      break;
    }
  }
  return c;
}

}  // namespace factor
```

int-kth-root.hpp

Description: Computes $\lfloor a^{1/k} \rfloor$

18 lines

```cpp
template <class I = u64> I int_kth_root(I a, I k) {
  if (k == 1) return a;
  if (k >= 64) return (a == 0 ? 0 : 1);

  using T = __uint128_t;
  auto works = [&](T x) -> bool {
    T r = 1;
    for (int n = int(k); n; n >>= 1) {
      if (n & 1) r *= x;
      x *= x;
    }
    return r <= a;
  };
  if (a == I(-1)) a--;
  I rt = I(pow(double(a), nextafter(1 / double(k), 0)));
  while (works(rt + 1)) rt++;
  return rt;
} // hash-cpp-all = df0dda344149ce60f0cfff3a65363fcc
```

eratosthenes.hpp

Description: Prime sieve for generating all primes up to a certain limit n

18 lines

```cpp
inline Vec<int> prime_enumerate(int n) {
  auto sieve = Vec<bool>(n / 3 + 1, true); // hash-cpp-1
  int qe = int(size(sieve));
  int n2 = int(sqrt(n));
  for (int p = 5, d = 4, i = 1; p <= n2; p += d = 6 - d, i
      ++) {
    if (!sieve[i]) continue;
    for (int q = p * p / 3, r = d * p / 3 + (d * p % 3 ==
        2), s = 2 * p;
        q < qe; q += r = s - r) {
      sieve[q] = false;
    }
  } // hash-cpp-1 = dd325b3ad774bc7c50b9ae91467d6a84
  auto res = Vec<int>{2, 3}; // hash-cpp-2
  for (int p = 5, d = 4, i = 1; p <= n; p += d = 6 - d, i
      ++) {
    if (sieve[i]) res.push_back(p);
  }
  while (!res.empty() && res.back() > n) res.pop_back();
  return res; // hash-cpp-2 =
    c90bbe8732ffd47485a6c953502a419d
}
```

multiplicative-sum.hpp

Description: Blackbox sieve. Modify `f(v, p, c)` to fit your wish

191 lines

```cpp
namespace multiplicative_sum {

using std::sqrt, std::cbrt;

inline i64 isqrt(i64 n) { return i64(sqrt(n)); }
inline i64 icbrt(i64 n) { return i64(cbrt(n)); }
inline i64 sq(i64 a) { return a * a; }

inline i64 sump(int k, i64 n) {
  if (k == 0) {
    return n;
  }
  // Unreachable
  assert(false);
  return 0;
}

template <class T, int K> struct counting_primes {
  using A = array<T, K>; // hash-cpp-1
  void add(A& a, const A& b) {
    for (int k = 0; k < K; k++) a[k] += b[k];
  }
  void sub(A& a, const A& b) {
    for (int k = 0; k < K; k++) a[k] -= b[k];
  }
  const Vec<int>& ps;
  const i64 n;
  const int n2, n3, n6;
  int s;
  Vec<i64> vs;
  Vec<A> sum, fw;
  A pref; // hash-cpp-1 = 5ba8cd301505a2b3b37ae1caef795746

  A getpows(T p) { // hash-cpp-2
    A res;
    res[0] = 1;
    for (int k = 1; k < K; k++) {
      res[k] = res[k - 1] * p;
    }
    return res;
  } // hash-cpp-2 = 1282c5b86030aad73569a7ce5b1492b8

  void trans(int i, int p) { // hash-cpp-3
    A w = getpows(p);
    int j = get_idx(vs[i] / p);
    for (int k = 0; k < K; k++) {
      sum[i][k] -= (sum[j][k] - pref[k]) * w[k];
    }
  } // hash-cpp-3 = 1b75b45369ff775f2ea7ab66bf8c1726

  void trans2(int i, int p) { // hash-cpp-4
    A w = getpows(p);
    int j = get_idx(vs[i] / p);
    A z = sum[j];
    if (j >= n3) {
      j -= n3;
      for (; j < int(fw.size()); j += (j + 1) & (-j - 1)) {
        add(z, fw[j]);
      }
    }
    for (int k = 0; k < K; k++) {
      sum[i][k] -= (z[k] - pref[k]) * w[k];
    }
  } // hash-cpp-4 = 4f7b414359537414dc08ba63b39ad6ec
```

```
void upd(int i, i64 cur, bool f) { // hash-cpp-5
  if (!f) {
    A w = getpows(cur);
    for (int j = get_idx(cur) - n3; j >= 0; j -= (j + 1)
      ↪& (-j - 1)) {
      sub(fw[j], w);
    }
  }
  for (int j = i; cur * ps[j] <= vs[n3]; j++) {
    upd(j, cur * ps[j], false);
  }
} // hash-cpp-5 = 439188de4fe2b38413e6f3a29720a190

counting_primes(i64 n_, const Vec<int>& ps_)
  : ps(ps_),
    n(n_),
    n2(int(isqrt(n))),
    n3(int(icbrt(n))),
    n6(int(icbrt(n2))) { // hash-cpp-6
  {
    i64 v = n;
    while (v) {
      vs.push_back(v);
      v = n / (n / v + 1);
    }
  }
  s = int(vs.size());

  sum.resize(s);
  for (int i = 0; i < s; i++) {
    for (int k = 0; k < K; k++) {
      sum[i][k] = sump(k, vs[i]) - 1;
    }
  }

  int idx = 0;
  pref = {};
  {
    while (ps[idx] <= n6) {
      for (int i = 0; i < s; i++) {
        if (sq(ps[idx]) > vs[i]) break;
        trans(i, ps[idx]);
      }
      add(pref, getpows(ps[idx]));
      idx++;
    }
  }
  {
    fw.resize(s - n3);
    while (ps[idx] <= n3) {
      for (int i = 0; i < n3; i++) {
        if (sq(ps[idx]) > vs[i]) break;
        trans2(i, ps[idx]);
      }
      upd(idx, ps[idx], true);
      add(pref, getpows(ps[idx]));
      idx++;
    }
    for (int i = s - n3 - 1; i >= 0; i--) {
      int j = i + ((i + 1) & (-i - 1));
      if (j < int(fw.size())) {
        add(fw[i], fw[j]);
      }
    }
    for (int i = 0; i < s - n3; i++) {
      add(sum[i + n3], fw[i]);
    }
  }
  {
    while (ps[idx] <= n2) {
      for (int i = 0; i < s; i++) {
        if (sq(ps[idx]) > vs[i]) break;
        trans(i, ps[idx]);
      }
      add(pref, getpows(ps[idx]));
      idx++;
    }
  }
} // hash-cpp-6 = 3e7e01a3e2d245e5d87c8b8857b1a63a

int get_idx(i64 a) { // hash-cpp-7
  return int(a <= n2 ? s - a : n / a - 1);
} // hash-cpp-7 = 3b7faedbd45f949fd7fa348ec51114b4

// f(v)=f(p^c), where p is some prime
// totient function as an example:
T f(i64, int p, int c) {
  T res = p - 1;
  for (int z = 0; z < c - 1; z++) {
    res *= p;
  }
  return res;
}

Vec<T> buf;
T multiplicative_sum() { // hash-cpp-8
  // sum of [p is prime] f(p)
  buf.resize(s);
  for (int i = 0; i < s; i++) {
    buf[i] = sum[i][1] - sum[i][0];
  }

  T ans = 1 + buf[0];
  auto dfs =
    yc([&](auto self, int i, int c, i64 v, i64 lim, T cur
      ↪) -> void {
      ans += cur * f(v * ps[i], ps[i], c + 1);
      if (lim >= sq(ps[i])) {
        self(i, c + 1, v * ps[i], lim / ps[i], cur);
      }
      cur *= f(v, ps[i], c);
      ans += cur * (buf[get_idx(lim)] - buf[get_idx(ps[i
        ↪])]);
      for (int j = i + 1; sq(ps[j]) <= lim; j++) {
        self(j, 1, ps[j], lim / ps[j], cur);
      }
    });
  for (int i = 0; true; i++) {
    if (sq(ps[i]) <= n) {
      dfs(i, 1, ps[i], n / ps[i], 1);
    } else {
      break;
    }
  }
  return ans;
} // hash-cpp-8 = 4f3d37cb3d7f7ca7c9d6e8ac6ea65fec
};

}  // namespace multiplicative_sum
```

String (8)

z-algorithm.hpp

Description: Returns $r_0, \ldots, r_N$ such that $s[0..r_i) = s[i..i+r_i)$. In particular, $r_0 = N$ and $r_N = 0$
Time: $\mathcal{O}(N)$

16 lines

```
template <class S> Vec<int> z_algo(const S& s) {
  int n = int(size(s));
  auto res = Vec<int>(n + 1);
  for (int i = 1, j = 0; i <= n; i++) {
    int& k = res[i];
    if (j + res[j] <= i) {
      k = 0;
    } else {
      k = min(res[j] + j - i, res[i - j]);
    }
    while (i + k < n && s[k] == s[i + k]) k++;
    if (j + res[j] < i + res[i]) j = i;
  }
  res[0] = n;
  return res;
} // hash-cpp-all = 4cea91273404f4082bf8a501cb55b583
```

manacher.hpp

Description: Returns maximum lengths of "palindromic" (whatever that means) substring of S centered at each point
Time: $\mathcal{O}(N)$

29 lines

```
/*
 * eq(i, j): whether [i, j] (inclusive) is palindromic,
 * given that [i+1, j-1] is palindromic.
 * Properties:
 *   * res[i] == i (mod 2)
 *   * k + res[i-k] < res[i] => res[i+k] = res[i-k]
 *   * k + res[i-k] >= res[i] => res[i-k] >= res[i] - k
 * [i, j) being palindromic <=> j-i <= res[i+j]
 * In particular, res[2*i+1] = -1 states that [i, i] is not
   ↪ palindromic.
 */
template <class Eq> Vec<int> manacher(int n, Eq eq) {
  auto res = Vec<int>(2 * n + 1);
  int i = 0, a = 0, b = 0;
  while (i <= 2 * n) {
    while (0 < a && b < n) {
      if (i - 2 * a >= -1 && !eq(a - 1, b)) break;
      a--, b++;
    }
    int j = b - a;
    res[i] = j;
    int k = 1;
    while (k < j && k + res[i - k] < j) {
      res[i + k] = res[i - k];
      k++;
    }
    i += k, a += k;
  }
  return res;
} // hash-cpp-all = 74419e7ff8d0f727783ee0493139f472
```

hashint.hpp

Description: Self-explanatory string hashing structure

41 lines

```
struct HashInt {
  using H = HashInt; // hash-cpp-1
  using T = unsigned long long;
  using L = u128;
  static constexpr T m = (T(1) << 61) - 1;
```

```
  static constexpr T m8 = m * 8;
  T v;
  HashInt() : v(0) {}
  HashInt(T a) : v(a % m * 8) {}
  T get() const { return v == m8 ? 0 : v; } // hash-cpp-1 =
    ↪ e05add30af8e33cb5045c566d05e5b48

  H& operator+=(const H& o) { // hash-cpp-2
    if (__builtin_uaddll_overflow(v, o.v, &v)) v -= m8;
    return *this;
  }
  H& operator-=(const H& o) {
    if (__builtin_usubll_overflow(v, o.v, &v)) v += m8;
    return *this;
  } // hash-cpp-2 = 03a79be35c3f8731c3c4e64a1799cc94

  H& operator*=(const H& o) { // hash-cpp-3
    L t = L(v) * o.v;
    T x = T(t >> 67 << 3);
    T y = T(t << 61 >> 64);
    if (__builtin_uaddll_overflow(x, y, &v)) v -= m8;
    return *this;
  } // hash-cpp-3 = c535ff913f601dd75b6c039556dda31a

  friend H operator+(const H& a, const H& b) {
    return H(a) += b;
  } // hash-cpp-4
  friend H operator-(const H& a, const H& b) { return H(a)
    ↪-= b; }
  friend H operator*(const H& a, const H& b) { return H(a)
    ↪*= b; }
  friend bool operator==(const H& a, const H& b) {
    return a.get() == b.get();
  } // hash-cpp-4 = a0691df50f2aaecfb8e6c06073f30887
};

inline HashInt rand_base() {
  return 2 * std::uniform_int_distribution<u64>(4e10, 5e10)
    ↪(mt) + 1;
}
```

hashed-string.hpp

27 lines

```
template <class H> struct HashedManager {
  const H base;
  Vec<H> pows;
  HashedManager(const H& base_) : base(base_), pows({H(1)})
    ↪ {}

  using Hashed = Vec<H>;
  template <class S> Hashed make(const S& s) {
    size_t n = size(s);
    while (size(pows) <= n) {
      pows.push_back(pows.back() * base);
    }
    auto hs = Hashed(n + 1);
    hs[0] = H(0);
    for (size_t i = 0; i < n; i++) {
      hs[i + 1] = hs[i] * base + H(s[i]);
    }
    return hs;
  }

  using Str = pair<H, int>;
  Str get(const Hashed& hs, int l, int r) const {
    return Str(hs[r] - hs[l] * pows[r - l], r - l);
  }
  Str concat(const Str& a, const Str& b) const {
    return Str(a.first * pows[b.second] + b.first, a.second
      ↪ + b.second);
  }
}; // hash-cpp-all = 6895cd30b67fbf765e699ab55530b10c
```

suffix-array.hpp

Description: Builds the suffix array given a string
Time: $\mathcal{O}(N)$ building

115 lines

```
// Work in progress

struct SuffixArray {
  int n;
  Vec<int> sa;
  Vec<int> isa;
  Vec<int> lcp;
  SuffixArray(int n_) : n(n_) {}

  template <class S> static SuffixArray construct(const S&
    ↪s) {
    int n = int(size(s));
    auto sa = SuffixArray(n);

    sa.build_sa_fast(s);

    sa.build_isa();
    sa.build_lcp(s);

    return sa;
  }

  template <class S> void build_sa_fast(const S& s) {
    auto a = Vec<int>(n);
    int s_min = int(*std::ranges::min_element(s));
    for (int i = 0; i < n; i++) {
      a[i] = int(s[i]) - s_min;
    }
    sa = sais(a);
  }

  static Vec<int> sais(Vec<int> a) {
    int n = int(size(a));
    int m = *std::ranges::max_element(a) + 1;
    auto pos = Vec<int>(m + 1);
    for (auto c : a) pos[c + 1]++;
    std::partial_sum(begin(pos), end(pos), begin(pos));
    auto s = Vec<i8>(n);
    for (int i = n - 2; i >= 0; i--) {
      s[i] = (a[i] != a[i + 1] ? a[i] < a[i + 1] : s[i +
        ↪1]);
    }

    auto x = Vec<int>(m);
    auto sa = Vec<int>(n);
    auto induce = [&](const Vec<int>& lms) {
      std::fill(begin(sa), end(sa), -1);
      auto push_L = [&](int i) {
        if (i >= 0 && !s[i]) sa[x[a[i]]++] = i;
      };
      auto push_S = [&](int i) {
        if (i >= 0 && s[i]) sa[--x[a[i]]] = i;
      };
      std::copy(begin(pos) + 1, end(pos), begin(x));
      for (int i = int(size(lms)) - 1; i >= 0; i--) {
        push_S(lms[i]);
      }
      std::copy(begin(pos), end(pos) - 1, begin(x));
      push_L(n - 1);
      for (int i = 0; i < n; i++) {
        push_L(sa[i] - 1);
      }
      std::copy(begin(pos) + 1, end(pos), begin(x));
      for (int i = n - 1; i >= 0; i--) {
        push_S(sa[i] - 1);
      }
    };

    auto ok = [&](int i) { return i == n || (!s[i - 1] && s
      ↪[i]); };
    auto eq = [&](int i, int j) {
      do {
        if (a[i++] != a[j++]) return false;
      } while (!ok(i) && !ok(j));
      return ok(i) && ok(j);
    };
    auto lms = Vec<int>();
    for (int i = 1; i < n; i++) {
      if (ok(i)) lms.push_back(i);
    }
    induce(lms);

    if (!lms.empty()) {
      int p = -1, w = 0;
      auto mp = Vec<int>(n);
      for (auto v : sa)
        if (v && ok(v)) {
          if (p != -1 && eq(p, v)) w--;
          mp[p = v] = w++;
        }
      auto b = lms;
      for (auto& v : b) v = mp[v];
      b = sais(b);
      for (auto& v : b) v = lms[v];
      induce(b);
    }

    return sa;
  }

  void build_isa() { // hash-cpp-1
    isa.resize(n);
    for (int i = 0; i < n; i++) isa[sa[i]] = i;
  } // hash-cpp-1 = 591197b8d061d2bdddf1cd2eaf45d3fa

  template <class S> void build_lcp(const S& s) {
    lcp.resize(n - 1);
    int k = 0;
    for (int i : isa) {
      if (k) k--;
      if (i + 1 < n) {
        int l = sa[i], r = sa[i + 1];
        while (l + k < n && r + k < n && s[l + k] == s[r +
          ↪k]) k++;
        lcp[i] = k;
      }
    }
  }
};
```

eertree.hpp

Description: Call `reset()` to move back to the root

64 lines

```
// 0, ..., K-1
```

```cpp
struct Eertree {
  struct Node { // hash-cpp-1
    std::forward_list<u32> ch;
    int par, fail;
    int l, r;  // location of the first ocurrence
    int len() const { return r - l; }
    Node(int par_, int fail_, int l_, int r_)
      : par(par_), fail(fail_), l(l_), r(r_) {}
  };
  Vec<Node> nodes;
  Vec<u8> buf;
  int cur; // hash-cpp-1 = 806ae082e53393439b423ea9dd73e4b0
  Eertree(int alloc = 0) {
    if (alloc) {
      nodes.reserve(alloc + 2);
      buf.reserve(alloc);
    }
    // 0: EVEN; 1: ODD
    nodes.emplace_back(0, 1, 0, 0);
    nodes.emplace_back(0, 0, 0, -1);
    reset();
  }

  void reset() {
    cur = 1;
    buf.clear();
    // sentinel character; this implies that you should not
       ↪ use u8(-1) in your input
    buf.push_back(u8(-1));
  }

  int get_ch(int v, u8 x) const {
    for (u32 cw : nodes[v].ch) {
      u8 c = u8(cw);
      if (c == x) return int(cw >> 8);
    }
    return 0;
  }

  int get_fail(int v) const {
    while (buf.back() != buf.end()[-(nodes[v].len() + 2)])
       ↪{
      v = nodes[v].fail;
    }
    return v;
  }

  int append(u8 a) { // hash-cpp-2
    buf.push_back(a);
    cur = get_fail(cur);
    int nxt = get_ch(cur, a);
    if (!nxt) {
      int nf = get_ch(get_fail(nodes[cur].fail), a);
      nxt = int(nodes.size());
      nodes[cur].ch.push_front(a | (nxt << 8));
      int i = int(buf.size()) - 2;
      nodes.emplace_back(cur, nf, i - nodes[cur].len() - 1,
         ↪ i + 1);
    }
    cur = nxt;
    return cur;
  } // hash-cpp-2 = 249faca4ac6dc4fc28ba7e0b82a52af7

  int size() const { return int(nodes.size()); }
  const Node& operator[](int i) const { return nodes[i]; }
};
```

Geometry (9)

base.hpp

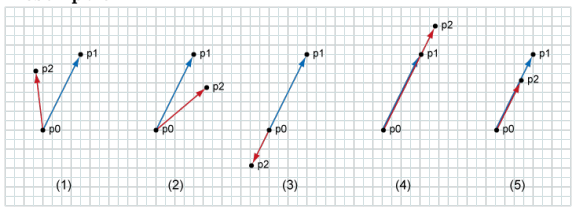
Description: Primitive operations

109 lines

```cpp
namespace geometry {

using std::fmod;

const double EPS = 1e-9;
template <class T> int sgn(T a) { return (a > EPS) - (a < -
   ↪EPS); }
template <class T> int sgn(T a, T b) { return sgn(a - b); }

const double PI = acos(-1.);

template <class T> struct Point {
  using P = Point;
  T x, y;
  Point(T x_ = T(), T y_ = T()) : x(x_), y(y_) {}

  P& operator+=(const P& p) { // hash-cpp-1
    x += p.x, y += p.y;
    return *this;
  }
  P& operator-=(const P& p) {
    x -= p.x, y -= p.y;
    return *this;
  }
  friend P operator+(const P& a, const P& b) { return P(a)
     ↪+= b; }
  friend P operator-(const P& a, const P& b) {
    return P(a) -= b;
  } // hash-cpp-1 = 32704ee5f47251cb7a5a8bcddb7996e3

  P& operator*=(const T& t) { // hash-cpp-2
    x *= t, y *= t;
    return *this;
  }
  P& operator/=(const T& t) {
    x /= t, y /= t;
    return *this;
  }
  friend P operator*(const P& a, const T& t) { return P(a)
     ↪*= t; }
  friend P operator/(const P& a, const T& t) {
    return P(a) /= t;
  } // hash-cpp-2 = 56a8dfabc9e0968b82d5006dda2d4d7e

  friend T dot(const P& a, const P& b) { return a.x * b.x +
     ↪ a.y * b.y; }
  friend T crs(const P& a, const P& b) { return a.x * b.y -
     ↪ a.y * b.x; }

  P operator-() const { return P(-x, -y); }

  friend int cmp(const P& a, const P& b) { // hash-cpp-3
    int z = sgn(a.x, b.x);
    return z ? z : sgn(a.y, b.y);
  } // hash-cpp-3 = 1553bdfc52835908d4fc0bd0a91b7134

  friend bool operator<(const P& a, const P& b) { return
     ↪cmp(a, b) < 0; }
  friend bool operator<=(const P& a, const P& b) { return
     ↪cmp(a, b) <= 0; }
  friend T dist2(const P& p) { return p.x * p.x + p.y * p.y
     ↪; }
  friend auto dist(const P& p) { return sqrt(D(dist2(p)));
     ↪}

  friend P unit(const P& p) { return p / p.dist(); }

  friend double arg(const P& p) { return atan2(p.y, p.x); }

  friend T rabs(const P& p) { return max(abs(p.x), abs(p.y)
     ↪); }

  friend bool operator==(const P& a, const P& b) {
    return sgn(rabs(a - b)) == 0;
  }
  friend bool operator!=(const P& a, const P& b) { return
     ↪!(a == b); }

  explicit operator pair<T, T>() const { return pair<T, T>(
     ↪x, y); }

  static P polar(double m, double a) { return P(m * cos(a),
     ↪ m * sin(a)); }
};

template <class T> std::istream& operator>>(std::istream&
   ↪is, Point<T>& p) {
  return is >> p.x >> p.y;
}

template <class T>
int sgncrs(const Point<T>& a, const Point<T>& b) { // hash-
   ↪cpp-4
  T cr = crs(a, b);
  if (abs(cr) <= (rabs(a) + rabs(b)) * EPS) return 0;
  return (cr < 0 ? -1 : 1);
} // hash-cpp-4 = 16e5d8b9630b699a9c02dd9f87a381ac

// not tested
template <class D> D norm_angle(D a) { // hash-cpp-5
  D res = fmod(a + PI, 2 * PI);
  if (res < 0) {
    res += PI;
  } else {
    res -= PI;
  }
  return res;
} // hash-cpp-5 = 8d996afb8002237f3ae57e1308edf700

// not tested
template <class D> D norm_nonnegative(D a) { // hash-cpp-6
  D res = fmod(a, 2 * PI);
  if (res < 0) res += 2 * PI;
  return res;
} // hash-cpp-6 = 9b568a78d4e45eabe33de16e27a603e2

// arg given lengths a, b, c,
// assumming a, b, c are valid
template <class D> D arg(D a, D b, D c) { // hash-cpp-7
  return acos(std::clamp<D>((a * a + b * b - c * c) / (2 *
     ↪a * b), -1, 1));
} // hash-cpp-7 = 2a5ba3e05833252f908cf246319cb8a5

}  // namespace geometry
```

ccw.hpp

Description:



"geometry/base.hpp" 23 lines

```cpp
namespace geometry {

// CGL_1_C
// 1: COUNTER_CLOCKWISE (1)
// -1: CLOCKWISE (2)
// 2: ONLINE_BACK (3)
// -2: ONLINE_FRONT (4)
// 0: ON_SEGMENT (5)
template <class T> int ccw(const Point<T>& a, const Point<T>& b) {
  int s = sgncrs(a, b); // hash-cpp-1
  if (s) return s;
  if (!sgn(rabs(b)) || !sgn(rabs(b - a))) return 0;
  if (dot(a, b) < 0) return 2;
  if (dot(-a, b - a) < 0) return -2;
  return 0; // hash-cpp-1 = 69
    ↪b3ea9eb828036b9188f3ad430e43c6
}

template <class T>
int ccw(const Point<T>& a, const Point<T>& b, const Point<T>& c) {
  return ccw(b - a, c - a);
}

}  // namespace geometry
```

linear.hpp

Description: Line/segment operations

"geometry/ccw.hpp" 78 lines

```cpp
namespace geometry {

// Work in progress
template <class T> struct L {
  using P = Point<T>; // hash-cpp-1
  P s, t;
  L(P s_ = P(), P t_ = P()) : s(s_), t(t_) {}
  friend P vec(const L& l) { return l.t - l.s; }
  friend auto dist(const L& l) { return dist(vec(l)); }
  friend double arg(const L& l) { return arg(vec(l)); } //
    ↪hash-cpp-1 = 3b47cb7801ce03c60d9e99647d747e3e
};

template <class T> Point<T> project(const L<T>& l, const Point<T>& p) {
  Point<T> v = vec(l); // hash-cpp-2
  return l.s + v * dot(v, p - l.s) / dist2(v); // hash-cpp
    ↪-2 = 1648b2909b8a019a73d6dca8c2221821
}

template <class T> int ccw(const L<T>& l, const Point<T>& p) {
  return ccw(l.s, l.t, p);
}

template <class T> bool insSL(const L<T>& s, const L<T>& l) {
  int a = ccw(l, s.s), b = ccw(l, s.t); // hash-cpp-3
  return (a % 2 == 0 || b % 2 == 0 || a != b); // hash-cpp
    ↪-3 = b9a91465128f28ef356c93ebaf83fe9a
}

template <class T> bool insSS(const L<T>& s, const L<T>& t) {
  int a = ccw(s, t.s), b = ccw(s, t.t), c = ccw(t, s.s),
    d = ccw(t, s.t); // hash-cpp-4
  return (a * b <= 0 && c * d <= 0); // hash-cpp-4 = 9
    ↪b7a89c2e911aa573091b5a1faa61c73
}

template <class T> double distLP(const L<T>& l, const Point<T>& p) {
  return abs(crs(vec(l), p - l.s)) / dist(l);
} // hash-cpp-4 = d41d8cd98f00b204e9800998ecf8427e

template <class T> double distSP(const L<T>& s, const Point<T>& p) {
  Point<T> q = project(s, p); // hash-cpp-5
  if (ccw(s, q) == 0) {
    return dist(p - q);
  } else {
    return min(dist(s.s - p), dist(s.t - p));
  } // hash-cpp-5 = 945f48b295abe750e175655b55622d68
}

template <class T> double distSS(const L<T>& s, const L<T>& t) {
  if (insSS(s, t)) return 0; // hash-cpp-6
  return min({distSP(s, t.s), distSP(s, t.t), distSP(t, s.s),
      distSP(t, s.t)}); // hash-cpp-6 = 137892
        ↪a1f3cc7267c5428d5afcd2f912
}

// TODO: usage
template <class T> int crossLL(const L<T>& l, const L<T>& m, Point<T>& r) {
  Point<T> vl = vec(l), vm = vec(m); // hash-cpp-7
  T cr1 = crs(vl, vm), cr2 = crs(vl, l.t - m.s);
  if (sgncrs(vl, vm) == 0) {
    r = l.s;
    if (sgncrs(vec(l), l.t - m.s)) return 0;
    return -1;
  }
  r = m.s + vm * cr2 / cr1;
  return 1; // hash-cpp-7 = 4
    ↪a241749cafeaf60a788de611ef3bfc7
}

// TODO usage
template <class T> int crossSS(const L<T>& l, const L<T>& m, Point<T>& r) {
  int u = crossLL(l, m, r); // hash-cpp-8
  if (u == 0) return 0;
  if (u == -1) {
    r = max(min(l.s, l.t), min(m.s, m.t));
    Point<T> q = min(max(l.s, l.t), max(m.s, m.t));
    return (q < r) ? 0 : (q == r ? 1 : -1);
  }
  if (ccw(l, r) == 0 && ccw(m, r) == 0) return 1;
  return 0; // hash-cpp-8 =
    ↪fd35bfd104a3ff8b53a0830d8c5fb4de
}

}  // namespace geometry
```

polygonal.hpp

Description: Polygon operations

"geometry/ccw.hpp", "geometry/linear.hpp" 127 lines

```cpp
namespace geometry {

template <class T> T area2(const Vec<Point<T>>& pol) {
  if (pol.empty()) return 0; // hash-cpp-1
  T res = 0;
  auto a = pol.back();
  for (auto b : pol) {
    res += crs(a, b);
    a = b;
  }
  return res; // hash-cpp-1 = 775
    ↪ae1ac4c8001aeb02f544d07a49976
}

// (1:left) | (2: right) is inside between v[i] -- v[i + 1]
template <class T>
Vec<pair<Point<T>, int>> insPolL(const Vec<Point<T>>& pol, const L<T>& l) {
  using Pi = pair<Point<T>, int>; // hash-cpp-2
  Vec<Pi> v;
  Point<T> a, b = pol.back();
  for (auto c : pol) {
    a = b;
    b = c;
    Point<T> p;
    if (crossLL({a, b}, l, p) != 1) continue;
    int sa = ccw(l, a) % 2, sb = ccw(l, b) % 2;
    if (sa > sb) swap(sa, sb);
    if (sa != 1 && sb == 1) v.push_back({p, 1});
    if (sa == -1 && sb != -1) v.push_back({p, 2});
  }
  sort(begin(v), end(v), [&](Pi x, Pi y) {
    auto vl = vec(l);
    return dot(vl, x.first - l.s) < dot(vl, y.first - l.s);
  });
  int m = int(size(v));
  Vec<Pi> res;
  for (int i = 0; i < m; i++) {
    if (i) v[i].second ^= v[i - 1].second;
    if (!res.empty() && res.back().first == v[i].first) res
      ↪.pop_back();
    res.push_back(v[i]);
  }
  return res; // hash-cpp-2 =
    ↪fa0aa36808c1117f5e0c435f1e650188
}

// 0: outside, 1: on line, 2: inside
template <class T> int contains(const Vec<Point<T>>& pol, const Point<T>& p) {
  if (pol.empty()) return 0; // hash-cpp-3
  int in = -1;
  Point<T> a_, b_ = pol.back();
  for (auto c : pol) {
    a_ = b_, b_ = c;
    Point<T> a = a_, b = b_;
    if (ccw(a, b, p) == 0) return 1;
    if (a.y > b.y) swap(a, b);
    if (!(a.y <= p.y && p.y < b.y)) continue;
```

```
    if (sgn(a.y, p.y) ? (crs(a - p, b - p) > 0) : (a.x > p.x)) in *= -1;
  }
  return in + 1; // hash-cpp-3 = 9ba68a043a41b17dc2cfad19ed936b10
}

// pol: sorted and distinct
template <class T> Vec<Point<T>> convex_lower(const Vec<Point<T>>& pts) {
  assert(size(pts) >= 2); // hash-cpp-4
  Vec<Point<T>> res;
  for (auto d : pts) {
    while (size(res) > 1) {
      //if (ccw(res.end()[-2], res.end()[-1], d) != -1) break;
      if (ccw(res.end()[-2], res.end()[-1], d) == 1) break;
      res.pop_back();
    }
    res.push_back(d);
  }
  return res; // hash-cpp-4 = 62c051fd3c3066045c90f92f8c68e03f
}

template <class T> Vec<Point<T>> convex(Vec<Point<T>> pts) {
  sort(begin(pts), end(pts)); // hash-cpp-5
  pts.erase(unique(begin(pts), end(pts)), end(pts));
  if (size(pts) <= 1) return pts;
  Vec<Point<T>> lo = convex_lower(pts);
  reverse(begin(pts), end(pts));
  Vec<Point<T>> up = convex_lower(pts);
  lo.insert(begin(lo), begin(up) + 1, end(up) - 1);
  return lo; // hash-cpp-5 = af18b531b56e6e036e34231d4e170357
}

template <class T>
Vec<Point<T>> convex_cut(const Vec<Point<T>>& pol, const L<T>& l) {
  if (pol.empty()) return {}; // hash-cpp-6
  Vec<Point<T>> q;
  Point<T> a, b = pol.back();
  for (auto c : pol) {
    a = b, b = c;
    if ((ccw(l, a) % 2) * (ccw(l, b) % 2) < 0) {
      Point<T> buf;
      crossLL(l, L(a, b), buf);
      q.push_back(buf);
    }
    if (ccw(l, b) != -1) q.push_back(b);
  }
  return q; // hash-cpp-6 = b9b1502c04e92d079177d5fe2332a098
}

// pol: convex; this calls f(a, b) for each candidate (a, b)
template <class T, class F> void diameter(const Vec<Point<T>> pol, F f) {
  int n = int(size(pol)); // hash-cpp-7
  if (n == 2) {
    f(pol[0], pol[1]);
    return;
  }
  int x = 0, y = 0;
  for (int i = 1; i < n; i++) {
    if (pol[i] < pol[x]) x = i;
    if (pol[y] < pol[i]) y = i;
  }
  int sx = x, sy = y;
  while (sx != y || sy != x) {
    f(pol[x], pol[y]);
    int nx = (x + 1 < n) ? x + 1 : 0, ny = (y + 1 < n) ? y + 1 : 0;
    if (crs(pol[nx] - pol[x], pol[ny] - pol[y]) < 0) {
      x = nx;
    } else {
      y = ny;
    }
  } // hash-cpp-7 = af059874cebb4defb8479540d6cc0a64
}

}  // namespace geometry
```

circular.hpp

Description: Circle operations

"geometry/base.hpp", "geometry/linear.hpp" 100 lines

```
namespace geometry {

template <class T = double> struct C {
  using P = Point<T>;
  P c;
  T r;
  C(P c_ = P(), T r_ = T()) : c(c_), r(r_) {}

  friend P eval(const C& a, const double& angle) {
    return a.c + P::polar(a.r, angle);
  }
};

// NOT TESTED
// 0: outside; 1: on; 2: inside
template <class T> int contains(const C<T>& c, const Point<T>& p) {
  return sgn(c.r - dist(p - c.c)) + 1;
}

// 0-apart; 1-coincide;
// 2-a<b; 3-a<=b;
// 4-a>b; 5-a>=b;
// 6-a touches b; 7-a cross b
template <class T> int insCC(const C<T>& a, const C<T>& b) {
  T c = dist(a.c - b.c); // hash-cpp-1
  if (sgn(c) == 0 && sgn(a.r, b.r) == 0) return 1;
  int d = sgn(c + a.r - b.r);
  if (d <= 0) return d + 3;
  int e = sgn(c + b.r - a.r);
  if (e <= 0) return e + 5;
  int f = sgn(c - a.r - b.r);
  if (f <= 0) return -f + 6;
  return 0; // hash-cpp-1 = 61a31bab15e0280eeef65e91f68fbb64
}

template <class T>
C<T> incircle(const Point<T>& a, const Point<T>& b, const Point<T>& c) {
  T da = dist(b - c); // hash-cpp-2
  T db = dist(a - c);
  T dc = dist(a - b);
  T s = da + db + dc;
  return C((a * da + b * db + c * dc) / s,
           abs(crs(b - a, c - a)) / s); // hash-cpp-2 = d09688b6ea5a6265adc9f01e2e1add42
}

template <class T> C<T> outcircle(const Point<T>& a, Point<T> b, Point<T> c) {
  b -= a, c -= a; // hash-cpp-3
  T bb = dist2(b) / 2;
  T cc = dist2(c) / 2;
  T g = crs(b, c);
  T x = (bb * c.y - b.y * cc) / g;
  T y = (b.x * cc - bb * c.x) / g;
  T r = sqrt(x * x + y * y);
  x += a.x, y += a.y;
  return C(P(x, y), r); // hash-cpp-3 = 2c91ea98a4cda854f4fa8655033c30f9
}

template <class T>
int crossCL(const C<T>& c, const L<T>& l, array<Point<T>, 2>& res) {
  T u = distLP(l, c.c); // hash-cpp-4
  int t = sgn(u, c.r);
  if (t == 1) return 0;
  Point<T> v = project(l, c.c);
  Point<T> d = (t == 0 ? Point<T>(0, 0)
                : vec(l) * (sqrt(c.r * c.r - u * u) / dist(l)));
  res = {v - d, v + d};
  return 1 - t; // hash-cpp-4 = 9845747b9f30e2ef9396ccc9a677a456
}

// args of two intersections r, l seen by a.c,
// assuming two circles cross
template <class T> pair<T, T> crossCC_args(const C<T>& a, const C<T>& b) {
  Point<T> diff = b.c - a.c; // hash-cpp-5
  T c = arg(diff);
  T d = arg(a.r, dist(diff), b.r);
  return {c - d, c + d}; // hash-cpp-5 = e8f0f4a9396b9a5ae56850fd644fa152
}

template <class T>
int crossCC(const C<T>& a, const C<T>& b, array<Point<T>, 2>& res) {
  int t = insCC(a, b); // hash-cpp-6
  if (t == 0 || t == 1 || t == 2 || t == 4) return 0;
  auto [l, r] = crossCC_args(a, b);
  res = {eval(a, l), eval(a, r)};
  return 2 - (t == 3 || t == 5 || t == 6); // hash-cpp-6 = 56e3f5fa57011d34e17135616a072b98
}

template <class T>
int tangent(const C<T>& c, const Point<T>& p, array<Point<T>, 2>& res) {
  Point<T> diff = p - c.c; // hash-cpp-7
  T dd = dist(diff);
  int t = sgn(c.r, dd);
  if (t == 1) return 0;
  T d = acos(min<T>(c.r / dd, 1));
  T a = arg(diff);
  res = {eval(c, a - d), eval(c, a + d)};
  return 1 - t; // hash-cpp-7 = 4220898d66b628e02feef4e341179834
}
```

```
}  // namespace geometry
```

closest-pair.hpp

Description: Given a set of points, returns an arbitrary closest pair of points.

"geometry/base.hpp" 54 lines

```
namespace geometry {

template <class T> using P = Point<T>;

// PRECONDITION: There are at least 2 points
template <class T, class F> void closest_pair(Vec<P<T>> pts
    ↪, F f) {
  int n = int(size(pts));
  using PT = P<T>;
  std::ranges::sort(pts, [](PT a, PT b) -> bool { return a.
    ↪x < b.x; });
  T d = std::numeric_limits<T>::max();

  auto st = multiset<PT, decltype([](PT a, PT b) { return a
    ↪.y < b.y; })>();
  auto its = Vec<typename decltype(st)::const_iterator>(
    ↪size(pts));

  auto update = [&](PT a, PT b) {
    T d2 = dist2(a - b);
    if (d2 < d) {
      d = d2;
      f(a, b);
    }
  };

  for (int i = 0, j = 0; i < n; i++) {
    PT p = pts[i];

    auto sq = [](T x) { return x * x; };
    while (j < i && sq(p.x - pts[j].x) >= d) {
      st.erase(its[j++]);
    }
    auto u = st.upper_bound(p);
    {
      auto t = u;
      while (true) {
        if (t == begin(st)) break;
        t = prev(t);
        update(*t, p);
        if (sq(p.y - t->y) >= d) break;
      }
    }
    {
      auto t = u;
      while (true) {
        if (t == end(st)) break;
        if (sq(p.y - t->y) >= d) break;
        update(*t, p);
        t = next(t);
      }
    }
    its[i] = st.emplace_hint(u, p);
  }
}

}  // namespace geometry
// hash-cpp-all = c4f8905539549bc6bcef2fb1ffa9c08d
```

Other (10)

two-sat.hpp

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||!b)\&\&(!a||c)\&\&(d||!b)\&\&\ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit inversions (~x)

Usage: `TwoSat ts(number of boolean variables);`
`ts.either(0, ~3); // Var 0 is true or var 3 is false`
`ts.set_value(2); // Var 2 is true`
`ts.at_most_one({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true`
`ts.solve(); // Returns true iff it is solvable`
`ts.values[0..N-1] holds the assigned values to the vars`

Time: $\mathcal{O}(N+E)$

"data-structure/flatten-vector.hpp" 62 lines

```
struct TwoSat {
  int n;
  Vec<pair<int, int>> edges;
  TwoSat(int n_ = 0) : n(n_) {}

  int add_var() { return n++; }
  void either(int a, int b) {
    a = max(2 * a, -1 - 2 * a);
    b = max(2 * b, -1 - 2 * b);
    edges.emplace_back(a ^ 1, b);
    edges.emplace_back(b ^ 1, a);
  }
  void set_value(int x) { either(x, x); }
  // NOT VERIFIED
  void at_most_one(const Vec<int>& vs) {
    if (size(vs) <= 1) return;
    int cur = ~vs[0];
    for (int v : vs | std::views::drop(2)) {
      int nxt = add_var();
      either(cur, ~v);
      either(cur, nxt);
      either(~v, nxt);
      cur = ~nxt;
    }
    either(cur, ~vs[1]);
  }

  Opt<Vec<i8>> solve() {
    auto r = Vec<i8>(n, -1);
    auto g = FlattenVector<int>(2 * n, std::move(edges));
    auto q = Vec<int>();
    auto bfs = [&](int s) -> bool {
      q.clear();
      q.push_back(s);
      r[s / 2] = !(s % 2);
      for (size_t z = 0; z < size(q); z++) {
        int v = q[z];
        for (int w : g[v]) {
          if (r[w / 2] == -1) {
            r[w / 2] = !(w % 2);
            q.push_back(w);
          } else if (r[w / 2] == w % 2) {
            return false;
          }
        }
      }
      return true;
    };
    for (int i = 0; i < n; i++) {
      if (r[i] != -1 || bfs(2 * i + 1)) {
        continue;
      }
      for (int v : q) {
        r[v / 2] = -1;
      }
      if (!bfs(2 * i)) {
        return std::nullopt;
      }
    }
    return r;
  }
}; // hash-cpp-all = 54b5da48c538588d96f402e4743ac38b
```