Chinese University of Hong Kong, Shenzhen

???

adapted from KTH ACM Contest Template Library

Contest (1)

base.hpp

<bits/stdc++.h> 33 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 i32 = int32_t;
using i64 = int64_t;
using u32 = uint32_t;
using u64 = uint64_t;
using u128 = __uint128_t;

template <class F> struct yc_result { // hash-cpp-1
  F f;
  template <class T> explicit yc_result(T&& f_) : f(std::forward<T>(f_)) {}
  template <class... A> decltype(auto) operator()(A&&... as) {
    return f(std::ref(*this), std::forward<A>(as)...);
  }
};
template <class F> decltype(auto) yc(F&& f) {
  return yc_result<std::decay_t<F>>(std::forward<F>(f));
} // hash-cpp-1 = 29f56ac8972876d9a2d0563f83238f42

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);
}
```

bashrc

5 lines

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

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
```

vimrc

9 lines

```
set nocp ai bs=2 hls ic is lbr ls=2 mouse=a nu ru sc scs smd so=3 sw=4 ts=4
filetype plugin indent on
syn on
map gA m'ggVG"+y''

set cindent cino=j1,(0,ws,Ws

com -range=% -nargs=1 P exe "<line1>,<line2>!".<q-args> |y|sil u|echom @"
au FileType cpp com! -buffer -range=% Hash <line1>,<line2>P cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum
```

hash-cpp.sh

1 lines

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

fast-input.hpp

Description: Fast scanner implementation based on `fread`

57 lines

```
namespace fast_input {

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

  void read() {} // hash-cpp-1
  template <class H, class... T> void read(H& h, T&... t) {
    read_single(h);
    read(t...);
  } // hash-cpp-1 = 1be9d87558b4e70f056af5e4bc8df866

  char buf[1 << 16]; // hash-cpp-2
  size_t s = 0, e = 0;
  char get() {
    if (s >= e) {
      buf[0] = 0;
      s = 0;
      e = fread(buf, 1, sizeof(buf), f);
    }
    return buf[s++];
  } // hash-cpp-2 = 836ba78888edb5fec27c4231ad0b7d2a

  template <class T> void read_single(T& r) { // hash-cpp-3
    char c;
    while ((c = get()) <= ' ') {
    }
    bool neg = false;
    if (c == '-') {
      neg = true;
      c = get();
    }
    r = 0;
    do {
      r = 10 * r + (c & 15);
    } while ((c = get()) >= '0');
    if (neg) r = -r;
  } // hash-cpp-3 = cebd1d5b920bf264178f488b1d2f482c

  void read_single(string& r) { // hash-cpp-4
    char c;
    while ((c = get()) <= ' ') {
    }
    r = {};
    do {
      r += c;
    } while ((c = get()) > ' ');
  } // hash-cpp-4 = f68e2d2b2c7e50653f7a8b6bc046ff29

  void read_single(double& r) { // hash-cpp-5
    string z;
    read_single(z);
    r = stod(z);
  } // hash-cpp-5 = 32d080eb6e36c0c1cede9030bbb31fa1
};

}  // namespace fast_input
```

Data Structure (2)

hash-map.hpp

Description: Faster and safer hash map.

<ext/pb_ds/assoc_container.hpp> 13 lines

```
struct CustomHash { // hash-cpp-1
  size_t operator()(uint64_t x) const {
    static const uint64_t z = std::chrono::steady_clock::now().time_since_epoch().count(),
                  c = uint64_t(4e18*acos(0))+71;
    return __builtin_bswap64((x^z)*c);
  }
}; // hash-cpp-1 = 3757d267e7e0ab0ba63368f565e830cd

template <class K, class V, class Hash = CustomHash> // hash-cpp-2
using HashMap = __gnu_pbds::gp_hash_table<K, V, Hash>; // hash-cpp-2 = 4fe2baba5ae354ac6dd53d37ad221011

template <class K, class Hash = CustomHash> // hash-cpp-3
using HashSet = HashMap<K, __gnu_pbds::null_type, Hash>; // hash-cpp-3 = 1d899df3bf29329f777189feb8d1944c
```

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
```

```cpp
  // 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

```cpp
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

```cpp
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 operator()(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 = 53644a8954cd96497e83e9d294062852
};
```

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

```cpp
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);
```

```cpp
      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

45 lines

```cpp
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 = 0b46cd5fba53f4c166094224da58ee1c
};
```

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

39 lines

```cpp
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 = 2b98c40c29f240ca9a861a8267ad00e5
  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

```cpp
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

83 lines

```
struct FastSet {
  using U = uint64_t; // hash-cpp-1
  int n, h;
  Vec<Vec<U>> x;
  FastSet(int n_ = 0) : n(n_) {
    int m = (n ? n : 1);
    do {
      x.push_back(Vec<U>((m + 63) >> 6));
      m = (m + 63) >> 6;
    } while (m > 1);
    h = int(x.size());
  } // hash-cpp-1 = 2da8265def2fc367747a41880f3db1dd

  bool empty() const { // hash-cpp-2
    return !x[h - 1][0];
  }
  bool operator[](int i) const {
    return (x[0][i >> 6] >> (i & 63)) & 1;
  } // hash-cpp-2 = e7139f9a9d939bcdaea656a0e3dcb204

  void set(int i) { // hash-cpp-3
    for (int d = 0; d < h; d++) {
      int q = i >> 6, r = i & 63;
      x[d][q] |= U(1) << r;
      i = q;
    }
  } // hash-cpp-3 = 3319dfe2dcef21686104393ed36b8705

  void reset(int i) { // hash-cpp-4
    for (int d = 0; d < h; d++) {
      int q = i >> 6, r = i & 63;
      if ((x[d][q] &= ~(U(1) << r))) break;
      i = q;
    }
  } // hash-cpp-4 = 1f4723e2daf4308e36bca9899dfea88c

  // min active j s.t. j >= i
  int next(int i) const { // hash-cpp-5
    if (i >= n) return n;
    i = max(i, 0);
    for (int d = 0; d < h; d++) {
      int q = i >> 6, r = i & 63;
      if (q >= int(x[d].size())) break;
      U up = x[d][q] >> r;
      if (up) {
        i += __builtin_ctzll(up);
        for (int e = d - 1; e >= 0; e--) {
          i = i << 6 | __builtin_ctzll(x[e][i]);
        }
        return i;
      }
      i = q + 1;
    }
    return n;
  } // hash-cpp-5 = 2a01cef716336e62e563b5d73eaaaf40

  // max active j s.t. j <= i
  int prev(int i) const { // hash-cpp-6
    if (i < 0) return -1;
    i = min(i, n - 1);
    for (int d = 0; d < h; d++) {
      if (i < 0) break;
      int q = i >> 6, r = i & 63;
      U lo = x[d][q] << (63 - r);
      if (lo) {
        i -= __builtin_clzll(lo);
        for (int e = d - 1; e >= 0; e--) {
          i = i << 6 | (63 - __builtin_clzll(x[e][i]));
        }
        return i;
      }
      i = q - 1;
    }
    return -1;
  } // hash-cpp-6 = f8f01973030c47d09562f7ad1e93b4cc

  // 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)

tree-dp.hpp

Description: All-direction tree DP blackbox

75 lines

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

template <class S> struct TreeDP {
  template <class RF, class CF> struct Inner {
    Vec<S> low, high;
    Vec<int> edges, par;
    const RF rake;
    const CF compress;
    Inner(const Vec<Vec<int>>& g, auto make, RF rake_, CF
      ↪compress_)
      : rake(rake_), compress(compress_) {
      int n = int(size(g));
      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 RF, class CF>
  static auto solve(const Vec<Vec<int>>& g, auto make, RF
    ↪rake, CF compress) {
    return Inner(g, make, rake, compress);
  }
}; // hash-cpp-all = e6eb5c7b0a756da0ff868f7529ab74f2
```

monotone-minima.hpp

Description: Given an $N \times M$ matrix A, returns $m_i = \text{argmin}_j A_{i,j}$ 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);
```

```cpp
  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 $a_{i+1} - a_i \leq a_{i+2} - a_{i+1}$, 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

```cpp
// 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

```cpp
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" 56 lines

```cpp
// 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

47 lines

```cpp
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 = 747b64cd3779b0e594a5a9027b3c39d1

  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 = 0d3449609c465ca434b9110ef55a1bbb

  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

Description: NTT; mostly the same with fft.hpp?

58 lines

```cpp
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 (int(rt.size()) < 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(rt.size()); 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 = 655f600a9a77631192fd75eafc85ca68
  int s = std::countr_zero(u32(rev.size()) / 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 = a.begin() + 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 = f23c43b126df8456580998bdd258e943
}

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(a.begin()+1, a.begin()+n);
  nft(a, n);
} // hash-cpp-3 = f8de950ed7edf5706c926011f03b2ec9

template <class T> Vec<T> multiply(Vec<T> a, Vec<T> b) { // hash-cpp-4
  int n = int(a.size()), m = int(b.size());
  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(a.begin() + 1, a.end());
  nft(a, s);
  a.resize(n + m - 1);
  return a;
} // hash-cpp-4 = 0498b2b8050168fa6522bca8beee2432
```

matrix.hpp

Description: Gaussian elimination and stuff. `solve_lineareq` returns the pair (some particular solution, a basis of the null space).

"algebra/modint.hpp" 116 lines

```cpp
namespace matrix {

template <class T>
using F_better = std::function<bool(T, T)>;

template <class T>
using F_zero = std::function<bool(T)>;

template <bool rref = false, class T>
pair<int, T> sweep(Vec<Vec<T>>& a,
  F_better<T> fb, F_zero<T> fz,
  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 (p == -1 || fb(a[i][j], a[p][j])) p = i;
    }
    if (p == -1 || fz(a[p][j])) {
      det = 0;
      continue;
    }
    if (r != p) {
      det = -det;
      swap(a[r], a[p]);
    }
    auto& ar = a[r];
    det *= ar[j]; // hash-cpp-1 = 68409b9e970dd293b0fbdda0e682d0c9

    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 (!fz(ai[j])) {
        T e = ai[j] * d;
        for (int k = j; k < w; k++) {
          ai[k] -= ar[k] * e;
        }
      }
    }
    r++;
  } // hash-cpp-3 = bf314b34183f0c8f2f977a8def861fab
  return {r, det};
}

template <class T>
pair<Vec<T>, Vec<Vec<T>>> solve_lineareq(Vec<Vec<T>> a, Vec<T> b,
  F_better<T> fb, F_zero<T> fz) {
  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, fb, fz, w).first;
  for (int i = r; i < h; i++) {
    if (!fz(a[i][w])) return {};
  }
  Vec<T> x(w);
  Vec<int> pivot(w, -1);
  int z = 0;
  for (int i = 0; i < r; i++) {
    while (fz(a[i][z])) z++;
    x[z] = a[i][w], pivot[z] = i;
  } // hash-cpp-4 = fb8df177c0b27e778f81b878f3f7ccbd
  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 {x, ker};
}

template <class T> Vec<Vec<T>> mat_inv(Vec<Vec<T>> a,
  F_better<T> fb, F_zero<T> fz) { // hash-cpp-6
  int n = int(a.size());
  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, fb, fz, 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 = 70c19d713df63577111f258894902980

template <class T> T mat_det(Vec<Vec<T>> a,
  F_better<T> fb, F_zero<T> fz) { // hash-cpp-7
  return sweep<false>(a, fb, fz).second;
} // hash-cpp-7 = fa5f2046ee1be299cee6c7f1f558ba9f

} // namespace matrix
```

Graph (5)

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

84 lines

```cpp
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(g.size()));
  assert(0 <= src && src < nv);

  Vec<Vec<E>::const_iterator> its(nv); // hash-cpp-1
  for (int i = 0; i < nv; i++) its[i] = g[i].begin();
  Vec<int> state(nv);
  if constexpr (!cyc_only) state[src]++;
  Vec<bool> seen(ne);
  Vec<E> res, stk = {E(src, -1)}; // hash-cpp-1 = 44
    ↪ffcd139b863856a1c1085a72f75835

  while (!stk.empty()) { // hash-cpp-2
    auto [i, p] = stk.back();
    auto& it = its[i];
    if (it == g[i].end()) {
      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(res.size()) != ne+1) return {};
  for (int s : state) if (s < 0) return {};
  return Vec<E>{res.rbegin(), res.rend()}; // hash-cpp-2 =
    ↪9d317858daa1c66b04cdbd533e28f25f
}

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 (g[i].size() % 2 == 1) src = i;
  } // hash-cpp-3 = 8f0c0499edac02a8775d04f38f6b519e
  return go<cyc_only>(nv, g, int(edges.size()), 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(g[i].size())) src = i;
  } // hash-cpp-4 = 78a6497411685fe139d007ac0cce4a8b
  return go<cyc_only>(nv, g, int(edges.size()), 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: $\mathcal{O}\left(E\sqrt{V}\right)$

83 lines

```cpp
struct Bipartite {
  int nl, nr;
  Vec<Vec<int>> g;
  Vec<int> mtl, mtr, lvl;
  Vec<bool> seen;
  Bipartite(int nl_, int nr_, const Vec<pair<int, int>>&
    ↪edges)
    : nl(nl_), nr(nr_),
    g(nl), mtl(nl, -1), mtr(nr, -1), lvl(nl), seen(nr) {
    for (auto [i, j] : edges) {
      g[i].push_back(j);
    }
    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 = 3c672de70b8adeba7d37b4685bbebca6

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

        yc([&](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(o)))
              ↪ {
              mtl[i] = j, mtr[j] = i;
              return true;
            }
          }
          return false;
        })(s);
      } // hash-cpp-2 = e7c3ecfd04d424a1bbdc5d50b4b78917
    }
  }

  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
};
```

hld.hpp

183 lines

```cpp
struct HLD {
  int n;
  Vec<int> ord, st, en, depth;
  Vec<pair<int, int>> heavy;
  HLD() {}
  HLD(const Vec<int>& par, int rt = -1) { build(par, rt); }

  void build(const Vec<int>& par, int rt = -1) {
    n = int(par.size()); // hash-cpp-1
    ord.resize(n);
    st.resize(n);
    en.resize(n);
    depth.resize(n);
    heavy.resize(n);
    Vec<Vec<int>> ch(n);
    for (int i = 0; i < n; i++) {
      if (par[i] != -1) ch[par[i]].push_back(i);
```

```
  } // hash-cpp-1 = b7299720443fc0c8737217c89884612c

  int i = 0;
  Vec<int> sub(n);
  auto go = [&](int g) -> void {
    yc([&](auto self, int v, int d = 0) -> void { // hash-cpp-2
      sub[v] = 1;
      depth[v] = d;
      for (int& w : ch[v]) {
        self(w, d+1);
        sub[v] += sub[w];
        if (sub[ch[v][0]] < sub[w]) swap(ch[v][0], w);
      }
    })(g); // hash-cpp-2 = f85fc1f8fd7047a905e24720a59d1d8b

    yc([&](auto self, int v, bool r = true) -> void { // hash-cpp-3
      ord[st[v] = i++] = v;
      if (r) {
        heavy[st[v]] = {par[v] == -1 ? -1 : st[par[v]], 1};
      } else {
        heavy[st[v]] = heavy[st[v]-1];
        heavy[st[v]].second++;
      }
      bool cr = false;
      for (int w : ch[v]) {
        self(w, cr);
        cr = true;
      }
      en[v] = i;
    })(g); // hash-cpp-3 = f1d1da4b4153cfda08dc6ef0502deaf4
  };

  if (rt == -1) {
    // rooted forest
    for (int v = 0; v < n; v++) {
      if (par[v] == -1) go(v);
    }
  } else {
    // rooted at rt
    assert(0 <= rt && rt < n);
    go(rt);
  }

  assert(i == n);
}

bool in_subtree(int a, int v) const {
  return st[a] <= st[v] && st[v] < en[a];
}

int get_ancestor(int a, int k) const { // hash-cpp-4
  assert(k >= 0);
  a = st[a];
  while (a != -1 && k) {
    if (k >= heavy[a].second) {
      k -= heavy[a].second;
      a = heavy[a].first;
    } else {
      a -= k;
      k = 0;
    }
  }
  if (a == -1) return -1;
  else return ord[a];
} // hash-cpp-4 = 38c66b004fd349c93647d7943f36251f

int lca(int a, int b) const { // hash-cpp-5
  a = st[a], b = st[b];
  while (true) {
    if (a > b) swap(a, b);
    if (a > b - heavy[b].second) {
      return ord[a];
    }
    b = heavy[b].first;
    if (b == -1) return -1;
  }
} // hash-cpp-5 = 9ee75bf6da246fa444c875c297d5c9a7

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<array<int, 2>> extract(int s, int t) { // hash-cpp-7
  static Vec<array<int, 2>> res;
  res.clear();
  s = st[s], t = st[t];
  while (true) {
    if (t > s - heavy[s].second) {
      res.push_back({s, t+1});
      break;
    }
    res.push_back({s, s - heavy[s].second + 1});
    s = heavy[s].first;
  }
  return res;
} // hash-cpp-7 = 28994ca6a34396e400fc93750c410e61

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(st[a], st[a]+1);
  }
  auto des = extract(t, a);
  reverse(des.begin(), des.end());
  for (auto&& [x, y] : des) {
    f(y, x+1);
  }
} // hash-cpp-8 = c27280d29b591909807fe89e7034137c

// NOT TESTED
template <class F> int get_lowest(int a, F f) const { // hash-cpp-9
  a = st[a];
  while (a != -1) {
    int t = a - heavy[a].second + 1;
    if (!f(ord[t])) {
      a = heavy[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 = b254f08e7b14254b490de927443e62ac

Vec<int> inds; // hash-cpp-10
  pair<Vec<int>, Vec<int>> compress(Vec<int> vs) {
  inds.resize(n, -1);
  auto cmp = [&](int a, int b) -> bool {
    return st[a] < st[b];
  };
  sort(vs.begin(), vs.end(), cmp);
  vs.erase(unique(vs.begin(), vs.end()), vs.end());
  int num = int(vs.size());
  assert(num >= 1);
  for (int z = 1; z < num; z++) {
    vs.push_back(lca(vs[z-1], vs[z]));
  }
  sort(vs.begin(), vs.end(), cmp);
  vs.erase(unique(vs.begin(), vs.end()), vs.end());
  num = int(vs.size());
  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 = ed607afe156c6e79bcbacffbf9298d5e
};
```

enumerate-triangles.hpp

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

25 lines

```
template <class F> void triangles(int n, const Vec<pair<int, int>>& edges, F f) {
  Vec<int> deg(n); // hash-cpp-1
  for (auto [a, b] : edges) {
    deg[a]++, deg[b]++;
  }
  Vec<Vec<int>> adj(n);
  for (auto [a, b] : edges) {
    if (tie(deg[a], a) > tie(deg[b], b)) swap(a, b);
    adj[a].push_back(b);
  } // hash-cpp-1 = 9b0f43606d4027f45b8f261dc68a595e

  Vec<int> ind(n); // hash-cpp-2
  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-2 = 9c84ca8b936d3ec0bc78ceeab0f70576
```

```
}
```

block-cut.hpp

47 lines

```
template <class E> Vec<Vec<int>> block_cut_tree(int n, const Vec<Vec<E>>& g) {
  Vec<Vec<int>> tr(n); // hash-cpp-1
  auto add = [&](int b, int v) -> void {
    tr[b].push_back(v);
    tr[v].push_back(b);
  }; // hash-cpp-1 = 71a481760eb715a52de9c708094631dc

  Vec<int> stk; stk.reserve(n);
  Vec<int> idx(n, -1);
  int t = 0;
  for (int s = 0; s < n; s++) {
    if (idx[s] != -1) continue;
    yc([&](auto self, int v, int p) -> int {
      stk.push_back(v); // hash-cpp-2
      idx[v] = t++;
      int low = idx[v] = t++;
      int c = 0;
      for (int w : g[v]) {
        if (w == p) continue;
        if (idx[w] == -1) {
          c++;
          auto z = stk.size();
          int nlow = self(w, v);
          low = min(low, nlow);
          if ((p == -1 && c > 1) || (p != -1 && idx[v] <= nlow)) {
            int b = int(tr.size());
            tr.resize(b+1);
            add(b, v);
            while (z < stk.size()) {
              add(b, stk.back());
              stk.pop_back();
            }
          }
        } else {
          low = min(low, idx[w]);
        }
      }
      return low; // hash-cpp-2 = 7cc064051424c44ab789d52113b58040
    })(s, -1);
    int b = int(tr.size()); // hash-cpp-3
    tr.resize(b+1);
    for (int v : stk) add(b, v);
    stk.clear(); // hash-cpp-3 = 98651a8db6af759650d4c4be638030dd
  }

  return tr;
}
```

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

72 lines

```
struct TwoSat {
  int n;
  Vec<Vec<int>> g;
  TwoSat(int n_ = 0) : n(n_), g(2*n) {}

  int add_var() {
    g.emplace_back(), g.emplace_back();
    return n++;
  }

  void either(int a, int b) { // hash-cpp-1
    a = max(2*a, -1-2*a);
    b = max(2*b, -1-2*b);
    g[a^1].push_back(b);
    g[b^1].push_back(a);
  } // hash-cpp-1 = 16e68cfa6a6fc20d6b21bb1a940571f2

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

  void at_most_one(const Vec<int>& vs) { // hash-cpp-2
    int m = int(vs.size());
    if (m <= 1) return;
    int cur = ~vs[0];
    for (int i = 2; i < m; i++) {
      int nxt = add_var();
      either(cur, ~vs[i]);
      either(cur, nxt);
      either(~vs[i], nxt);
      cur = ~nxt;
    }
    either(cur, ~vs[1]);
  } // hash-cpp-2 = db26391604156859ea3a1454f5e6542f

  Opt<Vec<bool>> solve() { // hash-cpp-3
    Vec<int> idx(2*n, -1), comp(2*n, -1), stk;
    int tm = 0;
    Vec<int> r(n, -1); // hash-cpp-3 = c3afc71507c12b6f401fe2a89834a739

    for (int s = 0; s < 2*n; s++) { // hash-cpp-4
      if (comp[s] != -1) continue;

      yc([&](auto self, int i) -> int {
        int low = idx[i] = tm++;
        stk.push_back(i);
        for (auto& j : g[i]) {
          if (comp[j] != -1) continue;
          low = min(low, idx[j] == -1 ? self(j) : idx[j]);
        }
        tm++;
        if (low == idx[i]) {
          while (true) {
            int z = stk.back();
            stk.pop_back();
            comp[z] = tm;
            if (r[z>>1] == -1) r[z>>1] = !(z&1);
            if (i == z) break;
          }
        }
        return idx[i] = low;
      })(s);
    } // hash-cpp-4 = 2a4eba424a97524adeb7fdf28a595304

    for (int i = 0; i < n; i++) { // hash-cpp-5
      if (comp[2*i] == comp[2*i+1]) return {};
    }
    Vec<bool> res(n);
    for (int i = 0; i < n; i++) res[i] = bool(r[i]);
    return res; // hash-cpp-5 = 4761de68c96f017f59d40fb2ebeccf75
  }
};
```

Number Theory (6)

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

```
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

```
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

```
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

```
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 (7)

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 E> Vec<int> manacher(int n, E e) {
  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 && !e(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 = 000c505275977a4289a6af3e23a739d2
```

hashint.hpp

Description: Self-explanatory string hashing structure

38 lines

```
struct HashInt {
  using H = HashInt; // hash-cpp-1
  using T = unsigned long long;
  using L = __uint128_t;
  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 =
    ↪ 441ee64fd18fdc7b1df56890de357f06

  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 =
    ↪b15740d449ec094c54eaf820a3f31571
};

inline HashInt rand_base() {
  return 2 * std::uniform_int_distribution<uint64_t>(4e10,
    ↪5e10)(mt) + 1;
}
```

suffix-array.hpp

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

117 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(s.size());
    SuffixArray sa(n);

    sa.build_sa_fast(s);

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

    return sa;
  }

  template <class S> void build_sa_fast(S s) {
    sa.resize(n+1);
    // kinda weird
    int sigma = 0;
    for (auto v : s) {
      sigma = max(sigma, int(v));
      assert(int(v) > 0);
    }
    ++sigma;
    s.push_back(0);
    // what exactly should be these sizes?
    Vec<int> freq(2 * max(n+1, sigma)), lms(2 * (n+1));
    Vec<char> type(2 * (n+1));
    sais(n, s.data(), sa.data(), sigma, freq.data(), lms.
      ↪data(), type.data());
  }

  template <class S> static void sais(int n, S* s, int* sa,
    ↪ int sigma,
                int* freq, int* lms, char* which) {
    int n2 = -1; // hash-cpp-1
    which[n] = 1;
    for (int i = n-1; i >= 0; i--) {
      which[i] = (s[i] == s[i+1] ? which[i+1] : s[i] < s[i
        ↪+1]);
      if (which[i] == 0 && which[i+1] == 1) {
        which[i+1] = 2;
        lms[++n2] = i+1;
      }
    }
    std::reverse(lms, lms + (n2+1));
    std::fill(freq, freq + sigma, 0);
    for (int i = 0; i <= n; i++) ++freq[int(s[i])];
    std::partial_sum(freq, freq + sigma, freq); // hash-cpp
      ↪-1 = d2b13dfc346726fd0e87317ea25a4085

    auto induce = [&](int* v) { // hash-cpp-2
      std::fill(sa, sa + n+1, 0);
      int* cur = freq + sigma;
      auto pushS = [&](int i) { sa[--cur[int(s[i])]] = i;
        ↪};
      auto pushL = [&](int i) { sa[cur[int(s[i])]++] = i;
        ↪};
      std::copy(freq, freq + sigma, cur);
      for (int i = n2; i >= 0; i--) pushS(v[i]);
      std::copy(freq, freq + sigma-1, cur + 1);
      for (int i = 0; i <= n; i++) {
        int j = sa[i]-1;
        if (j >= 0 && which[j] == 0) pushL(j);
      }
      std::copy(freq, freq + sigma, cur);
      for (int i = n; i >= 0; i--) {
        int j = sa[i]-1;
        if (j >= 0 && which[j]) pushS(j);
      }
    }; // hash-cpp-2 = e81905738085f0bfa7c9e4e52b1ac974

    auto eq = [&](int i, int j) { // hash-cpp-3
      if (s[i] == s[j]) {
        while (s[++i] == s[++j]) {
          if (which[i] == 2) return true;
        }
      }
      return false;
    }; // hash-cpp-3 = 208a2ae3b819fd126c46e4e3a88d30c0

    induce(lms); // hash-cpp-4
    int sigma2 = -1;
    int* s2 = std::remove_if(sa, sa + n, [&](int i) {
      ↪return which[i] != 2; });
    for (int i = 0; i <= n2; i++) {
      if (sigma2 <= 0 || !eq(sa[i], sa[i-1])) sigma2++;
      s2[sa[i]>>1] = sigma2;
    }
    for (int i = 0; i <= n2; i++) s2[i] = s2[lms[i]>>1];
    ++sigma2;
    if (sigma2 <= n2) {
      sais(n2, s2, sa, sigma2,
          freq + sigma, lms + (n2+1), which + (n+1));
    } else {
      for (int i = 0; i <= n2; i++) sa[s2[i]] = i;
    }
    auto buf = lms + (n2+1);
    for (int i = 0; i <= n2; i++) buf[i] = lms[sa[i]];
    induce(buf); // hash-cpp-4 = 2343
      ↪d6679d81286e35f22880aacf2343
  }

  void build_isa() { // hash-cpp-5
    isa.resize(n+1);
    for (int i = 0; i <= n; i++) isa[sa[i]] = i;
  } // hash-cpp-5 = bcb546b2fc94176fc80672b20a808f7f

  template <class S> void build_lcp(const S& s) {
    assert(n == int(s.size()));
    lcp.resize(n+1); // hash-cpp-6
    for (int i = 0, k = 0; i < n-1; i++) {
      int r = isa[i]-1, j = sa[r];
      while (k < n - max(i, j) && s[i+k] == s[j+k]) k++;
      lcp[r] = k;
      if (k) k--;
    } // hash-cpp-6 = 85193c3617ced5f805117ffdf20255aa
  }
};
```

eertree.hpp

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

55 lines

```
// 0, ..., K-1
template <int sigma> struct Eertree {
  struct Node { // hash-cpp-1
    array<int, sigma> ch;
    int par, fail;
    int l, r; // location of the first ocurrence
    Node(int p_, int f_, int l_, int r_) : ch{}, par(p_),
      ↪fail(f_), l(l_), r(r_) {}
    int len() const { return r-l; }
  };
  Vec<Node> x;
  Vec<int> buf;
  int cur; // hash-cpp-1 = d7cdae0bbec5fe81ccfad852c935e23c
  Eertree(int alloc = 0) {
    if (alloc) {
      x.reserve(alloc+2);
      buf.reserve(alloc);
    }
    x.emplace_back(-1, -1, 1, 0);
    x.emplace_back(0, 0, 0, 0);
    reset();
  }

  void reset() {
    cur = 1;
    buf.clear();
  }

  int append(int a) { // hash-cpp-2
    int i = int(buf.size());
    buf.push_back(a);
    auto works = [&](int v) -> bool {
      int l = i - x[v].len();
      return l > 0 && buf[l-1] == a;
    };
    for (; !works(cur); cur = x[cur].fail) {}
    if (!x[cur].ch[a]) {
      int f = x[cur].fail;
      if (f != -1) {
        for (; !works(f); f = x[f].fail) {}
      }
      int nf = (f == -1 ? 1 : x[f].ch[a]);
      x[cur].ch[a] = int(x.size());
      x.emplace_back(cur, nf, i - x[cur].len() - 1, i + 1);
    }
    cur = x[cur].ch[a];
    return cur;
  } // hash-cpp-2 = 304d80d300ae59c03bad13256f8fc973

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

Geometry (8)

8.1 2D

base.hpp

Description: Primitive operations

85 lines

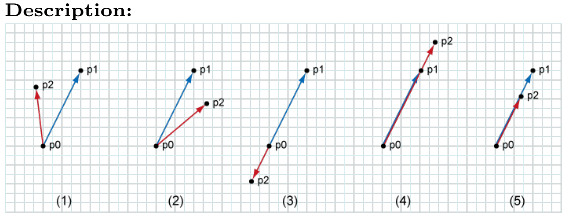
```
namespace geometry {

using std::fmod;

using D = double; // hash-cpp-1
const D EPS = D(1e-9);
inline int sgn(D a) { return (a > EPS) - (a < -EPS); }
inline int sgn(D a, D b) { return sgn(a - b); } // hash-cpp
  ↪-1 = eb6175a3f198588d18a518264d1eee5d

const D PI = acos(D(-1));
```

```cpp
template <class T = D> struct Point {
  using P = Point; // hash-cpp-2
  T x, y;
  Point(T x_ = T(), T y_ = T()) : x(x_), y(y_) {} // hash-cpp-2 = 6494c3c9bfac161e2c65d414a2c7bc83

  P& operator += (const P& p) { x += p.x, y += p.y; return *this; } // hash-cpp-3
  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-3 = 32704ee5f47251cb7a5a8bcddb7996e3

  P& operator *= (const T& t) { x *= t, y *= t; return *this; } // hash-cpp-4
  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-4 = 56a8dfabc9e0968b82d5006dda2d4d7e

  friend D dot(const P& a, const P& b) { return a.x * b.x + a.y * b.y; }
  friend D 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-5
    int z = sgn(a.x, b.x);
    return z ? z : sgn(a.y, b.y);
  } // hash-cpp-5 = 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 D arg(const P& p) { return atan2(p.y, p.x); }

  friend D 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(D m, D a) { return P(m * cos(a), m * sin(a)); }
};
using P = Point<D>;
```

```cpp
inline int sgncrs(const P& a, const P& b) { // hash-cpp-6
  D cr = crs(a, b);
  if (abs(cr) <= (rabs(a) + rabs(b)) * EPS) return 0;
  return (cr < 0 ? -1 : 1);
} // hash-cpp-6 = 8f483e87d2a4e2beaa58dec02e9b0dca

// not tested
inline D norm_angle(D a) { // hash-cpp-7
  D res = fmod(a + PI, 2*PI);
  if (res < 0) res += PI;
  else res -= PI;
  return res;
} // hash-cpp-7 = af057ce01a3fcce81b04c1504548eb73

// not tested
inline D norm_nonnegative(D a) { // hash-cpp-8
  D res = fmod(a, 2*PI);
  if (res < 0) res += 2*PI;
  return res;
} // hash-cpp-8 = b899a21e5dbdcde83a81a840e5f9e328

// arg given lengths a, b, c,
// assumming a, b, c are valid
inline D arg(D a, D b, D c) { // hash-cpp-9
  return acos(std::clamp<D>((a * a + b * b - c * c) / (2 * a * b), -1, 1));
} // hash-cpp-9 = 81e5f19cca6e81d3d02d386a7c106ba9

} // namespace geometry
```

ccw.hpp

Description:



"geometry/base.hpp" 22 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)
inline int ccw(const P& a, const P& b) { // hash-cpp-1
  int s = sgncrs(a, b);
  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 = fdf5d91850a67e77c2432aec81e836eb

inline int ccw(const P& a, const P& b, const P& c) {
  return ccw(b-a, c-a);
}

} // namespace geometry
```

linear.hpp

Description: Line/segment operations

"geometry/ccw.hpp" 76 lines

```cpp
namespace geometry {

// Work in progress
struct L { // 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 D arg(const L& l) { return arg(vec(l)); }
}; // hash-cpp-1 = 87c781f4f81ba18dc33d97a7d3de1743

inline P project(const L& l, const P& p) { // hash-cpp-2
  P v = vec(l);
  return l.s + v * dot(v, p - l.s) / dist2(v);
} // hash-cpp-2 = 6c1b8640934518c28805ff8abd24ab79

inline int ccw(const L& l, const P& p) { return ccw(l.s, l.t, p); }

inline bool insSL(const L& s, const L& l) { // hash-cpp-3
  int a = ccw(l, s.s), b = ccw(l, s.t);
  return (a % 2 == 0 || b % 2 == 0 || a != b);
} // hash-cpp-3 = f4cae3c1b1b14b35890348251586bbcc

inline bool insSS(const L& s, const L& t) { // hash-cpp-4
  int a = ccw(s, t.s), b = ccw(s, t.t),
    c = ccw(t, s.s), d = ccw(t, s.t);
  return (a * b <= 0 && c * d <= 0);
} // hash-cpp-4 = a8ed5652fe62541c10c7ea4b729906c4

inline D distLP(const L& l, const P& p) { // hash-cpp-5
  return abs(crs(vec(l), p - l.s)) / dist(l);
} // hash-cpp-5 = 912c601a37c109fbf50619d480ee4967

inline D distSP(const L& s, const P& p) { // hash-cpp-6
  P q = project(s, p);
  if (ccw(s, q) == 0) {
    return dist(p - q);
  } else {
    return min(dist(s.s - p), dist(s.t - p));
  }
} // hash-cpp-6 = 1606015a080bef59202968db31b60baa

inline D distSS(const L& s, const L& t) { // hash-cpp-7
  if (insSS(s, t)) return 0;
  return min({
    distSP(s, t.s), distSP(s, t.t), distSP(t, s.s), distSP(t, s.t)
  });
} // hash-cpp-7 = 7213f72dd7063b6226348e3eb1c0dbcc

// TODO: usage
inline int crossLL(const L& l, const L& m, P& r) { // hash-cpp-8
  P vl = vec(l), vm = vec(m);
  D 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-8 = 8518d588ab977248305ed2ff949b418f
```

```cpp
// TODO usage
inline int crossSS(const L& l, const L& m, P& r) { // hash-
  ↪cpp-9
  int u = crossLL(l, m, r);
  if (u == 0) return 0;
  if (u == -1) {
    r = max(min(l.s, l.t), min(m.s, m.t));
    P 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-9 = 57a9b2ceb12937b59727715a7da092c9

} // namespace geometry
```

polygonal.hpp

Description: Polygon operations

"geometry/ccw.hpp", "geometry/linear.hpp" 123 lines

```cpp
namespace geometry {

inline D area2(const Vec<P>& pol) { // hash-cpp-1
  if (pol.empty()) return 0;
  D res = 0;
  P a = pol.back();
  for (P b : pol) {
    res += crs(a, b);
    a = b;
  }
  return res;
} // hash-cpp-1 = 59a8b66d01f97cf1f8b877f3d93c4ebb

// (1:left) | (2: right) is inside between v[i] -- v[i + 1]
inline Vec<pair<P, int>> insPolL(const Vec<P>& pol, const L
  ↪& l) {
  using Pi = pair<P, int>;
  Vec<Pi> v;
  P a, b = pol.back();
  for (auto c: pol) {
    a = b; b = c;
    P 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(v.begin(), v.end(), [&](Pi x, Pi y){
    auto vl = vec(l);
    return dot(vl, x.first - l.s) < dot(vl, y.first - l.s);
  });
  int m = int(v.size());
  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;
}

// 0: outside, 1: on line, 2: inside
inline int contains(const Vec<P>& pol, const P& p) { //
  ↪hash-cpp-2
  if (!pol.size()) return 0;
  int in = -1;
  P a_, b_ = pol.back();
  for (auto c : pol) {
    a_ = b_, b_ = c;
    P 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-2 = e498813e5d4b7c7c1590d7b8f49c7a82

// pol: sorted and distinct
inline Vec<P> convex_lower(const Vec<P>& pts) { // hash-cpp
  ↪-3
  assert(pts.size() >= 2);
  Vec<P> res;
  for (P d : pts) {
    while (res.size() > 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-3 = 84ae9e414bbe71bf04258ee6bc5a18e2

inline Vec<P> convex(Vec<P> pts) { // hash-cpp-4
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(), pts.end()), pts.end());
  if (pts.size() <= 1) return pts;
  Vec<P> lo = convex_lower(pts);
  reverse(pts.begin(), pts.end());
  Vec<P> up = convex_lower(pts);
  lo.insert(lo.begin(), up.begin() + 1, up.end() - 1);
  return lo;
} // hash-cpp-4 = a27d22a6e1dd23363fffe8a81eda74e8

inline Vec<P> convex_cut(const Vec<P>& pol, const L& l) {
  ↪// hash-cpp-5
  if (pol.empty()) return {};
  Vec<P> q;
  P a, b = pol.back();
  for (auto c : pol) {
    a = b, b = c;
    if ((ccw(l, a) % 2) * (ccw(l, b) % 2) < 0) {
      P buf;
      crossLL(l, L(a, b), buf);
      q.push_back(buf);
    }
    if (ccw(l, b) != -1) q.push_back(b);
  }
  return q;
} // hash-cpp-5 = 290b74700980b5649c4b5669b0fc26ae

// pol: convex
inline D diameter(const Vec<P>& pol) { // hash-cpp-6
  int n = int(pol.size());
  if (n == 2) return dist(pol[1] - pol[0]);
  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;
  }
  D ans = 0;
  int sx = x, sy = y;
  while (sx != y || sy != x) {
    ans = max(ans, dist(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;
    }
  }
  return ans;
} // hash-cpp-6 = cd85d2573e817560d6c417f9ba23eeb2

} // namespace geometry
```

circular.hpp

Description: Circle operations

"geometry/base.hpp", "geometry/linear.hpp" 96 lines

```cpp
namespace geometry {

struct C {
  P c;
  D r;
  C(P c_ = P(), D r_ = D()) : c(c_), r(r_) {}

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

// NOT TESTED
// 0: outside; 1: on; 2: inside
inline int contains(const C& c, const P& 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
inline int insCC(const C& a, const C& b) {
  D 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 = 979142900
    ↪f69a2d13e301cb10a244248
}

inline C incircle(const P& a, const P& b, const P& c) {
  D da = dist(b - c); // hash-cpp-2
  D db = dist(a - c);
  D dc = dist(a - b);
  D s = da + db + dc;
  return C(
    (a * da + b * db + c * dc) / s,
    abs(crs(b-a, c-a)) / s
  ); // hash-cpp-2 = da4c09a96ebd36bc33b998f386d287b0
}

inline C outcircle(const P& a, P b, P c) {
```

```cpp
  b -= a, c -= a; // hash-cpp-3
  D bb = dist2(b) / 2;
  D cc = dist2(c) / 2;
  D g = crs(b, c);
  D x = (bb * c.y - b.y * cc) / g;
  D y = (b.x * cc - bb * c.x) / g;
  D r = sqrt(x * x + y * y);
  x += a.x, y += a.y;
  return C(P(x, y), r); // hash-cpp-3 = 275
    ↪f22a44380f4fd85b22d3b89f2182f
}

inline int crossCL(const C& c, const L& l, array<P, 2>& res
  ↪) {
  D u = distLP(l, c.c); // hash-cpp-4
  int t = sgn(u, c.r);
  if (t == 1) return 0;
  P v = project(l, c.c);
  P d = (t == 0 ? P(0, 0) : vec(l) * (sqrt(c.r * c.r - u *
    ↪u) / dist(l)));
  res = {v - d, v + d};
  return 1 - t; // hash-cpp-4 = 3480
    ↪cdacc62ec62303c83c9faca0f432
}

// args of two intersections r, l seen by a.c,
// assuming two circles cross
inline pair<D, D> crossCC_args(const C& a, const C& b) {
  P diff = b.c - a.c; // hash-cpp-5
  D c = arg(diff);
  D d = arg(a.r, dist(diff), b.r);
  return {c - d, c + d}; // hash-cpp-5 = 67
    ↪be5dcad05298bc4764693e9f72213a
}

inline int crossCC(const C& a, const C& b, array<P, 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
}

inline int tangent(const C& c, const P& p, array<P, 2>& res
  ↪) {
  P diff = p - c.c; // hash-cpp-7
  D dd = dist(diff);
  int t = sgn(c.r, dd);
  if (t == 1) return 0;
  D d = acos(min<D>(c.r / dd, 1));
  D a = arg(diff);
  res = {eval(c, a - d), eval(c, a + d)};
  return 1 - t; // hash-cpp-7 = 211
    ↪c53df8a8ec84bad176103bf68217d
}

} // namespace geometry
```

closest-pair.hpp

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

61 lines

```cpp
namespace closest_pair_impl {

template <class T> using P = pair<T, T>;

// PRECONDITION: There are at least 2 points
template <class T> inline tuple<T, P<T>, P<T>> closest_pair
  ↪(Vec<P<T>> pts) {
  int n = int(size(pts));
  using PT = P<T>;
  std::ranges::sort(pts,
            [](PT a, PT b) -> bool { return a.first < b.
              ↪first; });
  auto sq = [&](T a) -> T { return a * a; };
  auto dist2 = [&](PT a, PT b) -> T {
    return sq(a.first - b.first) + sq(a.second - b.second);
  };

  T d = std::numeric_limits<T>::max();
  PT pa, pb;
  auto update = [&](PT a, PT b) {
    auto nd = dist2(a, b);
    if (nd < d) {
      d = nd, pa = a, pb = b;
    }
  };

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

  for (int i = 0, f = 0; i < n; i++) {
    PT p = pts[i];
    while (f < i && sq(p.first - pts[f].first) >= d) {
      st.erase(its[f++]);
    }
    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.second - t->second) >= d) break;
      }
    }
    {
      auto t = u;
      while (true) {
        if (t == end(st)) break;
        if (sq(p.second - t->second) >= d) break;
        update(*t, p);
        t = next(t);
      }
    }
    its[i] = st.emplace_hint(u, p);
  }

  return {d, pa, pb};
}

}  // namespace closest_pair_impl
// hash-cpp-all = a4c5cd3e7888ad95f207d93422840850
```

Appendix (9)

techniques.txt

159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```