Team Reference Document

Heltion

March 16, 2024

Contents

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

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 Heavy Light Decomposition (Segment Tree)

507B40C9065E1D81C9562D053BA5EE42

```
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);
  }
};
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;
  }
};
```

2.3 Li Chao Tree (Get Minimum)

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;
  }
};
```

2.4 Dynamic Lines (Get Minimum)

E1E794A2503BF883D1F8EEBDD894212A

```
struct Line {
  mutable i64 k, b, p;
  bool operator<(const Line& rhs) const { return k < rhs.k; }
  bool operator<(const i64& x) const { return p < x; }
};
struct Lines : multiset<Line, less<>> {
  static constexpr i64 inf = numeric_limits<i64>::max();
  static i64 div(i64 a, i64 b) { return a / b - ((a ^ b) < 0 and a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, false;
    if (x->k == y->k)
      x->p = x->b > y->b ? inf : -inf;
    else
      x->p = div(y->b - x->b, x->k - y->k);
    return x->p >= y->p;
  }
  void add(i64 k, i64 b) {
    auto z = insert({k, b, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() and isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() and (--x)->p >= y->p) isect(x, erase(y));
  }
  optional<i64> get(i64 x) {
    if (empty()) return {};
    auto it = lower_bound(x);
    return it->k * x + it->b;
  }
};
```


3 Graph

3.1 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;
}
```

3.2 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;
}
```

3.3 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;
}
```

3.4 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;
}
```

3.5 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;
}
```

3.6 K Shortest Paths (Persistent Leftist Heap)

139DD0E4BEC848E1A8F5501244E6E484

```
template <typename T>
using MinHeap = priority_queue<T, vector<T>, greater<>>;
tuple<vector<int>, vector<int>, vector<i64>> shortest_tree(const vector
    <vector<pair<int, i64>>>& adj, int s) {
  int n = adj.size();
  MinHeap<pair<i64, int>> pq;
  vector<int> p(n, -1), order;
  vector<i64> d(n, -1);
  pq.emplace(d[s] = 0, s);
  while (not pq.empty()) {
    auto [du, u] = pq.top();
    pq.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;
        pq.emplace(d[v] = d[u] + w, v);
      }
    }
  }
  return {p, order, d};
}
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; }
};
vector<i64> k_shortest_paths(const vector<vector<pair<int, i64>>>& adj,
    int s, int t, int k) {
  int n = adj.size();
  auto [p, order, d] = shortest_tree(adj, s);
  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;
}
```

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

989CF1A97488AD4B81D7355DDEF78A14

```
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;
    }
  }
};
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);
  }
};
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};
}
```

3.8 Dominator Tree

A944605F16E354D8E9429D8425FC33FC

```
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;
}
```

3.9 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));
}
```

3.10 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;
  }
};
```

3.11 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;
  }
};
```

3.12 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};
}
```

3.13 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};
  }
};
```

4 String

4.1 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;
}
```

4.2 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;
}
```

4.3 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;
}
```

4.4 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;
}
```

4.5 Aho-Corasick

8B5C8AEB6B2D4217BE99B61721255D12

```
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);
      }
    }
  }
};
```

4.6 Palindrome Tree

7B6E73D28CB8226EAFBEC56EBD2AB8CF

```
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];
  }
};
```

4.7 Suffix Automaton

59E725E9066C3D4CE4DC238624B8C837

```
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;
  }
};
```

5 Math

5.1 Multiplication of Integers (Fast Fourier Transform)

E8D9F1845BDA3D73F34D087A868D636F

```
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;
}
string operator*(const string& a, const string& b) {
  vector<complex<f64>> af, bf;
  for (char c : a)
    if (c != '-') af.emplace_back(c - '0', 0);
  for (char c : b)
    if (c != '-') bf.emplace_back(c - '0', 0);
  ranges::reverse(af);
  ranges::reverse(bf);
  int n = bit_ceil(af.size() + bf.size());
  af.resize(n);
```

```
  bf.resize(n);
  fft(af);
  fft(bf);
  for (int i = 0; i < n; i += 1) af[i] = af[i] * bf[i];
  fft(af, true);
  vector<int> c;
  for (auto x : af) c.push_back(int(x.real() + .5));
  for (int i = 0; i < ssize(c); i += 1) {
    if (c[i] < 10) continue;
    if (i + 1 == ssize(c)) c.push_back(0);
    c[i + 1] += c[i] / 10;
    c[i] %= 10;
  }
  while (not c.empty() and c.back() == 0) c.pop_back();
  if (c.empty()) return "0";
  string s;
  for (int ci : c) s += '0' + ci;
  if ((a[0] == '-') != (b[0] == '-')) s += '-';
  ranges::reverse(s);
  return s;
}
```

5.2 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))); }
};
```

5.3 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;
}
```

5.4 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 {};
}
```

5.5 Factorize (Pollard Rho and Miller Rabin)

F2AE585A3C0CF873D512CF276EDF216B

```
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;
};
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;
}
```

5.6 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}};
}
```

5.7 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;
}
```

5.8 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);
}
```

5.9 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;
  }
};
```

5.10 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);
}
```

5.11 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);
}
```

5.12 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;
    }
};
```

6 Game

7 Geometry

DF1340EFEF813346D4E89FC14534E551

```
template <typename T>
T eps = 0;
template <>
f64 eps<f64> = 1e-9;
template <typename T>
int sign(T x) {
  return x < -eps<T> ? -1 : x > eps<T>;
}
template <typename T>
struct P {
  T x, y;
  explicit P(T x = 0, T y = 0) : x(x), y(y) {}
  P operator*(T k) { return P(x * k, y * k); }
  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); }
  T len2() { return x * x + y * y; }
  T cross(P p) { return x * p.y - y * p.x; }
  T dot(P p) { return x * p.x + y * p.y; }
  bool operator==(P p) { return sign(x - p.x) == 0 and sign(y - p.y) ==
      0; }
  int arg() { return y < 0 or (y == 0 and x > 0) ? -1 : x or y; }
  P rotate90() { return P(-y, x); }
};
template <typename T>
bool argument(P<T> lhs, P<T> rhs) {
  if (lhs.arg() != rhs.arg()) return lhs.arg() < rhs.arg();
  return lhs.cross(rhs) > 0;
}
template <typename T>
struct L {
  P<T> a, b;
  explicit L(P<T> a = {}, P<T> b = {}) : a(a), b(b) {}
  P<T> v() { return b - a; }
  bool contains(P<T> p) { return sign((p - a).cross(p - b)) == 0 and
      sign((p - a).dot(p - b)) <= 0; }
  int left(P<T> p) { return sign(v().cross(p - a)); }
  optional<pair<T, T>> intersection(L l) {
    auto y = v().cross(l.v());
    if (sign(y) == 0) return {};
    auto x = (l.a - a).cross(l.v());
    return y < 0 ? pair(-x, -y) : pair(x, y);
  }
};
template <typename T>
struct G {
  vector<P<T>> g;
  explicit G(int n) : g(n) {}
```

```
47    explicit G(const vector<P<T>>& g) : g(g) {}
48    optional<int> winding(P<T> p) {
49      int n = g.size(), res = 0;
50      for (int i = 0; i < n; i += 1) {
51        auto a = g[i], b = g[(i + 1) % n];
52        L l(a, b);
53        if (l.contains(p)) return {};
54        if (sign(l.v().y) < 0 and l.left(p) >= 0) continue;
55        if (sign(l.v().y) == 0) continue;
56        if (sign(l.v().y) > 0 and l.left(p) <= 0) continue;
57        if (sign(a.y - p.y) < 0 and sign(b.y - p.y) >= 0) res += 1;
58        if (sign(a.y - p.y) >= 0 and sign(b.y - p.y) < 0) res -= 1;
59      }
60      return res;
61    }
62    G convex() {
63      ranges::sort(g, {}, [&](P<T> p) { return pair(p.x, p.y); });
64      vector<P<T>> h;
65      for (auto p : g) {
66        while (ssize(h) >= 2 and sign((h.back() - h.end()[-2]).cross(p -
              h.back())) <= 0) h.pop_back();
67        h.push_back(p);
68      }
69      int m = h.size();
70      for (auto p : g | views::reverse) {
71        while (ssize(h) > m and sign((h.back() - h.end()[-2]).cross(p - h
              .back())) <= 0) h.pop_back();
72        h.push_back(p);
73      }
74      h.pop_back();
75      return G(h);
76    }
77    // Following function are valid only for convex.
78    T diameter2() {
79      int n = g.size();
80      T res = 0;
81      for (int i = 0, j = 1; i < n; i += 1) {
82        auto a = g[i], b = g[(i + 1) % n];
83        while (sign((b - a).cross(g[(j + 1) % n] - g[j])) > 0) j = (j +
              1) % n;
84        res = max(res, (a - g[j]).len2());
85        res = max(res, (a - g[j]).len2());
86      }
87      return res;
88    }
89    optional<bool> contains(P<T> p) {
90      if (g[0] == p) return {};
91      if (g.size() == 1) return false;
92      if (L(g[0], g[1]).contains(p)) return {};
93      if (L(g[0], g[1]).left(p) <= 0) return false;
94      if (L(g[0], g.back()).left(p) > 0) return false;
95      int i = *ranges::partition_point(views::iota(2, ssize(g)), [&](int
              i) { return sign((p - g[0]).cross(g[i] - g[0])) <= 0; });
96      int s = L(g[i - 1], g[i]).left(p);
97      if (s == 0) return {};
98      return s > 0;
99    }
100   int most(const function<P<T>(P<T>)>& f) {
101     int n = g.size();
102     auto check = [&](int i) { return sign(f(g[i]).cross(g[(i + 1) % n]
            - g[i])) >= 0; };
103     P<T> f0 = f(g[0]);
104     bool check0 = check(0);
105     if (not check0 and check(n - 1)) return 0;
106     return *ranges::partition_point(views::iota(0, n), [&](int i) ->
            bool {
107       if (i == 0) return true;
108       bool checki = check(i);
109       int t = sign(f0.cross(g[i] - g[0]));
110       if (i == 1 and checki == check0 and t == 0) return true;
111       return checki ^ (checki == check0 and t <= 0);
112     });
113   }
114   pair<int, int> tan(P<T> p) {
115     return {most([&](P<T> x) { return x - p; }), most([&](P<T> x) {
            return p - x; })};
116   }
117   pair<int, int> tan(L<T> l) {
118     return {most([&](P<T> _) { return l.v(); }), most([&](P<T> _) {
            return -l.v(); })};
119   }
120 };
121
122 template <typename T>
123 vector<L<T>> half(vector<L<T>> ls, T bound) {
124   // Ranges: bound ^ 6
125   auto check = [](L<T> a, L<T> b, L<T> c) {
126     auto [x, y] = b.intersection(c).value();
127     a = L(a.a * y, a.b * y);
128     return a.left(b.a * y + b.v() * x) < 0;
129   };
130   ls.emplace_back(P(-bound, (T)0), P(-bound, -(T)1));
131   ls.emplace_back(P((T)0, -bound), P((T)1, -bound));
132   ls.emplace_back(P(bound, (T)0), P(bound, (T)1));
133   ls.emplace_back(P((T)0, bound), P(-(T)1, bound));
134   ranges::sort(ls, [&](L<T> lhs, L<T> rhs) {
135     if (sign(lhs.v().cross(rhs.v())) == 0 and sign(lhs.v().dot(rhs.v())
          ) >= 0) return lhs.left(rhs.a) == -1;
136     return argument(lhs.v(), rhs.v());
137   });
138   deque<L<T>> q;
139   for (int i = 0; i < ssize(ls); i += 1) {
140     if (i and sign(ls[i - 1].v().cross(ls[i].v())) == 0 and sign(ls[i -
           1].v().dot(ls[i].v())) == 1) continue;
141     while (q.size() > 1 and check(ls[i], q.back(), q.end()[-2])) q.
          pop_back();
142     while (q.size() > 1 and check(ls[i], q[0], q[1])) q.pop_front();
143     if (not q.empty() and sign(q.back().v().cross(ls[i].v())) <= 0)
```

```
        return {};
144   q.push_back(ls[i]);
145  }
146  while (q.size() > 1 and check(q[0], q.back(), q.end()[-2])) q.
```

```
        pop_back();
147  while (q.size() > 1 and check(q.back(), q[0], q[1])) q.pop_front();
148  return vector<L<T>>(q.begin(), q.end());
149 }
```