Team Reference Document

Heltion

April 4, 2024

Contents

1 Contest **1**

2 Data Structure **2**

3 Tree **5**

4 Graph **8**

5 String **14**

6 Convolution **17**

7 Number Theory **18**

8 Numerical **20**

9 Geometry **22**

10 Game **25**

1 Contest

1.1 .vscode/setting.json

374A14FFAAF9DE1413091952620CBB6D

```
{
    "editor.formatOnSave": true,
    "C_Cpp.default.cppStandard": "gnu++20"
}
```

1.2 Makefile

E44F3EF2EF7DD82148E9AD13C68D39E9

```
%:%.cpp
	g++ $< -o $@ -std=gnu++20 -O2 -Wall -Wextra -DDEBUG -
	    D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
```

1.3 .clang-format

FCF5A060748135C7FCCA0397311EEF4A

```
BasedOnStyle: Google
IndentWidth: 2
ColumnLimit: 160
```

1.4 debug.hpp

130CD7C024729AD67615D4C85F89257A

```
#include <bits/stdc++.h>
using namespace std;
template <class T, size_t size = tuple_size<T>::value>
string to_debug(T, string s = "")
  requires(not ranges::range<T>);
string to_debug(auto x)
  requires requires(ostream &os) { os << x; }
{
  return static_cast<ostringstream>(ostringstream() << x).str();
}
string to_debug(ranges::range auto x, string s = "")
  requires(not is_same_v<decltype(x), string>)
{
  for (auto xi : x) s += ",␣" + to_debug(xi);

  return "[" + s.substr(s.empty() ? 0 : 2) + "]";
}
template <class T, size_t size>
string to_debug(T x, string s)
  requires(not ranges::range<T>)
{
  [&]<size_t... I>(index_sequence<I...>) { ((s += ",␣" + to_debug(get<I
      >(x))), ...); }(make_index_sequence<size>());
  return "(" + s.substr(s.empty() ? 0 : 2) + ")";
}
#define debug(...) cerr << __FILE__ ":" << __LINE__ << ":␣(" #
    __VA_ARGS__ ")␣=␣" << to_debug(tuple(__VA_ARGS__)) << "\n"
```

1.5 main.cpp

579C3BBDA69419295352FE0AF5865E15

```
#include <bits/stdc++.h>
using namespace std;
#ifdef DEBUG
#include "debug.hpp"
#else
#define debug(...) void(0)
#endif
using i64 = int64_t;
using u64 = uint64_t;
using f64 = double_t;
int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cout << fixed << setprecision(20);
}
```

2 Data Structure

2.1 pbds

order_of_key: Returns the number of elements less than the key.
142BDF67665710D15D50D9B938A87151

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
template <class KT, class VT = null_type>
using RBTree = tree<KT, VT, std::less<KT>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

2.2 Segment Tree

product: Returns product of the elements in $[l, r]$.
D300D0A56B614E914DCEA3149E12C09C

```
template <class T, auto bop, auto e>
struct SegmentTree {
  int n;
  vector<T> s;
  SegmentTree(int n) : n(n), s(n * 2, e()) {}
  void set(int i, T v) {
    for (s[i += n] = v; i /= 2;) s[i] = bop(s[i * 2], s[i * 2 + 1]);
  }
  T product(int l, int r) {
    T rl = e(), rr = e();
    for (l += n, r += n + 1; l != r; l /= 2, r /= 2) {
      if (l % 2) rl = bop(rl, s[l++]);
      if (r % 2) rr = bop(s[--r], rr);
    }
    return bop(rl, rr);
  }
};
```

2.3 Lines

D6103FE3ACFF803F30C77B695EBBDD6A

```
struct Line {
  i64 a, b, r;
  bool operator<(Line l) { return pair(a, b) > pair(l.a, l.b); }
  bool operator<(i64 x) { return r < x; }
};
struct Lines : vector<Line> {
  static constexpr i64 inf = numeric_limits<i64>::max();
  Lines(i64 a, i64 b) : vector<Line>{{a, b, inf}} {}
  Lines(vector<Line>& lines) {
    if (not ranges::is_sorted(lines, less())) ranges::sort(lines, less
        ());
    for (auto [a, b, _] : lines) {
      for (; not empty(); pop_back()) {
        if (back().a == a) continue;
        i64 da = back().a - a, db = b - back().b;
        back().r = db / da - (db < 0 and db % da);
        if (size() == 1 or back().r > end()[-2].r) break;
      }
      emplace_back(a, b, inf);
    }
  }
  Lines operator+(Lines& lines) {
    vector<Line> res(size() + lines.size());
    ranges::merge(*this, lines, res.begin(), less());
    return Lines(res);
  }
  i64 min(i64 x) {
    auto [a, b, _] = *lower_bound(begin(), end(), x, less());
    return a * x + b;
  }
};
```

2.4 Li Chao Tree

get: Returns maxinum at x.
848B61895F096134FFAA02D06281F858

```
struct Line {
  i64 k, b;
  i64 operator()(i64 x) const { return k * x + b; }
};
template <i64 L, i64 R>
struct Segments {
  struct Node {
    optional<Line> s;
    Node *l, *r;
  };
  Node *root;
  Segments() : root(nullptr) {}
  void add(i64 l, i64 r, i64 k, i64 b) {
    auto rec = [&](auto &rec, Node *&p, i64 tl, i64 tr, Line s) -> void
        {
      if (p == nullptr) p = new Node();
      i64 tm = midpoint(tl, tr);
      if (tl >= l and tr <= r) {
        if (not p->s) return p->s = s, void();
        auto t = p->s.value();
        if (t(tl) >= s(tl)) {
          if (t(tr) >= s(tr)) return;
          if (t(tm) >= s(tm)) return rec(rec, p->r, tm + 1, tr, s);
          return p->s = s, rec(rec, p->l, tl, tm, t);
        }
        if (t(tr) <= s(tr)) return p->s = s, void();
        if (t(tm) <= s(tm)) return p->s = s, rec(rec, p->r, tm + 1, tr,
            t);
        return rec(rec, p->l, tl, tm, s);
      }
      if (l <= tm) rec(rec, p->l, tl, tm, s);
```

```
      if (r > tm) rec(rec, p->r, tm + 1, tr, s);
    };
    rec(rec, root, L, R, {k, b});
  }
  optional<i64> get(i64 x) {
    optional<i64> res = {};
    auto rec = [&](auto &rec, Node *p, i64 tl, i64 tr) -> void {
      if (p == nullptr) return;
      i64 tm = midpoint(tl, tr);
      if (p->s) {
        i64 y = p->s.value()(x);
        if (not res or res.value() < y) res = y;
      }
      if (x <= tm)
        rec(rec, p->l, tl, tm);
      else
        rec(rec, p->r, tm + 1, tr);
    };
    rec(rec, root, L, R);
    return res;
  }
};
```

2.5 Treap

split: Returns two parts such that the size of left part is k.

32D9CC866C4B2320B6017BBE47170FE6

```
mt19937_64 mt{random_device{}()};
struct Treap {
  u64 hp;
  i64 v, s;
  int size;
  bool rev;
  array<Treap*, 2> ch;
  Treap(i64 v) : hp(mt()), v(v), s(v), size(1), rev(false) { ch.fill(
      nullptr); }
  void reverse() {
    rev ^= 1;
    swap(ch[0], ch[1]);
  }
  Treap* push() {
    if (not rev) return this;
    for (auto c : ch)
      if (c) c->reverse();
    rev = false;
    return this;
  }
  Treap* set_ch(int i, Treap* chi) {
    ch[i] = chi;
    size = 1;
    s = v;
    for (auto c : ch)
      if (c) {
        size += c->size;
        s += c->s;
      }
    return this;
  }
};
Treap* merge(Treap* l, Treap* r) {
  if (not l) return r;
  if (not r) return l;
  if (l->hp > r->hp) return l->push()->set_ch(1, merge(l->ch[1], r));
  return r->push()->set_ch(0, merge(l, r->ch[0]));
}
pair<Treap*, Treap*> split(Treap* p, int k) {
  if (not p) return {};
  int left = p->push()->ch[0] ? p->ch[0]->size : 0;
  if (k <= left) {
    auto [l, r] = split(p->ch[0], k);
    return {l, p->set_ch(0, r)};
  }
  auto [l, r] = split(p->ch[1], k - left - 1);
  return {p->set_ch(1, l), r};
}
int ma
```

2.6 Segment Beats

5483E14DFFBA8D5564FFF27B15A8BDEB

```
struct Beats {
  int n;
  Beats(int n) : n(n), s(n * 2) {
    auto rec = [&](auto rec, int p, int tl, int tr) -> void {
      if (tl + 1 == tr) return;
      int tm = midpoint(tl, tr), chl = tm * 2, chr = chl + 1;
      rec(rec, chl, tl, tm);
      rec(rec, chr, tm, tr);
      s[p].pull(s[chl], s[chr]);
    };
    rec(rec, 1, 0, n);
  }
  void add(int l, int r, i64 x) {
    auto rec = [&](auto rec, int p, int tl, int tr) -> void {
      if (tl >= l and tr <= r) return s[p].add(x, tl, tr);
      int tm = midpoint(tl, tr), chl = tm * 2, chr = chl + 1;
      s[p].push(s[chl], s[chr], tl, tr);
      if (l < tm) rec(rec, chl, tl, tm);
      if (r > tm) rec(rec, chr, tm, tr);
      s[p].pull(s[chl], s[chr]);
    };
    rec(rec, 1, 0, n);
  }
  void chmin(int l, int r, i64 x) {
    auto rec = [&](auto rec, int p, int tl, int tr) -> void {
      if (s[p].mx <= x) return;
```

```
27        if (tl >= l and tr <= r and s[p].smx < x) return s[p].chmin(x);
28        int tm = midpoint(tl, tr), chl = tm * 2, chr = chl + 1;
29        s[p].push(s[chl], s[chr], tl, tr);
30        if (l < tm) rec(rec, chl, tl, tm);
31        if (r > tm) rec(rec, chr, tm, tr);
32        s[p].pull(s[chl], s[chr]);
33      };
34      rec(rec, 1, 0, n);
35    }
36    void chmax(int l, int r, i64 x) {
37      auto rec = [&](auto rec, int p, int tl, int tr) -> void {
38        if (s[p].mn >= x) return;
39        if (tl >= l and tr <= r and s[p].smn > x) return s[p].chmax(x);
40        int tm = midpoint(tl, tr), chl = tm * 2, chr = chl + 1;
41        s[p].push(s[chl], s[chr], tl, tr);
42        if (l < tm) rec(rec, chl, tl, tm);
43        if (r > tm) rec(rec, chr, tm, tr);
44        s[p].pull(s[chl], s[chr]);
45      };
46      rec(rec, 1, 0, n);
47    }
48    i64 sum(int l, int r) {
49      auto rec = [&](auto rec, int p, int tl, int tr) -> i64 {
50        if (tl >= l and tr <= r) return s[p].sum;
51        int tm = midpoint(tl, tr), chl = tm * 2, chr = chl + 1;
52        s[p].push(s[chl], s[chr], tl, tr);
53        i64 res = 0;
54        if (l < tm) res += rec(rec, chl, tl, tm);
55        if (r > tm) res += rec(rec, chr, tm, tr);
56        return res;
57      };
58      return rec(rec, 1, 0, n);
59    }
60
61  private:
62    struct Node {
63      static constexpr i64 inf = numeric_limits<i64>::max();
64      int tl, tr;
65      i64 sum, added, mn, smn, tmn, mx, smx, tmx;
66      int cmn, cmx;
67      Node() {
68        sum = added = mn = mx = 0;
69        cmn = cmx = 1;
70        smn = tmn = inf;
71        smx = tmx = -inf;
72      }
73      void pull(Node lhs, Node rhs) {
74        sum = lhs.sum + rhs.sum;
75        mn = min(lhs.mn, rhs.mn);
76        smn = min(lhs.mn == mn ? lhs.smn : lhs.mn, rhs.mn == mn ? rhs.smn
              : rhs.mn);
77        cmn = (lhs.mn == mn ? lhs.cmn : 0) + (rhs.mn == mn ? rhs.cmn : 0)
              ;
78        mx = max(lhs.mx, rhs.mx);
79        smx = max(lhs.mx == mx ? lhs.smx : lhs.mx, rhs.mx == mx ? rhs.smx
              : rhs.mx);
80        cmx = (lhs.mx == mx ? lhs.cmx : 0) + (rhs.mx == mx ? rhs.cmx : 0)
              ;
81      }
82      void push(Node& lhs, Node& rhs, int tl, int tr) {
83        if (added) {
84          int tm = midpoint(tl, tr);
85          lhs.add(added, tl, tm);
86          rhs.add(added, tm, tr);
87          added = 0;
88        }
89        if (tmn != inf) {
90          lhs.chmin(tmn);
91          rhs.chmin(tmn);
92          tmn = inf;
93        }
94        if (tmx != -inf) {
95          lhs.chmax(tmx);
96          rhs.chmax(tmx);
97          tmx = -inf;
98        }
99      }
100     void add(i64 x, int tl, int tr) {
101       sum += (tr - tl) * x;
102       added += x;
103       mn += x;
104       mx += x;
105       if (smn != inf) smn += x;
106       if (tmn != inf) tmn += x;
107       if (smx != -inf) smx += x;
108       if (tmx != -inf) tmx += x;
109     }
110     void chmin(i64 x) {
111       if (x >= mx) return;
112       sum += (x - mx) * cmx;
113       if (smn == mx) smn = x;
114       if (mn == mx) mn = x;
115       if (tmx > x) tmx = x;
116       mx = tmn = x;
117     }
118     void chmax(i64 x) {
119       if (x <= mn) return;
120       sum += (x - mn) * cmn;
121       if (smx == mn) smx = x;
122       if (mx == mn) mx = x;
123       if (tmn < x) tmn = x;
124       mn = tmx = x;
125     }
126   };
127   vector<Node> s;
128 };
```

3 Tree

3.1 Directed Minimum Spanning Tree (Rollback Union Find and Skew Heap)

12541369F4AE919F8FA1AE003C88FEB5

```
struct RollbackUnionFind {
  vector<pair<int, int>> stack;
  vector<int> uf;
  RollbackUnionFind(int n) : uf(n, -1) {}
  int find(int u) { return uf[u] < 0 ? u : find(uf[u]); }
  int time() { return ssize(stack); }
  bool merge(int u, int v) {
    if ((u = find(u)) == (v = find(v))) return false;
    if (uf[u] < uf[v]) swap(u, v);
    stack.emplace_back(u, uf[u]);
    uf[v] += uf[u];
    uf[u] = v;
    return true;
  }
  void rollback(int t) {
    while (ssize(stack) > t) {
      auto [u, uf_u] = stack.back();
      stack.pop_back();
      uf[uf[u]] -= uf_u;
      uf[u] = uf_u;
    }
  }
};
```

819128ED17730A2B04874DC740DAC011

```

struct Skew {
  int u, v;
  i64 w, lazy;
  Skew *chl, *chr;
  static Skew *merge(Skew *x, Skew *y) {
    if (not x) return y;
    if (not y) return x;
    if (x->w > y->w) swap(x, y);
    x->push();
    x->chr = merge(x->chr, y);
    swap(x->chl, x->chr);
    return x;
  }
  Skew(tuple<int, int, i64> e) : lazy(0) {
    tie(u, v, w) = e;
    chl = chr = nullptr;
  }
  void add(i64 x) {
    w += x;
    lazy += x;
  }
  void push() {
    if (chl) chl->add(lazy);
    if (chr) chr->add(lazy);
    lazy = 0;
  }
  Skew *pop() {
    push();
    return merge(chl, chr);
  }
};
```

678739B53597C5D3190EAF7921305905

```

pair<i64, vector<int>> directed_minimum_spanning_tree(int n, const
    vector<tuple<int, int, i64>> &edges, int s) {
  i64 ans = 0;
  vector<Skew *> heap(n), in(n);
  RollbackUnionFind uf(n), rbuf(n);
  vector<pair<Skew *, int>> cycles;
  for (auto [u, v, w] : edges) heap[v] = Skew::merge(heap[v], new Skew
      ({u, v, w}));
  for (int i = 0; i < n; i += 1) {
    if (i == s) continue;
    for (int u = i;;) {
      if (not heap[u]) return {};
      ans += (in[u] = heap[u])->w;
      in[u]->add(-in[u]->w);
      int v = rbuf.find(in[u]->u);
      if (uf.merge(u, v)) break;
      int t = rbuf.time();
      while (rbuf.merge(u, v)) {
        heap[rbuf.find(u)] = Skew::merge(heap[u], heap[v]);
        u = rbuf.find(u);
        v = rbuf.find(in[v]->u);
      }
      cycles.emplace_back(in[u], t);
      while (heap[u] and rbuf.find(heap[u]->u) == rbuf.find(u)) heap[u]
          = heap[u]->pop();
    }
  }
  for (auto [p, t] : cycles | views::reverse) {
    int u = rbuf.find(p->v);
    rbuf.rollback(t);
    int v = rbuf.find(in[u]->v);
    in[v] = exchange(in[u], p);
  }
  vector<int> res(n, -1);
  for (int i = 0; i < n; i += 1) res[i] = i == s ? i : in[i]->u;
  return {ans, res};
}
```

3.2 Dominator Tree

A944605F16E354D8E9429D8425FC33FC

```cpp
vector<int> dominator(const vector<vector<int>> &adj, int s) {
  int n = adj.size();
  vector<int> pos(n, -1), p, label(n), dom(n), sdom(n), dsu(n), par(n);
  vector<vector<int>> rg(n), bucket(n);
  auto dfs = [&](auto &dfs, int u) -> void {
    int t = p.size();
    p.push_back(u);
    label[t] = sdom[t] = dsu[t] = pos[u] = t;
    for (int v : adj[u]) {
      if (pos[v] == -1) {
        dfs(dfs, v);
        par[pos[v]] = t;
      }
      rg[pos[v]].push_back(t);
    }
  };
  dfs(dfs, s);
  auto find = [&](auto &find, int u, int x) {
    if (u == dsu[u]) return x ? -1 : u;
    int v = find(find, dsu[u], x + 1);
    if (v < 0) return u;
    if (sdom[label[dsu[u]]] < sdom[label[u]]) label[u] = label[dsu[u]];
    dsu[u] = v;
    return x ? v : label[u];
  };
  for (int i = 0; i < n; i += 1) dom[i] = i;
  for (int i = ssize(p) - 1; i >= 0; i -= 1) {
    for (int j : rg[i]) sdom[i] = min(sdom[i], sdom[find(find, j, 0)]);
    if (i) bucket[sdom[i]].push_back(i);
    for (int k : bucket[i]) {
      int j = find(find, k, 0);
      dom[k] = sdom[j] == sdom[k] ? sdom[j] : j;
    }
    if (i > 1) dsu[i] = par[i];
  }
  for (int i = 1; i < ssize(p); i += 1)
    if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
  vector<int> res(n, -1);
  res[s] = s;
  for (int i = 1; i < ssize(p); i += 1) res[p[i]] = p[dom[i]];
  return res;
}
```

3.3 Heavy Light Decomposition

98C94620C4E578F01013F76E27332CE7

```cpp
struct HeavyLigthDecomposition {
  vector<int> p, pos, top;
  HeavyLigthDecomposition(const vector<vector<int>>& adj) {
    int n = adj.size(), m = 0;
    p.resize(n, -1);
    pos.resize(n);
    top.resize(n);
    vector<int> size(n, 1), h(n, -1);
    auto dfs0 = [&](auto& dfs, int u) -> void {
      for (int v : adj[u]) {
        if (v == p[u]) continue;
        p[v] = u;
        dfs(dfs, v);
        size[u] += size[v];
        if (h[u] == -1 or size[h[u]] < size[v]) h[u] = v;
      }
    };
    dfs0(dfs0, 0);
    auto dfs1 = [&](auto& dfs, int u) -> void {
      pos[u] = m++;
      if (~h[u]) {
        top[h[u]] = top[u];
        dfs(dfs, h[u]);
      }
      for (int v : adj[u]) {
        if (v == p[u] or v == h[u]) continue;
        dfs(dfs, top[v] = v);
      }
    };
    dfs1(dfs1, top[0] = 0);
  }
  vector<tuple<int, int, bool>> dec(int u, int v) {
    vector<tuple<int, int, bool>> pu, pv;
    while (top[u] != top[v]) {
      if (pos[u] > pos[v]) {
        pu.emplace_back(pos[top[u]], pos[u], true);
        u = p[top[u]];
      } else {
        pv.emplace_back(pos[top[v]], pos[v], false);
        v = p[top[v]];
      }
    }
    if (pos[u] <= pos[v])
      pv.emplace_back(pos[u], pos[v], false);
    else
      pu.emplace_back(pos[v], pos[u], true);
    ranges::reverse(pv);
    pu.insert(pu.end(), pv.begin(), pv.end());
    return pu;
  }
};
```

3.4 Link Cut Tree

438C0A3982DAC34C581FC3A34675C8E0

```cpp
template <class T, auto bop, auto e>
struct Node {
  T t, s, r;
  bool rev;
```

```
  Node* p;
  array<Node*, 2> ch;
  Node(T t = e()) : t(t), s(t), r(t), rev(false) { p = ch[0] = ch[1] =
      nullptr; }
  int h() {
    for (int i : {0, 1})
      if (p and p->ch[i] == this) return i;
    return -1;
  }
  void reverse() {
    rev ^= 1;
    swap(s, r);
    swap(ch[0], ch[1]);
  }
  void pull() {
    s = bop(bop(ch[0] ? ch[0]->s : e(), t), ch[1] ? ch[1]->s : e());
    r = bop(bop(ch[1] ? ch[1]->r : e(), t), ch[0] ? ch[0]->r : e());
  }
  void push() {
    if (rev) {
      for (auto chi : ch)
        if (chi) chi->reverse();
      rev = false;
    }
  }
  void attach(int h, Node* u) {
    if ((ch[h] = u)) u->p = this;
    pull();
  }
  void rotate() {
    auto pp = p->p;
    int oh = h(), ph = p->h();
    p->attach(oh, ch[oh ^ 1]);
    attach(oh ^ 1, p);
    if (~ph) pp->attach(ph, this);
    p = pp;
  }
  void flush() {
    if (~h()) p->flush();
    push();
  }
  void splay() {
    for (flush(); ~h(); rotate())
      if (~p->h()) (h() == p->h() ? p : this)->rotate();
  }
  void access() {
    splay();
    attach(1, nullptr);
    while (p) {
      p->splay();
      p->attach(1, this);
      rotate();
    }
  }
  void make_root() {
    access();
    reverse();
    push();
  }
  void link(Node* u) {
    u->make_root();
    access();
    attach(1, u);
  }
  void cut(Node* u) {
    u->make_root();
    access();
    if (ch[0] == u) {
      ch[0] = u->p = nullptr;
      pull();
    }
  }
  void set(T t) {
    access();
    this->t = t;
    pull();
  }
  T query(Node* u) {
    make_root();
    u->access();
    return u->s;
  }
};
```

3.5 Least Common Ancestor

1276D3B3CBD007F020F32B06A3C04F22

```
struct SparseTable {
  vector<vector<int>> table;
  SparseTable() {}
  SparseTable(const vector<int>& a) {
    int n = a.size(), h = bit_width(a.size());
    table.resize(h);
    table[0] = a;
    for (int i = 1; i < h; i += 1) {
      table[i].resize(n - (1 << i) + 1);
      for (int j = 0; j + (1 << i) <= n; j += 1) table[i][j] = min(
          table[i - 1][j], table[i - 1][j + (1 << (i - 1))]);
    }
  }
  int query(int l, int r) {
    int h = bit_width(unsigned(r - l + 1)) - 1;
    return min(table[h][l], table[h][r - (1 << h) + 1]);
  }
};
struct LeastCommonAncestor {
  SparseTable st;
```

```
  vector<int> p, time, a, par;
  LeastCommonAncestor(int root, const vector<vector<int>>& adj) {
    int n = adj.size();
    time.resize(n, -1);
    par.resize(n, -1);
    auto dfs = [&](auto& dfs, int u) -> void {
      time[u] = p.size();
      p.push_back(u);
      for (int v : g[u]) {
        if (time[v] == -1) {
          par[v] = u;
          dfs(dfs, v);
        }
      }
    };
    dfs(dfs, root);
    a.resize(n);
    for (int i = 1; i < n; i += 1) a[i] = time[par[p[i]]];
    st = SparseTable(a);
  }
  int query(int u, int v) {
    if (u == v) return u;
    if (time[u] > time[v]) swap(u, v);
    return p[st.query(time[u] + 1, time[v] + 1)];
  }
};
```

4 Graph

4.1 Strongly Connected Components

5EE3A6AB55CD60246D7CD2DCC527E23B

```
vector<vector<int>> strongly_connected_components(const vector<vector<
    int>>& adj) {
  int n = adj.size();
  vector<bool> done(n);
  vector<int> pos(n, -1), stack;
  vector<vector<int>> res;
  auto dfs = [&](auto& dfs, int u) -> int {
    int low = pos[u] = stack.size();
    stack.push_back(u);
    for (int v : adj[u])
      if (not done[v]) low = min(low, ~pos[v] ? pos[v] : dfs(dfs, v));
    if (low == pos[u]) {
      res.emplace_back(stack.begin() + low, stack.end());
      for (int v : res.back()) done[v] = true;
      stack.resize(low);
    }
    return low;
  };
  for (int i = 0; i < n; i += 1)
    if (not done[i]) dfs(dfs, i);
  ranges::reverse(res);
  return res;
}
```

4.2 Two Vertex Connected Components

BFEA67F9BE4D326D372D36BDEA4EB5AD

```
vector<vector<int>> two_vertex_connected_components(const vector<vector
    <int>>& adj) {
  int n = adj.size();
  vector<int> pos(n, -1), stack;
  vector<vector<int>> res;
  auto dfs = [&](auto& dfs, int u, int p) -> int {
    int low = pos[u] = stack.size();
    bool cut = ~p;
    stack.push_back(u);
    for (int v : adj[u]) {
      if (v == p) continue;
      if (~pos[v]) {
        low = min(low, pos[v]);
        continue;
      }
      int end = stack.size(), low_v = dfs(dfs, v, u);
      low = min(low, low_v);
      if (low_v >= pos[u] and exchange(cut, true)) {
        res.emplace_back(stack.begin() + end, stack.end());
        res.back().push_back(u);
        stack.resize(end);
      }
    }
    return low;
  };
  for (int i = 0; i < n; i += 1)
    if (pos[i] == -1) {
      dfs(dfs, i, -1);
      res.emplace_back(move(stack));
    }
  return res;
}
```

4.3 Two Edge Connected Components

5A3855AB265AB88C5C71C0569A4D765A

```
vector<vector<int>> two_edge_connected_components(const vector<vector<
    int>>& adj) {
  int n = adj.size();
  vector<int> pos(n, -1), stack;
  vector<vector<int>> res;
  auto dfs = [&](auto& dfs, int u, int p) -> int {
    int low = pos[u] = stack.size();
    bool mul = false;
    stack.push_back(u);
```

```
    for (int v : adj[u]) {
      if (~pos[v]) {
        if (v != p or exchange(mul, true)) low = min(low, pos[v]);
        continue;
      }
      low = min(low, dfs(dfs, v, u));
    }
    if (low == pos[u]) {
      res.emplace_back(stack.begin() + low, stack.end());
      stack.resize(low);
    }
    return low;
  };
  for (int i = 0; i < n; i += 1)
    if (pos[i] == -1) dfs(dfs, i, -1);
  return res;
}
```

4.4 Three Edge Connected Components

A2971B9135EC1EFB65D9AEADF8B049F1

```
struct DisjointSetUnion {
  vector<int> dsu;
  DisjointSetUnion(int n) : dsu(n, -1) {}
  int find(int u) { return dsu[u] < 0 ? u : dsu[u] = find(dsu[u]); }
  void merge(int u, int v) {
    u = find(u);
    v = find(v);
    if (u == v) return;
    if (dsu[u] > dsu[v]) swap(u, v);
    dsu[u] += dsu[v];
    dsu[v] = u;
  }
};

vector<vector<int>> three_edge_connected_components(const vector<vector
    <int>> &adj) {
  int n = adj.size(), dft = -1;
  vector<int> pre(n, -1), post(n), path(n, -1), low(n), deg(n);
  DisjointSetUnion dsu(n);
  auto dfs = [&](auto &dfs, int u, int p) -> void {
    int pc = 0;
    low[u] = pre[u] = dft += 1;
    for (int v : adj[u]) {
      if (v == u or (v == p and not pc++)) continue;
      if (pre[v] != -1) {
        if (pre[v] < pre[u]) {
          deg[u] += 1;
          low[u] = min(low[u], pre[v]);
          continue;
        }
        deg[u] -= 1;
        for (int &p = path[u]; p != -1 and pre[p] <= pre[v] and pre[v]
            <= post[p];) {
          dsu.merge(u, p);
          deg[u] += deg[p];
          p = path[p];
        }
        continue;
      }
      dfs(dfs, v, u);
      if (path[v] == -1 and deg[v] <= 1) {
        low[u] = min(low[u], low[v]);
        deg[u] += deg[v];
        continue;
      }
      if (deg[v] == 0) v = path[v];
      if (low[u] > low[v]) {
        low[u] = min(low[u], low[v]);
        swap(v, path[u]);
      }
      for (; v != -1; v = path[v]) {
        dsu.merge(u, v);
        deg[u] += deg[v];
      }
    }
    post[u] = dft;
  };
  for (int i = 0; i < n; i += 1)
    if (pre[i] == -1) dfs(dfs, i, -1);
  vector<vector<int>> _res(n);
  for (int i = 0; i < n; i += 1) _res[dsu.find(i)].push_back(i);
  vector<vector<int>> res;
  for (auto &res_i : _res)
    if (not res_i.empty()) res.emplace_back(move(res_i));
  return res;
}
```

4.5 Directed Eulerian Path

DAE3F2F074EAEF67481B8A4A1888663D

```
optional<vector<int>> directed_eulerian_path(int n, const vector<pair<
    int, int>>& e) {
  vector<int> res;
  if (e.empty()) return res;
  vector<vector<int>> adj(n);
  vector<int> in(n);
  for (int i = 0; i < ssize(e); i += 1) {
    auto [u, v] = e[i];
    adj[u].push_back(i);
    in[v] += 1;
  }
  int s = -1;
  for (int i = 0; i < n; i += 1) {
    if (ssize(adj[i]) <= in[i]) continue;
    if (ssize(adj[i]) > in[i] + 1 or ~s) return {};
```

```
    s = i;
  }
  for (int i = 0; i < n and s == -1; i += 1)
    if (not adj[i].empty()) s = i;
  auto dfs = [&](auto& dfs, int u) -> void {
    while (not adj[u].empty()) {
      int j = adj[u].back();
      adj[u].pop_back();
      dfs(dfs, e[j].second);
      res.push_back(j);
    }
  };
  dfs(dfs, s);
  if (res.size() != e.size()) return {};
  ranges::reverse(res);
  return res;
}
```

4.6 Undirected Eulerian Path

3ECD5C02B83290BFC466F0114F48DD91

```
optional<vector<pair<int, bool>>> undirected_eulerian_path(int n, const
    vector<pair<int, int>>& e) {
  vector<pair<int, bool>> res;
  if (e.empty()) return res;
  vector<vector<pair<int, bool>>> adj(n);
  for (int i = 0; i < ssize(e); i += 1) {
    auto [u, v] = e[i];
    adj[u].emplace_back(i, true);
    adj[v].emplace_back(i, false);
  }
  int s = -1, odd = 0;
  for (int i = 0; i < n; i += 1) {
    if (ssize(adj[i]) % 2 == 0) continue;
    if (odd++ >= 2) return {};
    s = i;
  }
  for (int i = 0; i < n and s == -1; i += 1)
    if (not adj[i].empty()) s = i;
  vector<bool> visited(e.size());
  auto dfs = [&](auto& dfs, int u) -> void {
    while (not adj[u].empty()) {
      auto [j, k] = adj[u].back();
      adj[u].pop_back();
      if (visited[j]) continue;
      visited[j] = true;
      dfs(dfs, k ? e[j].second : e[j].first);
      res.emplace_back(j, k);
    }
  };
  dfs(dfs, s);
  if (res.size() != e.size()) return {};
  ranges::reverse(res);
  return res;
}
```

4.7 K Shortest Paths (Persistent Leftist Heap)

7903807FE203BDF3570351D57FDA02D9

```
template <class T>
struct Node {
  static int get(Node* x) { return x ? x->d : 0; }
  static Node* merge(Node* x, Node* y) {
    if (not x) return y;
    if (not y) return x;
    if (x->key > y->key) swap(x, y);
    Node* res = new Node(*x);
    res->chr = merge(res->chr, y);
    if (get(res->chr) > get(res->chl)) swap(res->chl, res->chr);
    res->d = get(res->chr) + 1;
    return res;
  }
  int d;
  T key;
  Node *chl, *chr;
  Node(T key) : d(1), key(key) { chl = chr = nullptr; }
};
```

3AD011C8C0D4DAE5F4ABBD944E9E7570

```
template <typename T>
using MinHeap = priority_queue<T, vector<T>, greater<>>;
vector<i64> k_shortest_paths(const vector<vector<pair<int, i64>>>& adj,
    int s, int t, int k) {
  int n = adj.size();
  MinHeap<pair<i64, int>> dq;
  vector<int> p(n, -1), order;
  vector<i64> d(n, -1);
  dq.emplace(d[s] = 0, s);
  while (not dq.empty()) {
    auto [du, u] = dq.top();
    dq.pop();
    if (du != d[u]) continue;
    order.push_back(u);
    for (auto [v, w] : adj[u]) {
      if (d[v] == -1 or d[v] > d[u] + w) {
        p[v] = u;
        dq.emplace(d[v] = d[u] + w, v);
      }
    }
  }
  vector<i64> res;
  res.push_back(d[t]);
  if (d[t] == -1) return res;
  using Leftist = Node<pair<i64, int>>;
  vector<Leftist*> roots(n);
  vector<int> mul(n);
  for (int u = 0; u < n; u += 1) {
```

```
    if (d[u] == -1) continue;
    for (auto [v, w] : adj[u]) {
      if (d[v] == -1) continue;
      w += d[u] - d[v];
      if (p[v] != u or w or exchange(mul[v], 1)) roots[v] = Leftist::
          merge(roots[v], new Node(pair(w, u)));
    }
  }
  for (int u : order)
    if (u != s) roots[u] = Leftist::merge(roots[u], roots[p[u]]);
  if (not roots[t]) return res;
  MinHeap<pair<i64, Leftist*>> pq;
  pq.emplace(d[t] + roots[t]->key.first, roots[t]);
  while (not pq.empty() and ssize(res) < k) {
    auto [d, p] = pq.top();
    pq.pop();
    res.push_back(d);
    auto [w, v] = p->key;
    for (auto ch : {p->chl, p->chr}) {
      if (ch) pq.emplace(d - w + ch->key.first, ch);
    }
    if (roots[v]) pq.emplace(d + roots[v]->key.first, roots[v]);
  }
  return res;
}
```

4.8 Global Minimum Cut

C254C4A91E023D0783B9BE57D1D3396E

```
i64 stoer_wagner(vector<vector<i64>> &w) {
  int n = w.size();
  if (n == 2) return w[0][1];
  vector<bool> in(n);
  vector<int> add;
  vector<i64> s(n);
  i64 st = 0;
  for (int i = 0; i < n; i += 1) {
    int k = -1;
    for (int j = 0; j < n; j += 1)
      if (not in[j] and (k == -1 or s[j] > s[k])) k = j;
    add.push_back(k);
    st = s[k];
    in[k] = true;
    for (int j = 0; j < n; j += 1) s[j] += w[j][k];
  }
  int x = add.end()[-2], y = add.back();
  if (x == n - 1) swap(x, y);
  for (int i = 0; i < n; i += 1) {
    swap(w[y][i], w[n - 1][i]);
    swap(w[i][y], w[i][n - 1]);
  }
  for (int i = 0; i + 1 < n; i += 1) {
    w[i][x] += w[i][n - 1];
    w[x][i] += w[n - 1][i];
  }
  w.pop_back();
  return min(st, stoer_wagner(w));
}
```

4.9 Dinic

BE2CB3B0B002CCD218C4B8B3BE592376

```
struct Dinic {
  int n;
  vector<tuple<int, int, i64>> e;
  vector<vector<int>> adj;
  vector<int> level;
  Dinic(int n) : n(n), adj(n) {}
  int add(int u, int v, int c) {
    int i = e.size();
    e.emplace_back(u, v, c);
    e.emplace_back(v, u, 0);
    adj[u].push_back(i);
    adj[v].push_back(i ^ 1);
    return i;
  }
  i64 max_flow(int s, int t) {
    i64 flow = 0;
    queue<int> q;
    vector<int> cur;
    auto bfs = [&]() {
      level.assign(n, -1);
      level[s] = 0;
      q.push(s);
      while (not q.empty()) {
        int u = q.front();
        q.pop();
        for (int i : adj[u]) {
          auto [_, v, c] = e[i];
          if (c and level[v] == -1) {
            level[v] = level[u] + 1;
            q.push(v);
          }
        }
      }
      return ~level[t];
    };
    auto dfs = [&](auto &dfs, int u, i64 limit) -> i64 {
      if (u == t) return limit;
      i64 res = 0;
      for (int &i = cur[u]; i < ssize(adj[u]) and limit; i += 1) {
        int j = adj[u][i];
        auto [_, v, c] = e[j];
        if (level[v] == level[u] + 1 and c)
          if (i64 d = dfs(dfs, v, min(c, limit)); d) {
            limit -= d;
```

```
            res += d;
            get<2>(e[j]) -= d;
            get<2>(e[j ^ 1]) += d;
          }
      }
      return res;
    };
    while (bfs()) {
      cur.assign(n, 0);
      while (i64 f = dfs(dfs, s, numeric_limits<i64>::max())) flow += f
          ;
    }
    return flow;
  }
};
```

4.10 Highest Label Preflow Push

8FBAA34ADE7AD3E245338319313E5A07

```
struct HighestLabelPreflowPush {
  int n;
  vector<vector<int>> adj;
  vector<tuple<int, int, i64>> e;
  HighestLabelPreflowPush(int n) : n(n), adj(n) {}
  int add(int u, int v, i64 f) {
    if (u == v) return -1;
    int i = ssize(e);
    e.emplace_back(u, v, f);
    e.emplace_back(v, u, 0);
    adj[u].push_back(i);
    adj[v].push_back(i ^ 1);
    return i;
  }
  i64 max_flow(int s, int t) {
    vector<i64> p(n);
    vector<int> h(n), cur(n), count(n * 2);
    vector<vector<int>> pq(n * 2);
    auto push = [&](int i, i64 f) {
      auto [u, v, _] = e[i];
      if (not p[v] and f) pq[h[v]].push_back(v);
      get<2>(e[i]) -= f;
      get<2>(e[i ^ 1]) += f;
      p[u] -= f;
      p[v] += f;
    };
    h[s] = n;
    count[0] = n - 1;
    p[t] = 1;
    for (int i : adj[s]) push(i, get<2>(e[i]));
    for (int hi = 0;;) {
      while (pq[hi].empty())
        if (not hi--) return -p[s];
      int u = pq[hi].back();
      pq[hi].pop_back();
      while (p[u] > 0)
        if (cur[u] == ssize(adj[u])) {
          h[u] = n * 2 + 1;
          for (int i = 0; i < ssize(adj[u]); i += 1) {
            auto [_, v, f] = e[adj[u][i]];
            if (f and h[u] > h[v] + 1) {
              h[u] = h[v] + 1;
              cur[u] = i;
            }
          }
          count[h[u]] += 1;
          if (not(count[hi] -= 1) and hi < n)
            for (int i = 0; i < n; i += 1)
              if (h[i] > hi and h[i] < n) {
                count[h[i]] -= 1;
                h[i] = n + 1;
              }
          hi = h[u];
        } else {
          int i = adj[u][cur[u]];
          auto [_, v, f] = e[i];
          if (f and h[u] == h[v] + 1)
            push(i, min(p[u], f));
          else
            cur[u] += 1;
        }
    }
    return 0;
  }
};
```

4.11 Minimum Perfect Matching on Biartite Graph

BC7F8A31264DA33B2A1A22278F5A1F3A

```
pair<i64, vector<int>> minimum_perfect_matching_on_bipartite_graph(
    const vector<vector<i64>> &w) {
  i64 n = w.size();
  vector<int> rm(n, -1), cm(n, -1);
  vector<i64> pi(n);
  auto resid = [&](int r, int c) { return w[r][c] - pi[c]; };
  for (int c = 0; c < n; c += 1) {
    int r = ranges::min(views::iota(0, n), {}, [&](int r) { return w[r
        ][c]; });
    pi[c] = w[r][c];
    if (rm[r] == -1) {
      rm[r] = c;
      cm[c] = r;
    }
  }
  vector<int> cols(n);
  for (int i = 0; i < n; i += 1) cols[i] = i;
  for (int r = 0; r < n; r += 1) {
```

```
    if (rm[r] != -1) continue;
    vector<i64> d(n);
    for (int c = 0; c < n; c += 1) d[c] = resid(r, c);
    vector<int> pre(n, r);
    int scan = 0, label = 0, last = 0, col = -1;
    [&]() {
      while (true) {
        if (scan == label) {
          last = scan;
          i64 min = d[cols[scan]];
          for (int j = scan; j < n; j += 1) {
            int c = cols[j];
            if (d[c] <= min) {
              if (d[c] < min) {
                min = d[c];
                label = scan;
              }
              swap(cols[j], cols[label++]);
            }
          }
          for (int j = scan; j < label; j += 1)
            if (int c = cols[j]; cm[c] == -1) {
              col = c;
              return;
            }
        }
        int c1 = cols[scan++], r1 = cm[c1];
        for (int j = label; j < n; j += 1) {
          int c2 = cols[j];
          i64 len = resid(r1, c2) - resid(r1, c1);
          if (d[c2] > d[c1] + len) {
            d[c2] = d[c1] + len;
            pre[c2] = r1;
            if (len == 0) {
              if (cm[c2] == -1) {
                col = c2;
                return;
              }
              swap(cols[j], cols[label++]);
            }
          }
        }
      }
    }();
    for (int i = 0; i < last; i += 1) {
      int c = cols[i];
      pi[c] += d[c] - d[col];
    }
    for (int t = col; t != -1;) {
      col = t;
      int r = pre[col];
      cm[col] = r;
      swap(rm[r], t);
    }
  }
  i64 res = 0;
  for (int i = 0; i < n; i += 1) res += w[i][rm[i]];
  return {res, rm};
}
```

4.12 Matching on General Graph

CA8A66783B2F152AB3EF0CC95FBCBFC0

```
vector<int> matching(const vector<vector<int>> &adj) {
  int n = adj.size(), count = 0;
  vector<int> matched(n, -1), f(n), p(n, -1), mark(n);
  auto augment = [&](int u) {
    while (u != -1) {
      int v = matched[p[u]];
      matched[matched[u] = p[u]] = u;
      u = v;
    }
  };
  auto lca = [&](int u, int v) {
    count += 1;
    while (true) {
      if (u == -1) swap(u, v);
      if (mark[u] == count) return u;
      mark[u] = count;
      u = matched[u] == -1 ? -1 : f[p[matched[u]]];
    }
    return 0;
  };
  for (int i = 0; i < n; i += 1)
    if (matched[i] == -1)
      [&]() {
        vector<int> type(n, -1);
        for (int i = 0; i < n; i += 1) f[i] = i;
        queue<int> q;
        type[i] = 0;
        q.push(i);
        auto up = [&](int u, int v, int w) {
          while (f[u] != w) {
            p[u] = v;
            v = matched[u];
            if (type[v] == 1) {
              type[v] = 0;
              q.push(v);
            }
            f[u] = f[v] = w;
            u = p[v];
          }
        };
        while (not q.empty()) {
          int u = q.front();
          q.pop();
          for (int v : adj[u])
```

```
            if (type[v] == -1) {
              p[v] = u;
              type[v] = 1;
              int mv = matched[v];
              if (mv == -1) return augment(v);
              q.push(mv);
              type[mv] = 0;
            } else if (not type[v] and f[u] != f[v]) {
              int w = lca(u, v);
              up(u, v, w);
              up(v, u, w);
              for (int i = 0; i < n; i += 1) f[i] = f[f[i]];
            }
          }
        }();
    return matched;
}
```

4.13 Minimum Cost Maxinum Flow

69C3DC15D81E78FB3545DD6379F6CBD1

```
struct MinimumCostMaximumFlow {
  int n;
  vector<tuple<int, int, i64, i64>> e;
  vector<vector<int>> adj;
  MinimumCostMaximumFlow(int n) : n(n), adj(n) {}
  int add_edge(int u, int v, i64 f, i64 c) {
    int i = e.size();
    e.emplace_back(u, v, f, c);
    e.emplace_back(v, u, 0, -c);
    adj[u].push_back(i);
    adj[v].push_back(i + 1);
    return i;
  }
  pair<i64, i64> flow(int s, int t) {
    constexpr i64 inf = numeric_limits<i64>::max();
    vector<i64> d, h(n);
    vector<int> p;
    auto dijkstra = [&]() {
      d.assign(n, inf);
      p.assign(n, -1);
      priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<
          pair<i64, int>>> q;
      q.emplace(d[s] = 0, s);
      while (not q.empty()) {
        auto [du, u] = q.top();
        q.pop();
        if (du != d[u]) continue;
        for (int i : adj[u]) {
          auto [_, v, f, c] = e[i];
          if (f and d[v] > d[u] + h[u] - h[v] + c) {
            p[v] = i;
            q.emplace(d[v] = d[u] + h[u] - h[v] + c, v);
          }
        }
      }
      return ~p[t];
    };
    i64 f = 0, c = 0;
    while (dijkstra()) {
      for (int i = 0; i < n; i += 1) h[i] += d[i];
      vector<int> path;
      for (int u = t; u != s; u = get<0>(e[p[u]])) path.push_back(p[u])
          ;
      i64 mf = get<2>(e[ranges::min(path, {}, [&](int i) { return get
          <2>(e[i]); })]);
      f += mf;
      c += mf * h[t];
      for (int i : path) {
        get<2>(e[i]) -= mf;
        get<2>(e[i ^ 1]) += mf;
      }
    }
    return {f, c};
  }
};
```

5 String

5.1 Z

6F6DBB227709B41D81A4F9B31A566DDF

```
vector<int> fz(const string& s) {
  int n = s.size();
  vector<int> z(n);
  for (int i = 1, j = 0; i < n; i += 1) {
    z[i] = max(min(j + z[j] - i, z[i - j]), 0);
    while (s[z[i]] == s[i + z[i]]) z[i] += 1;
    if (i + z[i] > j + z[j]) j = i;
  }
  z[0] = n;
  return z;
}
```

5.2 Manacher

935EDD60183B12CBED8917FD0AA462AF

```
vector<int> fp(const string& s) {
  int n = s.size();
  vector<int> p(n * 2 - 1);
  for (int i = 0, j = 0; i < n * 2 - 1; i += 1) {
    if (j + p[j] > i) p[i] = min(j + p[j] - i, p[2 * j - i]);
    while (i >= p[i] and i + p[i] <= 2 * n and ((i - p[i]) % 2 == 0 or
        s[(i - p[i]) / 2] == s[(i + p[i] + 1) / 2])) p[i] += 1;
```

```
    if (i + p[i] > j + p[j]) j = i;
  }
  return p;
}
```

5.3 Lyndon Factorization

86B6B58329D25955C7FD43781BDD6AEC

```
vector<int> lyndon_factorization(string const &s) {
  int n = s.size();
  vector<int> res = {0};
  for (int i = 0; i < n;) {
    int j = i + 1, k = i;
    for (; j < n and s[k] <= s[j]; j += 1) k = s[k] < s[j] ? i : k + 1;
    while (i <= k) res.push_back(i += j - k);
  }
  return res;
}
```

5.4 Run (Suffix Array and Longest Common Prefix of Suffix)

A18A28732B85A91584A14C7F64F0231C

```
struct LongestCommonPrefix {
  int n;
  vector<int> p, rank;
  vector<vector<int>> st;
  LongestCommonPrefix(const string &s) : n(s.size()), p(n), rank(n) {
    int k = 0;
    vector<int> q, count;
    for (int i = 0; i < n; i += 1) p[i] = i;
    ranges::sort(p, {}, [&](int i) { return s[i]; });
    for (int i = 0; i < n; i += 1) rank[p[i]] = i and s[p[i]] == s[p[i
        - 1]] ? rank[p[i - 1]] : k++;
    for (int m = 1; m < n; m *= 2) {
      q.resize(m);
      for (int i = 0; i < m; i += 1) q[i] = n - m + i;
      for (int i : p)
        if (i >= m) q.push_back(i - m);
      count.assign(k, 0);
      for (int i : rank) count[i] += 1;
      for (int i = 1; i < k; i += 1) count[i] += count[i - 1];
      for (int i = n - 1; i >= 0; i -= 1) p[count[rank[q[i]]] -= 1] = q
          [i];
      auto cur = rank;
      cur.resize(2 * n, -1);
      k = 0;
      for (int i = 0; i < n; i += 1) rank[p[i]] = i and cur[p[i]] ==
          cur[p[i - 1]] and cur[p[i] + m] == cur[p[i - 1] + m] ? rank[p
          [i - 1]] : k++;
    }
    st.emplace_back(n);
    for (int i = 0, k = 0; i < n; i += 1) {
      if (not rank[i]) continue;
      k = max(k - 1, 0);
      int j = p[rank[i] - 1];
      while (i + k < n and j + k < n and s[i + k] == s[j + k]) k += 1;
      st[0][rank[i]] = k;
    }
    for (int i = 1; (1 << i) < n; i += 1) {
      st.emplace_back(n - (1 << i) + 1);
      for (int j = 0; j <= n - (1 << i); j += 1) st[i][j] = min(st[i -
          1][j], st[i - 1][j + (1 << (i - 1))]);
    }
  }
  int get(int i, int j) {
    if (i == j) return n - i;
    if (i == n or j == n) return 0;
    i = rank[i];
    j = rank[j];
    if (i > j) swap(i, j);
    int k = bit_width(u64(j - i)) - 1;
    return min(st[k][i + 1], st[k][j - (1 << k) + 1]);
  }
};
vector<tuple<int, int, int>> run(const string &s) {
  int n = s.size();
  auto r = s;
  ranges::reverse(r);
  LongestCommonPrefix lcp(s), lcs(r);
  vector<tuple<int, int, int>> runs;
  for (bool inv : {false, true}) {
    vector<int> lyn(n, n), stack;
    for (int i = 0; i < n; i += 1) {
      while (not stack.empty()) {
        int j = stack.back(), k = lcp.get(i, j);
        if (i + k < n and ((s[i + k] > s[j + k]) ^ inv)) break;
        lyn[j] = i;
        stack.pop_back();
      }
      stack.push_back(i);
    }
    for (int i = 0; i < n; i += 1) {
      int j = lyn[i], t = j - i, l = i - lcs.get(n - i, n - j), r = j +
          lcp.get(i, j);
      if (r - l >= 2 * t) runs.emplace_back(t, l, r);
    }
  }
  ranges::sort(runs);
  runs.erase(ranges::unique(runs).begin(), runs.end());
  return runs;
}
```

5.5 Aho-Corasick

8B5C8AEB6B2D4217BE99B61721255D12

```cpp
template <int sigma = 26, char first = 'a'>
struct AhoCorasick {
  struct Node : array<int, sigma> {
    int link;
    Node() : link(0) { this->fill(0); }
  };
  vector<Node> nodes;
  AhoCorasick() : nodes(1) {}
  int insert(const string& s) {
    int p = 0;
    for (char c : s) {
      int ci = c - first;
      if (not nodes[p][ci]) {
        nodes[p][ci] = nodes.size();
        nodes.emplace_back();
      }
      p = nodes[p][ci];
    }
    return p;
  }
  void init() {
    queue<int> q;
    q.push(0);
    while (not q.empty()) {
      int u = q.front();
      q.pop();
      for (int i = 0; i < sigma; i += 1) {
        int &v = nodes[u][i], w = nodes[nodes[u].link][i];
        if (not v) {
          v = w;
          continue;
        }
        nodes[v].link = u ? w : 0;
        q.push(v);
      }
    }
  }
};
```

5.6 Palindrome Tree

7B6E73D28CB8226EAFBEC56EBD2AB8CF

```cpp
template <int sigma = 26, char first = 'a'>
struct PalindromeTree {
  struct Node : array<int, sigma> {
    int len, link, count;
    Node(int len) : len(len) {
      link = count = 0;
      this->fill(0);
    }
  };
  int last;
  string s;
  vector<Node> nodes;
  PalindromeTree() : last(0), nodes({0, -1}) { nodes[0].link = 1; }
  int get_link(int u, int i) {
    while (i < nodes[u].len + 1 or s[i - nodes[u].len - 1] != s[i]) u =
        nodes[u].link;
    return u;
  }
  void extend(char c) {
    int i = s.size(), ci = c - first;
    s.push_back(c);
    int cur = get_link(last, i);
    if (not nodes[cur][ci]) {
      int now = nodes.size();
      nodes.push_back(nodes[cur].len + 2);
      nodes.back().link = nodes[get_link(nodes[cur].link, i)][ci];
      nodes.back().count = nodes[nodes.back().link].count + 1;
      nodes[cur][ci] = now;
    }
    last = nodes[cur][ci];
  }
};
```

5.7 Suffix Automaton

59E725E9066C3D4CE4DC238624B8C837

```cpp
template <int sigma = 26, char first = 'a'>
struct SuffixAutomaton {
  struct Node : array<int, sigma> {
    int link, len;
    Node() : link(-1), len(0) { this->fill(-1); }
  };
  vector<Node> nodes;
  SuffixAutomaton() : nodes(1) {}
  int extend(int p, char c) {
    int ci = c - first;
    if (~nodes[p][ci]) {
      int q = nodes[p][ci];
      if (nodes[p].len + 1 == nodes[q].len) return q;
      int clone = nodes.size();
      nodes.push_back(nodes[q]);
      nodes.back().len = nodes[p].len + 1;
      while (~p and nodes[p][ci] == q) {
        nodes[p][ci] = clone;
        p = nodes[p].link;
      }
      nodes[q].link = clone;
      return clone;
    }
    int cur = nodes.size();
    nodes.emplace_back();
    nodes.back().len = nodes[p].len + 1;
    while (~p and nodes[p][ci] == -1) {
```

```
      nodes[p][ci] = cur;
      p = nodes[p].link;
    }
    if (~p) {
      int q = nodes[p][ci];
      if (nodes[p].len + 1 == nodes[q].len)
        nodes.back().link = q;
      else {
        int clone = nodes.size();
        nodes.push_back(nodes[q]);
        nodes.back().len = nodes[p].len + 1;
        while (~p and nodes[p][ci] == q) {
          nodes[p][ci] = clone;
          p = nodes[p].link;
        }
        nodes[q].link = nodes[cur].link = clone;
      }
    } else
      nodes.back().link = 0;
    return cur;
  }
};
```

6 Convolution

6.1 Convex Min Plus

$a_{i+1} - a_i \le a_{i+2} - a_{i+1}$. 4C768727332928C0D783582F71C0A5A9

```
vector<i64> min_plus(vector<i64> a, vector<i64> b) {
  int n = a.size(), m = b.size();
  vector<int> f(n + m - 1);
  auto get = [&](int i, int j) { return b[i] + a[j]; };
  auto rec = [&](auto& rec, int l, int r, int bl, int br) -> void {
    int mid = midpoint(l, r), pl = max(bl, mid - n + 1), pr = min(br,
        mid), &am = f[mid] = pl;
    for (int j = pl + 1; j <= pr; j += 1)
      if (get(j, mid - j) < get(am, mid - am)) am = j;
    if (l < mid) rec(rec, l, mid - 1, bl, am);
    if (mid < r) rec(rec, mid + 1, r, am, br);
  };
  rec(rec, 0, n + m - 2, 0, m - 1);
  vector<i64> g(n + m - 1);
  for (int i = 0; i < n + m - 1; i += 1) g[i] = get(f[i], i - f[i]);
  return g;
}
```

6.2 Fast Fourier Transform

E3F0551A52C8BA57A69C7DE8454C2860

```
void fft(vector<complex<f64>>& a, bool inverse = false) {
  int n = a.size();
  vector<int> r(n);
  for (int i = 0; i < n; i += 1) r[i] = r[i / 2] / 2 | (i % 2 ? n / 2 :
      0);
  for (int i = 0; i < n; i += 1)
    if (i < r[i]) swap(a[i], a[r[i]]);
  for (int m = 1; m < n; m *= 2) {
    complex<f64> wn(exp((inverse ? 1.i : -1.i) * numbers::pi / (f64)m))
        ;
    for (int i = 0; i < n; i += m * 2) {
      complex<f64> w = 1;
      for (int j = 0; j < m; j += 1, w = w * wn) {
        auto &x = a[i + j + m], &y = a[i + j], t = w * x;
        tie(x, y) = pair(y - t, y + t);
      }
    }
  }
  if (inverse)
    for (auto& ai : a) ai /= n;
}
```

6.3 Fast Fourier Transform on Finite Field

Primes with root 3: $7 \times 2^{26} + 1$, $29 \times 2^{57} + 1$.

638390327A0DAF82C6303152D452D846

```
void fft(vector<i64>& a, bool inverse = false) {
  int n = a.size();
  vector<int> r(n);
  for (int i = 1; i < n; i += 1) r[i] = r[i / 2] / 2 | (i % 2 ? n / 2 :
      0);
  for (int i = 0; i < n; i += 1)
    if (i < r[i]) swap(a[i], a[r[i]]);
  for (int m = 1; m < n; m *= 2) {
    i64 wm = power(inverse ? g : power(g, mod - 2), (mod - 1) / m / 2);
    for (int i = 0; i < n; i += m * 2)
      for (int j = 0, w = 1; j < m; j += 1, w = w * wm % mod) {
        i64 &x = a[i + j + m], &y = a[i + j], t = w * x % mod;
        tie(x, y) = pair((y + mod - t) % mod, (y + t) % mod);
      }
  }
  if (i64 in = power(n, mod - 2); inverse)
    for (int i = 0; i < n; i += 1) a[i] = a[i] * in % mod;
}
```

6.4 Newton's Method

If $G(h) = 0$, $h_{i+1} = h_i - \frac{G(h_i)}{G'(h_i)}$.

Example:

- If $h = \frac{1}{f}$, $h_{i+1} = h_i(2 - h_i f)$.

- If $h = \sqrt{f}$, $h_{i+1} = \dfrac{h_i + \frac{f}{h_i}}{2}$.
- For $f = pg + r$, $p^T = f^T g^T - 1$.
- For $h = \log f$, $h = \displaystyle\int \frac{\mathrm{d}f}{f}$.
- If $h = \exp f$, $h_{i+1} = h_i(1 + f - \log h_i)$.

6.5 Interpolation

$$g(x) = \prod_i (x - x_i).$$

$$f(x) = \sum_{i=0}^{n-1} y_i \left(\prod_{j \neq i} \frac{x - x_i}{x_i - x_j}\right).$$

$$f(x) = \sum_{i=0}^{n-1} \frac{y_i}{g_i'(x)} \left(\prod_{j \neq i} x - x_j\right).$$

6.6 Circular Transform (Exclusive OR Transform)

$$A_{ij} = w_k^{ij}, A_{ij}^{-1} = \frac{1}{k} w_k^{-ij}.$$

6.7 Truncated Transform

$$\chi_i = \sum_{j=0}^{n} \frac{i}{\prod_{k=0}^{j} m_k} \bmod n \text{ for } 0 \leq i < \prod_{j=0}^{n-1} m_j.$$

7 Number Theory

7.1 Gaussian of Integers

B18EFB69F7E440F8826405C7378FB3FE

```
struct GaussInteger {
  i64 x, y;
  i64 norm() { return x * x + y * y; }
  bool operator!=(i64 r) { return y or x != r; }
  GaussInteger operator~() { return {x, -y}; }
  GaussInteger operator-(GaussInteger gi) { return {x - gi.x, y - gi.y}; }
  GaussInteger operator*(GaussInteger gi) { return {x * gi.x - y * gi.y, x * gi.y + y * gi.x}; }
  GaussInteger operator/(GaussInteger gi) {
    auto [x, y] = operator*(~gi);
    auto div_floor = [&](i64 x, i64 y) { return x / y - (x % y < 0); };
    auto div_round = [&](i64 x, i64 y) { return div_floor(2 * x + y, 2 * y); };
    return {div_round(x, gi.norm()), div_round(y, gi.norm())};
  }
  GaussInteger operator%(GaussInteger gi) { return operator-(gi*(operator/(gi))); }
};
```

7.2 Modular Sqrt

ED4C71625EB9E657C667F228AB5952B0

```
optional<i64> sqrt_mod(i64 y, i64 p) {
  if (y <= 1) return y;
  auto power = [&]<class T>(auto mul, T a, i64 r, auto res) {
    for (; r; r >>= 1, a = mul(a, a))
      if (r & 1) res = mul(res, a);
    return res;
  };
  auto mul_mod = [&](i64 x, i64 y) { return x * y % p; };
  if (power(mul_mod, y, (p - 1) / 2, 1) != 1) return {};
  i64 x, w;
  do {
    x = random_device()() % p;
    w = (x * x + p - y) % p;
  } while (power(mul_mod, w, (p - 1) / 2, 1) == 1);
  using P = pair<i64, i64>;
  auto mul_pair = [&](P p0, P p1) {
    auto [x0, y0] = p0;
    auto [x1, y1] = p1;
    return pair((x0 * x1 + y0 * y1 % p * w) % p, (x0 * y1 + y0 * x1) % p);
  };
  return power(mul_pair, P(x, 1), (p + 1) / 2, P(1, 0)).first;
}
```

7.3 Modular Logarithm

3A2E55E2D78E3EC281F5C05C3EE23346

```
optional<i64> log_mod(i64 x, i64 y, i64 m) {
  if (y == 1 or m == 1) return 0;
  if (not x) return y ? nullopt : optional(1);
  i64 k = 0, z = 1;
  for (i64 d; z != y and (d = gcd(x, m)) != 1; k += 1) {
    if (y % d) return {};
    m /= d;
    y /= d;
    z = z * (x / d) % m;
  }
  if (z == y) return k;
  unordered_map<i64, i64> mp;
  i64 p = 1, n = sqrt(m);
```

```
  for (int i = 0; i < n; i += 1, p = p * x % m) mp[y * p % m] = i;
  z = z * p % m;
  for (int i = 1; i <= n; i += 1, z = z * p % m)
    if (mp.contains(z)) return k + i * n - mp[z];
  return {};
}
```

7.4 Miller Rabin and Pollard Rho

1AE3286B5491F74907F40D7E680C2E0C

```
using i128 = __int128_t;
i64 power(i64 a, i64 r, i64 mod) {
  i64 res = 1;
  for (; r; r >>= 1, a = (i128)a * a % mod)
    if (r & 1) res = (i128)res * a % mod;

  return res;
}
bool miller_rabin(i64 n) {
  static constexpr array<int, 9> p = {2, 3, 5, 7, 11, 13, 17, 19, 23};
  if (n == 1) return false;
  if (n == 2) return true;
  if (n % 2 == 0) return false;
  int r = countr_zero(u64(n - 1));
  i64 d = (n - 1) >> r;
  for (int pi : p) {
    if (pi < n) {
      i64 x = power(pi, d, n);
      if (x == 1 or x == n - 1) continue;
      for (int j = 1; j < r; j += 1) {
        x = (i128)x * x % n;
        if (x == n - 1) break;
      }
      if (x != n - 1) return false;
    }
  }
  return true;
};
```

C07DC77ACE1C5990889EE1E42798B1B2

```
vector<i64> pollard_rho(i64 n) {
  if (n == 1) return {};
  vector<i64> res, stack = {n};
  while (not stack.empty()) {
    i64 n = stack.back();
    stack.pop_back();
    if (miller_rabin(n)) {
      res.push_back(n);
      continue;
    }
    i64 d = n;
    for (i64 c = random_device()() % n; d == n; c += 1) {
      d = 1;
      for (i64 k = 1, y = 0, x = 0, s = 1; d == 1; k <<= 1, y = x, s =
          1)
        for (int i = 1; i <= k; i += 1) {
          x = ((i128)x * x + c) % n;
          s = (i128)s * abs(x - y) % n;
          if (not(i % 63) or i == k) {
            d = gcd(s, n);
            if (d != 1) break;
          }
        }
      }
      stack.push_back(d);
      stack.push_back(n / d);
    };
    return res;
}
```

7.5 Extended Euclidean

1CDFD21D3A0E853D9DCFDC4976CAC641

```
template <typename T>
tuple<T, T, T> exgcd(T a, T b) {
  T x = 1, y = 0, x1 = 0, y1 = 1;
  while (b) {
    T q = a / b;
    tie(x, x1) = pair(x1, x - q * x1);
    tie(y, y1) = pair(y1, x - q * y1);
    tie(a, b) = pair(b, a - q * b);
  }
  return {a, x, y};
}
template <typename T>
optional<pair<T, T>> crt(T a0, T b0, T a1, T b1) {
  auto [d, x, y] = exgcd(a0, a1);
  if ((b1 - b0) % d) return {};
  T a = a0 / d * a1, b = (b1 - b0) / d * x % (a1 / d);
  if (b < 0) b += a1 / d;
  b = (a0 * b + b0) % a;
  if (b < 0) b += a;
  return {{a, b}};
}
```

7.6 Sum of Floor of Linear

00ED0F1DDDE601F3E4C2F0439C7B2625

```
i64 sum_of_floor(i64 n, i64 m, i64 a, i64 b) {
  i64 res = 0;
  while (n) {
    res += a / m * n * (n - 1) / 2;
    a %= m;
    res += b / m * n;
    b %= m;
```

```
    i64 y = a * n + b;
    if (y < m) break;
    tie(n, m, a, b) = tuple(y / m, a, m, y % m);
  }
  return res;
}
```

7.7 Mininum of Modulo of Linear

3C1C5590B7B339560B0C942BB9340E9B

```
i64 min_of_mod(i64 n, i64 m, i64 a, i64 b, bool rev = false, i64 p = 1,
     i64 q = 1) {
  if (not a) return b;
  if (rev) {
    if (b < m - a) {
      i64 t = (m - b - 1) / a, d = t * p;
      if (n <= d) return (n - 1) / p * a + b;
      n -= d;
      b += a * t;
    }
    b = m - 1 - b;
  } else {
    if (b >= a) {
      i64 t = (m - b + a - 1) / a, d = (t - 1) * p + q;
      if (n <= d) return b;
      n -= d;
      b += a * t - m;
    }
    b = a - 1 - b;
  }
  return (rev ? m : a) - 1 - min_of_mod(n, a, m % a, b, not rev, (m / a
       - 1) * p + q, m / a * p + q);
}
```

7.8 Stern Brocot Tree

B1074711F1E3432069DB519A2144CD2F

```
struct Node {
  int a, b;
  vector<pair<i64, char>> p;
  Node(i64 a, i64 b) : a(a), b(b) {
    assert(gcd(a, b) == 1);
    while (a != 1 or b != 1)
      if (a > b) {
        int k = (a - 1) / b;
        p.emplace_back(k, 'R');
        a -= k * b;
      } else {
        int k = (b - 1) / a;
        p.emplace_back(k, 'L');
        b -= k * a;
      }
  }
  Node(vector<pair<i64, char>> p, i64 _a = 1, i64 _b = 1) : a(_a), b(_b
      ), p(p) {
    for (auto [c, d] : p | views::reverse)
      if (d == 'R')
        a += c * b;
      else
        b += c * a;
  }
};
```

8 Numerical

8.1 Trigonometric Function

$$\frac{\mathrm{d}}{\mathrm{d}x}\arcsin x = \frac{1}{\sqrt{1-x^2}},$$
$$\frac{\mathrm{d}}{\mathrm{d}x}\arccos x = -\frac{1}{\sqrt{1-x^2}},$$
$$\frac{\mathrm{d}}{\mathrm{d}x}\arctan x = \frac{1}{1+x^2},$$
$$\frac{\mathrm{d}}{\mathrm{d}x}\tan x = 1+\tan^2 x,$$
$$\int \tan x \mathrm{d}x = -\log\cos x.$$

8.2 Green Formula

$$\iint (\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}) = \oint P\mathrm{d}x + Q\mathrm{d}y.$$

8.3 Double Integral Substitution

$$\iint f(x,y)\mathrm{d}x\mathrm{d}y = \iint f(x(u,v),y(u,v)) \left|\begin{matrix}\frac{\partial x}{\partial u} & \frac{\partial x}{\partial v}\\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v}\end{matrix}\right| \mathrm{d}u\mathrm{d}v.$$

Specially, $\iint f(x,y)\mathrm{d}x\mathrm{d}y = \iint f(r\cos\theta, r\sin\theta) r\mathrm{d}r\mathrm{d}\theta.$

8.4 Golden Search

8090B9F5D8D1CAFE4C29DB05D3972384

```
template <int step>
f64 local_minimum(auto& f, f64 l, f64 r) {
  auto get = [&](f64 l, f64 r) { return (numbers::phi - 1) * l + (2 -
      numbers::phi) * r; };
  f64 ml = get(l, r), mr = get(r, l), fml = f(ml), fmr = f(mr);
  for (int _ = 0; _ < step; _ += 1)
```

```
    if (fml > fmr) {
      l = exchange(ml, mr);
      fml = exchange(fmr, f(mr = get(r, l)));
    } else {
      r = exchange(mr, ml);
      fmr = exchange(fml, f(ml = get(l, r)));
    }
  return midpoint(l, r);
}
```

8.5 Adaptive Simpson

2F056B986BF14AA30558D1E44E119993

```
f64 simpson(auto& f, f64 l, f64 r) { return (r - l) * (f(l) + f(r) + 4 * f(midpoint(l, r))) / 6; }
f64 adaptive_simpson(auto&& f, f64 l, f64 r, f64 eps) {
  f64 m = midpoint(l, r);
  f64 s = simpson(f, l, r);
  f64 sl = simpson(f, l, m);
  f64 sr = simpson(f, m, r);
  f64 d = sl + sr - s;
  if (abs(d) < 15 * eps) return (sl + sr) + d / 15;
  return adaptive_simpson(f, l, m, eps / 2) + adaptive_simpson(f, m, r, eps / 2);
}
```

8.6 Simplex

45C0107D5A45D4F9442E835FF27D6551

```
template <typename T = long double>
struct Simplex {
  static constexpr T eps = 1e-9;
  int n, m;
  T z;
  vector<vector<T>> a;
  vector<T> b, c;
  vector<int> base;
  Simplex(int n, int m) : n(n), m(m), z(0), a(m, vector<T>(n)), b(m), c(n), base(n + m) {
    for (int i = 0; i < n + m; i += 1) base[i] = i;
  }
  void pivot(int out, int in) {
    swap(base[out + n], base[in]);
    T f = 1 / a[out][in];
    for (T &aij : a[out]) aij *= f;
    b[out] *= f;
    a[out][in] = f;
    for (int i = 0; i <= m; i += 1)
      if (i != out) {
        auto &ai = i == m ? c : a[i];
        T &bi = i == m ? z : b[i];
        T f = -ai[in];
        if (f < -eps or f > eps) {
          for (int j = 0; j < n; j += 1) ai[j] += a[out][j] * f;
          ai[in] = a[out][in] * f;
          bi += b[out] * f;
        }
      }
  }
  bool feasible() {
    while (true) {
      int i = ranges::min_element(b) - b.begin();
      if (b[i] > -eps) break;
      int k = -1;
      for (int j = 0; j < n; j += 1)
        if (a[i][j] < -eps and (k == -1 or base[j] > base[k])) k = j;
      if (k == -1) return false;
      pivot(i, k);
    }
    return true;
  }
  bool bounded() {
    while (true) {
      int i = ranges::max_element(c) - c.begin();
      if (c[i] < eps) break;
      int k = -1;
      for (int j = 0; j < m; j += 1)
        if (a[j][i] > eps) {
          if (k == -1)
            k = j;
          else {
            f64 d = b[j] * a[k][i] - b[k] * a[j][i];
            if (d < -eps or (d < eps and base[j] > base[k])) k = j;
          }
        }
      if (k == -1) return false;
      pivot(k, i);
    }
    return true;
  }
  vector<T> x() const {
    vector<T> res(n);
    for (int i = n; i < n + m; i += 1)
      if (base[i] < n) res[base[i]] = b[i - n];
    return res;
  }
};
```

9 Geometry

9.1 2D Geometry

9.1.1 Point

EFB833642FD0B46931468B6CAFDB219C

```
constexpr T eps = 1e-9;
int sign(T x) { return x < -eps ? -1 : x > eps; }
struct P {
  T x, y;
  explicit P(T x = 0, T y = 0) : x(x), y(y) {}
  P operator+(P p) { return P(x + p.x, y + p.y); }
  P operator-(P p) { return P(x - p.x, y - p.y); }
  P operator-() { return P(-x, -y); }
  P operator*(T k) { return P(x * k, y * k); }
  P rot90() { return P(-y, x); }
  T cross(P p) { return x * p.y - y * p.x; }
  T dot(P p) { return x * p.x + y * p.y; }
  T left(P p) { return sign(cross(p)); }
  R length() { return hypot(x, y); };
  T length2() { return x * x + y * y; };
  bool operator==(P p) { return x == p.x and y == p.y; }
  friend ostream& operator<<(ostream& os, P p) { return os << "(" << p.
      x << ",␣" << p.y << ")"; }
};
```

9.1.2 Line

C532B75B6E310FB6A2C0F45FD2632A32

```
struct L {
  P a, b;
  explicit L(P a = P(), P b = P()) : a(a), b(b) {}
  P v() { return b - a; }
  int left(L l) { return v().left(l.v()); }
  int left(P p) { return left(L(a, p)); }
  R length() { return v().length(); };
  T length2() { return v().length2(); }
  L reverse() { return L(b, a); }
  int inside(P p) {
    if (left(p) != 0) return -1;
    T pa = (p - a).dot(v()), pb = (p - b).dot(-v());
    return pa < 0 or pb < 0 ? -1 : pa > 0 and pb > 0;
  }
  friend ostream& operator<<(ostream& os, L l) { return os << "(" << l.
      a << ",␣" << l.b << ")"; }
};
```

9.1.3 Polygon

5BAD5A6F5813FBC0A7425746E27C85B1

```
struct G : vector<P> {
  G(int n = 0) : vector<P>(n) {}
  P vertex(int i) {
    int n = size();
    return at((i % n + n) % n);
  }
  L edge(int i) { return L(vertex(i), vertex(i + 1)); }
  T area2() {
    T res = 0;
    for (int i = 0; i < (int)size(); i += 1) res += vertex(i).cross(
        vertex(i + 1));
    return res;
  }
  int inside(P p) {
    int res = 0;
    for (int i = 0; i < (int)size(); i += 1) {
      auto a = vertex(i), b = vertex(i + 1);
      L l(a, b);
      if (l.inside(p) >= 0) return 0;
      if (sign(l.v().y) < 0 and l.left(p) >= 0) continue;
      if (sign(l.v().y) == 0) continue;
      if (sign(l.v().y) > 0 and l.left(p) <= 0) continue;
      if (sign(a.y - p.y) < 0 and sign(b.y - p.y) >= 0) res += 1;
      if (sign(a.y - p.y) >= 0 and sign(b.y - p.y) < 0) res -= 1;
    }
    return res > 0;
  }
};
```

9.1.4 Convex Hull

7E4F655EB44CC8D45F35ED3B39C0A4A0

```
struct H : G {
  H(G g, bool raw) : G(g) { assert(raw); }
  H(G g) {
    ranges::sort(g, {}, [](P p) { return pair(p.x, p.y); });
    for (auto p : g) {
      while (size() >= 2 and (back() - end()[-2]).left(p - back()) !=
          1) pop_back();
      push_back(p);
    }
    auto n = size();
    for (auto p : g | views::reverse) {
      while (size() > n and (back() - end()[-2]).left(p - back()) != 1)
          pop_back();
      push_back(p);
    }
    pop_back();
  }
  L diameter() {
    L res(vertex(0), vertex(1));
    for (int i = 0, j = 1; i < (int)size(); i += 1) {
```

```
      while (edge(i).left(edge(j)) > 0) j += 1;
      L l(vertex(i), vertex(j));
      if (l.length2() > res.length2()) res = l;
    }
    return res;
  }
  bool inside(P p) {
    if (edge(0).left(p) != 1) return false;
    if (edge(-1).left(p) != 1) return false;
    int i = *ranges::partition_point(views::iota(1, (int)size()), [&](
        int i) { return (at(i) - at(0)).left(p - at(0)) != -1; });
    return edge(i - 1).left(p) == 1;
  }
  template <class F>
  int most(F f) {
    if (size() == 1) return 0;
    auto check = [&](int i) { return f(vertex(i)).left(edge(i).v()) <=
        0; };
    bool c0 = check(0);
    if (not c0 and check(-1)) return 0;
    auto f0 = f(at(0));
    return *ranges::partition_point(views::iota(1, (int)size()), [&](
        int i) {
      bool ci = check(i);
      int t = f0.left(at(i) - at(0));
      return ci ^ ((ci == c0) and ((not c0 and t >= 0) or (c0 and t >
          0)));
    });
  }
};
```

9.1.5 Intersection and Tangent

B378C0D2CA906243ED3939538F613BDB

```
bool cmp_argument(P& a, P& b) {
  auto pos = [&](P p) { return p.y < 0 ? -1 : p.y > 0 or (p.y == 0 and
      p.x < 0); };
  int pa = pos(a), pb = pos(b);
  return pa == pb ? a.left(b) == 1 : pa < pb;
}
pair<int, int> tangent(L l, H& h) {
  return pair(h.most([&](...) { return l.v(); }), h.most([&](...) {
      return -l.v(); }));
}
pair<int, int> tangent(P p, H& h) {
  return pair(h.most([&](P a) { return a - p; }), h.most([&](P a) {
      return p - a; }));
}
optional<pair<T, T>> instersection(L l, L m) {
  T den = m.v().cross(l.v()), num = m.v().cross(m.a - l.a);
  if (den == 0) return {};
  if (den < 0) {
    den = -den;
    num = -num;
  }
  return pair(num, den);
}
vector<pair<T, T>> instersection(L l, G g) {
  vector<pair<T, T>> res;
  for (int i = 0; i < ssize(g); i += 1) {
    L m = g.edge(i), n = g.edge(i - 1);
    auto x = instersection(m, l);
    if (not x) {
      auto y = instersection(n, l).value();
      if (y.first == y.second and m.left(n) == -1) res.push_back(
          instersection(l, n).value());
      continue;
    }
    auto y = x.value();
    if (y.first < 0 or y.first >= y.second) continue;
    if (y.first == 0) {
      if (l.left(n) == 0 and n.left(m) < 0) continue;
      if (l.left(m) * l.left(n) == -1) continue;
    }
    res.push_back(instersection(l, m).value());
  }
  ranges::sort(res, [](auto p0, auto p1) {
    auto [a0, b0] = p0;
    auto [a1, b1] = p1;
    return TT(a0) * b1 < TT(a1) * b0;
  });
  return res;
}
```

9.1.6 Convex Indenpent Increment

541F0DA0843B37250300D2AEE8466E7E

```
struct CH : H {
  vector<T> sum;
  map<pair<T, T>, int> mp;
  CH(H h) : H(h), sum(h.size() * 2) {
    for (int i = 0; i < ssize(sum); i += 1) sum[i] = (i ? sum[i - 1] :
        0) + h.vertex(i).cross(h.vertex(i + 1));
    for (int i = 0; i < (int)size(); i += 1) mp[pair(at(i).x, at(i).y)]
         = i;
  }
  T area2(vector<P> p) {
    G g;
    for (auto pi : p) {
      if (~inside(pi)) continue;
      auto [l, r] = tangent(pi, static_cast<H&>(*this));
      g.push_back(pi);
      g.push_back(at(l));
      g.push_back(at(r));
    }
    if (g.empty()) return sum[(int)size() - 1];
```

```
    H h(g);
    T res = 0;
    for (int i = 0; i < ssize(h); i += 1) {
      auto p = h[i], q = h.vertex(i + 1);
      auto pp = pair(p.x, p.y), pq = pair(q.x, q.y);
      int pi = mp.contains(pp) ? mp[pp] : -1, qi = mp.contains(pq) ? mp
          [pq] : -1;
      if (~pi and ~qi) {
        if (qi < pi) qi += size();
        res += sum[qi - 1] - (pi ? sum[pi - 1] : 0);
      } else
        res += p.cross(q);
    }
    return res;
  }
};
```

9.1.7 Half-plane Intersection

A879972DE53B2C9CB73922D383628172

```
vector<L> half_plane(vector<L> ls) {
  auto check = [](L a, L b, L c) {
    auto [x, y] = instersection(b, c).value();
    a = L(a.a * y, a.b * y);
    return a.left(b.a * y + b.v() * x) < 0;
  };
  ranges::sort(ls, [&](L lhs, L rhs) {
    if (lhs.v().left(rhs.v()) == 0 and sign(lhs.v().dot(rhs.v())) >= 0)
         return lhs.left(rhs.a) == -1;
    return cmp_argument(lhs.v(), rhs.v());
  });
  deque<L> q;
  for (int i = 0; i < ssize(ls); i += 1) {
    if (i and ls[i - 1].v().left(ls[i].v()) == 0 and sign(ls[i - 1].v()
        .dot(ls[i].v())) == 1) continue;
    while (ssize(q) > 1 and check(ls[i], q.back(), q.end()[-2])) q.
        pop_back();
    while (ssize(q) > 1 and check(ls[i], q[0], q[1])) q.pop_front();
    if (not q.empty() and q.back().v().left(ls[i].v()) <= 0) return {};
    q.push_back(ls[i]);
  }
  while (ssize(q) > 1 and check(q[0], q.back(), q.end()[-2])) q.
      pop_back();
  while (ssize(q) > 1 and check(q.back(), q[0], q[1])) q.pop_front();
  return vector(q.begin(), q.end());
}
```

9.2 3D Geometry

619303CF4D8E6B96F9AD45147445FBAF

```
using T = f64;
struct P {
  T x, y, z;
  explicit P(T x = 0, T y = 0, T z = 0) : x(x), y(y), z(z) {}
  T dot(P p) { return x * p.x + y * p.y + z * p.z; }
  P cross(P p) { return P(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y
      - y * p.x); }
  P operator-(P p) { return P(x - p.x, y - p.y, z - p.z); }
  T length() { return sqrt(x * x + y * y + z * z); }
  friend ostream& operator<<(ostream& os, P p) { return os << "(" << p.
      x << ",␣" << p.y << ",␣" << p.z << ")"; }
};
vector<tuple<int, int, int>> hull(vector<P> p, T eps = 1e-9) {
  mt19937_64 mt(random_device{}());
  uniform_real_distribution<f64> urd(0, eps);
  for (auto& [x, y, z] : p) {
    x += urd(mt);
    y += urd(mt);
    z += urd(mt);
  }
  vector<tuple<int, int, int>> res;
  res.emplace_back(0, 1, 2);
  res.emplace_back(0, 2, 1);
  for (int i = 3; i < ssize(p); i += 1) {
    vector<tuple<int, int, int>> nxt;
    set<pair<int, int>> edge;
    for (auto [a, b, c] : res) {
      T prod = (p[b] - p[a]).cross(p[c] - p[a]).dot(p[i] - p[a]);
      if (prod > 0) {
        edge.emplace(a, b);
        edge.emplace(b, c);
        edge.emplace(c, a);
      } else
        nxt.emplace_back(a, b, c);
    }
    for (auto [x, y] : edge) {
      if (edge.contains({y, x})) continue;
      nxt.emplace_back(x, y, i);
    }
    res.swap(nxt);
  }
  return res;
}
f64 volume(vector<P> p, vector<tuple<int, int, int>> f) {
  f64 res = 0;
  for (auto [a, b, c] : f) res += p[a].cross(p[b]).dot(p[c]);
  return res / 6;
}
f64 area(vector<P> p, vector<tuple<int, int, int>> f) {
  f64 res = 0;
  for (auto [a, b, c] : f) res += (p[b] - p[a]).cross(p[c] - p[b]).
      length();
  return res / 2;
}
```

10 Game

10.1 Nim Product

6F6A7EBCD62BE34A43F852BA4AF6963D

```
struct NimProduct {
  array<array<u64, 64>, 64> mem;
  NimProduct() {
    for (int i = 0; i < 64; i += 1)
      for (int j = 0; j < 64; j += 1) {
        int k = i & j;
        if (k == 0)
          mem[i][j] = u64(1) << (i | j);
        else {
          int x = k & -k;
          mem[i][j] = mem[i ^ x][j] ^ mem[(i ^ x) | (x - 1)][(j ^ x) |
              (i & (x - 1))];
        }
      }
  }
  u64 nim_product(u64 x, u64 y) {
    u64 res = 0;
    for (int i = 0; i < 64 and x >> i; i += 1)
      if ((x >> i) % 2)
        for (int j = 0; j < 64 and y >> j; j += 1)
          if ((y >> j) % 2) res ^= mem[i][j];
    return res;
  }
};
```