Sumário

1 Geral

1.1 128.cpp

```cpp
ostream& operator<<(ostream& out, __int128 x) {
  if (x == 0) return out << 0;
  string s;
  bool sig = x < 0;
  x = x < 0 ? -x : x;
  while (x > 0) s += x % 10 + '0', x /= 10;
  if (sig) s += '-';
  reverse(all(s));
  return out << s;
}

istream& operator>>(istream& in, __int128& x) {
  char c, neg = 0;
  while (isspace(c = in.get()))
    ;
  if (!isdigit(c))
    neg = (c == '-'), x = 0;
  else
    x = c - '0';
  while (isdigit(c = in.get())) x = (x << 3) + (x << 1) - '0' + c;
  x = neg ? -x : x;
  return in;
}
```

1.2 aho-corasick.cpp

```cpp
#define to_i(ch) (ch - 'a');  // TODO
const int K = 26;             // TODO
struct node {
  int term = 0, p, pc, link = -1, exi = -1, occ = 0;
  vi nxt, go, ids;
  node(int _p = 0, int _pc = 0) : p(_p), pc(_pc), nxt(K, -1), go(K, -1) {}
};

vector<node> aca;  // TODO: criar o no raiz
int occ[MAX];
int go(int u, int c);

int link(int u) {
  if (aca[u].link != -1) return aca[u].link;
  return aca[u].link = !aca[u].p ? 0 : go(link(aca[u].p), aca[u].pc);
}

int go(int u, int c) {
  if (aca[u].go[c] != -1) return aca[u].go[c];
  if (aca[u].nxt[c] != -1) return aca[u].go[c] = aca[u].nxt[c];
  return aca[u].go[c] = !u ? 0 : go(link(u), c);
}

int exi(int u) {
  if (aca[u].exi != -1) return aca[u].exi;
  int v = link(u);
```

```
  return aca[u].exi = (!v || aca[v].term) ? v : exi(v);
}

void process(string word) {
  for (int i = 0, u = 0; i < word.size(); i++) {
    int c = to_i(word[i]);
    u = go(u, c);
    for (int v = u; v; v = exi(v)) {
      aca[v].occ++;
    }
  }
  for (auto &v : aca) {
    for (auto &i : v.ids) {
      occ[i] += v.occ;
    }
  }
}

void ins(string word, int id) {
  int u = 0;
  for (int i = 0; i < word.size(); i++) {
    int c = to_i(word[i]);
    if (aca[u].nxt[c] == -1) {
      aca[u].nxt[c] = aca.size();
      aca.emplace_back(u, c);
    }
    u = aca[u].nxt[c];
  }
  aca[u].term = 1;
  aca[u].ids.push_back(id);
}
```

1.3 all-divisors.cpp

```
vi find_all_divisors(int n) {
  vi x;
  for (int i = 1; i * i <= n; i++) {
    if (n % i == 0) {
      if (n / i == i) {
        x.push_back(i);
      } else {
        x.push_back(i);
        x.push_back(n / i);
      }
    }
  }
  x.push_back(n);
  return x;
}
```

1.4 articbridges.cpp

```
int tk = 0;
vi tin(MAX), low(MAX);
vector<ii> brid;
set<int> arti;
```

```
void dfs(int u, int p) {
  tin[u] = low[u] = tk++;
  int ch = 0;
  for (auto v : g[u]) {
    if (v == p) continue;
    if (tin[v] == -1) {
      dfs(v, u);
      ch++;
      if ((low[v] >= tin[u] && p != u) || (ch >= 2 && p == u)) arti.insert(u);
      if (low[v] > tin[u]) brid.push_back(ii(u, v));
      low[u] = min(low[u], low[v]);
    } else {
      low[u] = min(low[u], tin[v]);
    }
  }
}

void articbridges(int n) {
  fill(all(tin), -1);
  tk = 0;
  arti.clear();
  brid.clear();
  for (int i = 0; i < n; i++) {
    dfs(i, i);  // TODO?
  }
}
```

1.5 binomio.cpp

```
void build() {
  binom[0][0] = 1;
  for (int i = 1; i < MAX; i++) {
    binom[i][i] = binom[i][0] = 1;
    for (int j = 1; j < i; j++) {
      binom[i][j] = binom[i - 1][j - 1] + binom[i - 1][j];
    }
  }
}
```

1.6 bipartido.cpp

```
#define MAX 3 * (212345)  // TODO

// TODO: meu codigo do dsu

int offset = 212345;
int bipartido = 1;

void special_join(int u, int v) {
  join(u, v + offset);
  join(u + offset, v);
  if (__find(u) == __find(u + offset) || __find(v) == __find(v + offset)) {
    bipartido = 0;
  }
}
```

1.7 bit2d.cpp

```
ll bit[MAX][MAX];

void add(int i, int j, ll v) {
  for (; i < MAX; i += i & (-i))
    for (int jj = j; jj < MAX; jj += jj & (-jj)) bit[i][jj] += v;
}

ll getbit(int i, int j) {
  ll sum = 0;
  for (; i; i -= i & (-i))
    for (int jj = j; jj; jj -= jj & (-jj)) sum += bit[i][jj];
  return sum;
}

ll getbit(int lx, int ly, int rx, int ry) {  // getbit(1, 1, lin, col)
  return getbit(rx, ry) - getbit(rx, ly - 1) - getbit(lx - 1, ry) +
         getbit(lx - 1, ly - 1);
}

void add(int lx, int ly, int rx, int ry, ll v) {  // canto superior esquerdo
  rx++;                                           // canto inferior direito
  ry++;
  add(lx, ly, +v);
  add(lx, ry, -v);
  add(rx, ly, -v);
  add(rx, ry, +v);
}
```

1.8 bit2d-esparsa.cpp

```
ordered_set<ii> bit[MAX];

void insert(int x, int y) {
  for (int i = x; i < MAX; i += i & -i) bit[i].insert(ii(y, x));
}

void remove(int x, int y) {
  for (int i = x; i < MAX; i += i & -i) bit[i].erase(ii(y, x));
}

int get(int x, int y) {
  int ans = 0;
  for (int i = x; i > 0; i -= i & -i) ans += bit[i].order_of_key(ii(y + 1, 0))
      ;
  return ans;
}

int get(int lx, int ly, int rx, int ry) {
  return get(rx, ry) - get(rx, ly - 1) - get(lx - 1, ry) + get(lx - 1, ly - 1)
      ;
}
```

1.9 bit.cpp

```cpp
vi bit(MAX);

void addbit(int i, int delta) {
  for (; i < MAX; i += i & (-i)) bit[i] += delta;
}

int getbit(int i) {
  int ans = 0;
  for (; i > 0; i -= i & (-i)) ans += bit[i];
  return ans;
}

int getbit(int l, int r) { return getbit(r) - getbit(l - 1); }
```

1.10 block-cut-tree.cpp

```cpp
int in[MAX], low[MAX], id[MAX], tk;  // block-cut-tree
bool art[MAX];
vi adj[MAX], stk;
vector<vi> children, blocks;

void dfs_blk(int u, int p) {
  in[u] = low[u] = ++tk;
  stk.push_back(u);
  for (auto &v : adj[u]) {
    if (v == p) continue;
    if (in[v]) {  // back
      low[u] = min(low[u], in[v]);
    } else {  // fwd
      dfs_blk(v, u);
      low[u] = min(low[u], low[v]);

      if (low[v] >= in[u]) {
        art[u] = (in[u] > 1 || in[v] > 2);
        blocks.push_back({u});
        while (blocks.back().back() != v) {
          // TODO: entender porque não é a outra condicao
          blocks.back().push_back(stk.back());
          stk.pop_back();
        }
      }
    }
  }
}

void blockcut(int n) {
  for (int i = 0; i < n; i++)
    if (!in[i]) dfs_blk(i, i);

  children.resize(blocks.size());
  for (int i = 0; i < n; i++)
    if (art[i]) {
      id[i] = children.size();
      children.emplace_back();
    }
  for (int i = 0; i < blocks.size(); i++) {
```

```
    for (auto &u : blocks[i]) {
      if (!art[u]) {
        id[u] = i;
      } else {
        children[id[u]].emplace_back(i);
        children[i].emplace_back(id[u]);
      }
    }
  }
}
```

1.11 centroid.cpp

```
vi adj[MAX];
int sz[MAX];
bool mark[MAX];

int dfs_sz(int u, int p = -1) {
  sz[u] = 1;
  for (auto &v : adj[u])
    if (v != p && !mark[v]) sz[u] += dfs_sz(v, u);
  return sz[u];
}

int dfs_cent(int u, int n, int p = -1) {
  for (auto &v : adj[u])
    if (v != p && !mark[v] && sz[v] >= (n >> 1)) return dfs_cent(v, n, u);
  return u;
}

void centroid(int u = 0) {
  int c = dfs_cent(u, dfs_sz(u));
  mark[c] = 1;

  // TODO

  for (auto &v : adj[c])
    if (!mark[v]) centroid(v);
}
```

1.12 color-update.cpp

```
struct ColorUpdate {
  struct Node {
    int l, r;
    mutable int v;

    Node(const int &_l, const int &_r, const int &_v) : l(_l), r(_r), v(_v) {}

    bool operator<(const Node &o) const { return l < o.l; }
  };

  int n;
  set<Node> s;

  ColorUpdate(int _n) : n(_n) {
```

```
    s.emplace(1, n, 1);  // TODO:
  }

  auto split(int x) {
    if (x > n) return s.end();
    auto it = --s.upper_bound(Node{x, 0, 0});
    if (it->l == x) return it;
    int l = it->l, r = it->r, v = it->v;
    s.erase(it);
    s.emplace(l, x - 1, v);
    return s.emplace(x, r, v).first;
  }

  void assign(int l, int r, int v) {
    auto itr = split(r + 1), itl = split(l);
    s.erase(itl, itr);
    s.emplace(l, r, v);
  }

  int getv(int x) {
    auto it = --s.upper_bound(Node{x, 0, 0});
    return it->v;
  }

  void apply(int l, int r) {  // TODO:
    auto itr = split(r + 1), itl = split(l);
    for (; itl != itr; ++itl) {
      auto [a, b, c] = *itl;
    }
  }
};
```

1.13 combinatoria.cpp

```

void build() {
  fat[0] = 1;
  for (int i = 1; i < MAX; i++) {
    fat[i] = (fat[i - 1] * i) % P;
  }

  invfat[MAX - 1] = expbin(fat[MAX - 1], P - 2);
  for (int i = MAX - 1; i >= 1; i--) {
    invfat[i - 1] = (invfat[i] * (i)) % P;
  }
}
```

1.14 composicao.cpp

```
int weak(int n, int k) {
  return c(n + k - 1, k - 1);  // binomio
}

int strong(int n, int k) { return c(n - 1, k - 1); }
```

1.15 compress.cpp

```cpp
vector<ii> tmp;

void compress(vi &vet) {
  int n = vet.size();
  tmp.resize(n);
  for (int i = 0; i < n; i++) {
    tmp[i].first = vet[i];
    tmp[i].second = i;
  }
  sort(all(tmp));
  int cnt = 0;
  for (int i = 0; i < n; i++) {
    if (i > 0 && tmp[i].first != tmp[i - 1].first) cnt++;
    vet[tmp[i].second] = cnt;
  }
}
```

1.16 conectividade-dinamica.cpp

```cpp
const int MAX = 212345;
int tamseg = 0;  // TODO
vi queries(MAX, -1);
vector<ii> seg[4 * MAX];  // TODO: tamanho mais preciso?

dsu_rollback dsu(MAX);  // TODO

void add(iiii &val, int pos, int lx, int rx) {
  auto &[l, r, u, v] = val;
  if (lx >= r || rx <= l) return;
  if (lx >= l && rx <= r) {
    seg[pos].emplace_back(u, v);
    return;
  }
  int mid = lx + (rx - lx) / 2;
  add(val, 2 * pos + 1, lx, mid);
  add(val, 2 * pos + 2, mid, rx);
}

void solve(int pos, int lx, int rx) {
  int antes = dsu.checkpoint();
  for (auto &[u, v] : seg[pos]) dsu.join(u, v);
  if (rx - lx == 1) {
    if (queries[lx] != -1) {
      // TODO: resposta no tempo lx
    }
    dsu.undo(antes);
    return;
  }
  int mid = lx + (rx - lx) / 2;
  solve(2 * pos + 1, lx, mid);
  solve(2 * pos + 2, mid, rx);
  dsu.undo(antes);
}

vector<iiii> lifetime;  // TODO: pode ser removido em caso de MLE
map<ii, int> edges;
```

```

void addEdge(int u, int v, int timer) {
  if (u > v) swap(u, v);
  edges[ii(u, v)] = timer;
}

void remEdge(int u, int v, int timer) {  // assume que (u, v) existe
  if (u > v) swap(u, v);
  int l = edges[ii(u, v)], r = timer;
  lifetime.emplace_back(l, r, u, v);
  edges.erase(ii(u, v));
}

void doAll(int timer) {
  for (auto &[uv, l] : edges) {
    auto [u, v] = uv;
    if (u > v) swap(u, v);
    int r = timer;
    lifetime.emplace_back(l, r, u, v);
  }
  for (auto &val : lifetime) add(val, 0, 0, tamseg);
  solve(0, 0, tamseg);
}

void zerar() {
  int sz = 1;
  while (sz < tamseg) sz *= 2;
  sz *= 2;
  for (int i = 0; i < sz; i++) seg[i].clear();
  for (int i = 0; i < tamseg; i++) queries[i] = -1;
  edges.clear();
  lifetime.clear();
}
```

1.17 convex-hull.cpp

```

const double EPS = 1e-9;
using pt = complex<double>;

#define px real()
#define py imag()

struct cmp {
  bool operator()(const pt &a, const pt &b) const {
    return a.px < b.px || (a.px == b.px && a.py < b.py);
  }
};

double dot(pt a, pt b) { return (conj(a) * b).px; }
double cross(pt a, pt b) { return (conj(a) * b).py; }
pt vec(pt a, pt b) { return b - a; }
int sgn(double v) { return (v > -EPS) - (v < EPS); }
// -1 (cw), 0 (colinear), +1 (ccw)
int seg_ornt(pt a, pt b, pt c) { return sgn(cross(vec(a, b), vec(a, c))); }
int ccw(pt a, pt b, pt c, bool col) {
  int o = seg_ornt(a, b, c);
```

```
  return (o == 1) || (o == 0 && col);
}
const double PI = acos(-1);       // opcional
double angle(pt a, pt b, pt c) {  // opcional
  return abs(remainder(arg(a - b) - arg(c - b), 2.0 * PI));
}

double dist(pt a, pt b) {  // opcional
  double dx = a.px - b.px;
  double dy = a.py - b.py;
  return sqrt(dx * dx + dy * dy);
}

// O(n lg n)
vector<pt> convex_hull(vector<pt> &ps, bool col = false) {
  int k = 0, n = ps.size();
  vector<pt> ans(2 * n);
  sort(all(ps), [](pt a, pt b) {
    return make_pair(a.px, a.py) < make_pair(b.px, b.py);
  });
  for (int i = 0; i < n; i++) {
    /* lower hull */
    while (k >= 2 && !ccw(ans[k - 2], ans[k - 1], ps[i], col)) {
      k--;
    }
    ans[k++] = ps[i];
  }
  if (k == n) {
    ans.resize(n);
    return ans;
  }
  for (int i = n - 2, t = k + 1; i >= 0; i--) {
    /* upper hull */
    while (k >= t && !ccw(ans[k - 2], ans[k - 1], ps[i], col)) {
      k--;
    }
    ans[k++] = ps[i];
  }
  ans.resize(k - 1);
  return ans;
}

vector<pt> convex_hull(vector<ii> &pontos) {
  set<pt, cmp> s_ps;
  for (auto &[x, y] : pontos) {
    s_ps.insert(pt(x, y));
  }
  vector<pt> ps;
  for (auto &x : s_ps) {
    ps.push_back(x);
  }
  vector<pt> hull = convex_hull(ps, true);
  return hull;
}
```

1.18 convex-hull-trick.cpp

```cpp
struct cht_line {
  int a, b, x;
  cht_line(int _a, int _b, int _x) : a(_a), b(_b), x(_x) {}
  bool operator<(int oth) { return x < oth; }
};

deque<cht_line> cht;

int inter(cht_line x, cht_line y) {
  assert(x.a != y.a);
  if (((x.b - y.b) > 0 && (y.a - x.a) < 0) ||
      ((x.b - y.b) < 0 && (y.a - x.a) > 0)) {
    return -((abs(x.b - y.b) + (y.a - x.a - 1)) / abs(y.a - x.a));
  }
  return (x.b - y.b) / (y.a - x.a);
}

void add_line_back(int a, int b) {  // coefience angular maior
  if (!cht.empty() && cht.back().a == a) cht.pop_back();
  while (cht.size() >= 2) {
    int x = inter(cht[cht.size() - 1], {a, b, 0});
    int y = inter(cht[cht.size() - 2], cht[cht.size() - 1]);
    if (x < y)
      cht.pop_back();
    else
      break;
  }
  if (!cht.empty()) {
    cht.back().x = inter(cht.back(), {a, b, 0});
  }
  cht.emplace_back(a, b, oo);
}

void add_line_front(int a, int b) {  // coeficiente angular menor
  if (!cht.empty() && cht[0].a == a) return;
  while (cht.size() >= 2) {
    int x = inter({a, b, 0}, cht[0]);
    int y = inter(cht[0], cht[1]);
    if (x > y)
      cht.pop_front();
    else
      break;
  }
  cht.push_front({a, b, oo});
  if (cht.size() > 1) {
    cht[0].x = inter(cht[0], cht[1]);
  }
}

int query(int x) {  // maximo
  int pos = lower_bound(all(cht), x) - cht.begin();
  return cht[pos].a * x + cht[pos].b;
}
```

1.19 dijkstra.cpp

```cpp
void dijkstra(int s, int t) {
  priority_queue<ii, vector<ii>, greater<ii>> q;
  fill(dist, dist + MAX, oo);

  dist[s] = 0;
  q.emplace(dist[s], s);
  while (!q.empty()) {
    ll cost, u;
    tie(cost, u) = q.top();
    q.pop();
    if (dist[u] < cost) continue;
    if (u == t) break;
    for (auto& [v, c] : adj[u]) {
      if (dist[v] > dist[u] + c) {
        dist[v] = dist[u] + c;
        q.emplace(dist[v], v);
      }
    }
  }
}
```

1.20 dinic.cpp

```cpp
// O(min(m * max_flow, n^2 m)) - from: Bruno Monteiro
// Grafo com capacidades 1: O(min(m sqrt(m), m * n^(2/3)))
// Todo vertice tem grau de entrada ou saida 1: O(m sqrt(n))

struct dinitz {
  const bool scaling = false;  // com scaling -> O(nm log(MAXCAP)),
  int lim;                     // com constante alta
  struct edge {
    int to, cap, rev, flow;
    bool res;
    edge(int to_, int cap_, int rev_, bool res_)
        : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
  };

  vector<vector<edge>> g;
  vector<int> lev, beg;
  ll F;
  dinitz(int n) : g(n), F(0) {}

  void add(int a, int b, int c) {
    g[a].emplace_back(b, c, g[b].size(), false);
    g[b].emplace_back(a, 0, g[a].size() - 1, true);
  }
  bool bfs(int s, int t) {
    lev = vector<int>(g.size(), -1);
    lev[s] = 0;
    beg = vector<int>(g.size(), 0);
    queue<int> q;
    q.push(s);
    while (q.size()) {
      int u = q.front();
```

```
      q.pop();
      for (auto& i : g[u]) {
        if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
        if (scaling and i.cap - i.flow < lim) continue;
        lev[i.to] = lev[u] + 1;
        q.push(i.to);
      }
    }
    return lev[t] != -1;
  }
  int dfs(int v, int s, int f = INF) {
    if (!f or v == s) return f;
    for (int& i = beg[v]; i < g[v].size(); i++) {
      auto& e = g[v][i];
      if (lev[e.to] != lev[v] + 1) continue;
      int foi = dfs(e.to, s, min(f, e.cap - e.flow));
      if (!foi) continue;
      e.flow += foi, g[e.to][e.rev].flow -= foi;
      return foi;
    }
    return 0;
  }
  ll max_flow(int s, int t) {
    for (lim = scaling ? (1 << 30) : 1; lim; lim /= 2)
      while (bfs(s, t))
        while (int ff = dfs(s, t)) F += ff;
    return F;
  }
};

// Recupera as arestas do corte s-t
vector<pair<int, int>> get_cut(dinitz& g, int s, int t) {
  g.max_flow(s, t);
  vector<pair<int, int>> cut;
  vector<int> vis(g.g.size(), 0), st = {s};
  vis[s] = 1;
  while (st.size()) {
    int u = st.back();
    st.pop_back();
    for (auto e : g.g[u])
      if (!vis[e.to] and e.flow < e.cap) vis[e.to] = 1, st.push_back(e.to);
  }
  for (int i = 0; i < g.g.size(); i++)
    for (auto e : g.g[i])
      if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e.to);
  return cut;
}
```

1.21 disjoint-sparse-table.cpp

```
int tab[LOGN][2 * MAXN];

int op(int a, int b) { return min(a, b); }

void build(vi &vet) {
  int N = vet.size();
```

```
  int n = 1;
  while (n < N) n *= 2;
  for (int i = 0; (1 << i) < n; i++) {
    int sz = 1 << i;
    for (int m = 0; m < n; m += 2 * sz) {
      tab[i][m + sz - 1] = vet[m + sz - 1];
      for (int j = 1; j < sz; j++) {
        tab[i][m + sz - 1 - j] =
            op(tab[i][m + sz - 1 - j + 1], vet[m + sz - j - 1]);
      }
    }
    for (int m = sz; m < n; m += 2 * sz) {
      tab[i][m] = vet[m];
      for (int j = 1; j < sz; j++) {
        tab[i][m + j] = op(tab[i][m + j - 1], vet[m + j]);
      }
    }
  }
}

int get(int l, int r) {  // [l, r]
  if (l == r) return tab[0][l];
  int h = __builtin_clz(1) - __builtin_clz(l ^ r);
  return op(tab[h][l], tab[h][r]);
}
```

1.22 double-hash.cpp

```
const int P = 1e9 + 7;
mt19937 rmg((int)chrono::steady_clock::now().time_since_epoch().count());
int b1 = uniform_int_distribution<int>(1, P - 1)(rmg);
int b2 = uniform_int_distribution<int>(1, P - 1)(rmg);

struct hash_str {
  vector<ll> h1, p1, h2, p2;
  hash_str() {}
  hash_str(string &s) : h1(s.size()), p1(s.size()), h2(s.size()), p2(s.size())
      {
    int n = s.size();
    h1[0] = s[0];
    h2[0] = s[0];
    for (int i = 1; i < n; i++) {
      h1[i] = (h1[i - 1] * b1 + s[i]) % P;
      h2[i] = (h2[i - 1] * b2 + s[i]) % P;
    }
    p1[0] = 1;
    p2[0] = 1;
    for (int i = 1; i < n; i++) {
      p1[i] = (p1[i - 1] * b1) % P;
      p2[i] = (p2[i - 1] * b2) % P;
    }
  }
  ii substr(int l, int r) {  // <- 4 3 2 1 0
    if (l == 0) return {h1[r], h2[r]};
    ll ans1 = (h1[r] - h1[l - 1] * p1[r - l + 1]) % P;
    ll ans2 = (h2[r] - h2[l - 1] * p2[r - l + 1]) % P;
```

```
    if (ans1 < 0) {
      ans1 += P;
    }
    if (ans2 < 0) {
      ans2 += P;
    }
    return {ans1, ans2};
  }
  int size() { return h1.size(); }
};
```

1.23 dp-digit.cpp

```
int memo[20][2][4];

vi num2v(int n) {
  vi retval;
  if (n == 0) retval.push_back(0);
  while (n > 0) {
    retval.push_back(n % 10);
    n /= 10;
  }
  reverse(all(retval));
  return retval;
}

int dp(vi &digitos, int i = 0, bool empatado = true, int x = 0) {
  if (i == digitos.size()) {
    return 1;
  }
  int &ans = memo[i][empatado][x];
  if (ans != -1) return ans;
  ans = 0;
  int r = empatado ? digitos[i] : 9;
  for (int val = 0; val <= r; val++) {
    bool new_empatado = empatado && val == r;
    ans += dp(digitos, i + 1, new_empatado, x);
  }
  return ans;
}
```

1.24 dp-digit-single.cpp

```
int memo[20][2][2][4];

int dp(vi &x, vi &y, int i, bool el = true, bool er = true, int p = 0) {
  if (i == -1) {
    return p > 1;
  }
  int &ans = memo[i][el][er][p];
  if (ans != -1) return ans;
  ans = 0;
  int r = er ? y[i] : 9;
  int l = el ? x[i] : 0;
  for (int val = l; val <= r; val++) {
    bool new_er = er && val == r;
```

```
    bool new_el = el && val == l;
    ans += dp(x, y, i - 1, new_el, new_er, p + (val != 1));
  }
  return ans;
}

vi num2v(int n) {
  vi retval;
  if (n == 0) retval.push_back(0);
  while (n > 0) {
    retval.push_back(n % 10);
    n /= 10;
  }
  return retval;
}

int solve(int l, int r) {
  vi a = num2v(l);
  vi b = num2v(r);
  while (a.size() < b.size()) a.emplace_back(0);
  memset(memo, -1, sizeof(memo));
  return dp(a, b, a.size() - 1);
}
```

1.25 dsu.cpp

```
struct dsu {
  vi p, rnk;
  dsu(int n) : p(n), rnk(n) { iota(all(p), 0); }

  int find(int u) {
    if (p[u] == u) return u;
    return p[u] = find(p[u]);
  }

  void join(int u, int v) {
    u = find(u);
    v = find(v);
    if (u == v) return;
    if (rnk[u] > rnk[v]) {
      p[v] = u;
    } else {
      p[u] = v;
      if (rnk[u] == rnk[v]) rnk[v]++;
    }
  }
};
```

1.26 dsu-rollback.cpp

```
struct dsu_rollback {
  vi p, rnk, sz;
  dsu_rollback(int n) : p(n), rnk(n), sz(n, 1) { iota(all(p), 0); }

  int find(int u) {
    if (p[u] == u) return u;
```

```
    return find(p[u]);
  }

  stack<pair<int*, int>> stk;
  stack<int> qt;

  void join(int u, int v) {
    u = find(u);
    v = find(v);
    if (u == v) {
      qt.emplace(0);
      return;
    }
    if (rnk[u] > rnk[v]) {
      qt.emplace(2);
      stk.emplace(&p[v], p[v]);
      stk.emplace(&sz[u], sz[u]);
      p[v] = u;
      sz[u] += sz[v];
    } else {
      qt.emplace(3);
      stk.emplace(&p[u], p[u]);
      stk.emplace(&rnk[v], rnk[v]);
      stk.emplace(&sz[v], sz[v]);
      p[u] = v;
      sz[v] += sz[u];
      if (rnk[u] == rnk[v]) rnk[v]++;
    }
  }

  int checkpoint() { return stk.size(); }

  void undo(int prev_size) {
    while (stk.size() > prev_size) {
      auto [a, b] = stk.top();
      *a = b;
      stk.pop();
    }
  }
  void undo() {  // TODO:
    int q = qt.top();
    qt.pop();
    undo(stk.size() - q);
  }
};
```

1.27 euclides-estendido.cpp

```
#include <bits/stdc++.h>
using namespace std;

tuple<int, int, int> f(int a, int b) {
  if (b == 0) return {a, 1, 0};

  int g, x1, y1;
  tie(g, x1, y1) = f(b, a % b);
```

```
  return {g, y1, x1 - (a / b) * y1};
}

ll ext_euclid(ll a, ll b, ll &x, ll &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }

  ll _x, _y;
  ll gcd = ext_euclid(b, a % b, _x, _y);
  x = _y;
  y = _x - (a / b) * _y;
  return gcd;
}

// inverso modular
ll inv_mod(ll a, ll n) {
  ll x, y;
  ext_euclid(a, n, x, y);

  return (x % n + n) % n;
}

int main() {
  int a, b;
  scanf("%d %d", &a, &b);

  int x, y, mdc;
  tie(x, y, mdc) = f(a, b);

  cout << x << " " << y << " " << mdc << "\n";
  return 0;
}
```

1.28 expbin.cpp

```
ll expbin(ll a, ll b) {
  if (b == 0) return 1;
  if (b & 1) return a * expbin(a, b - 1);
  ll tmp = expbin(a, b / 2);
  return tmp * tmp;
}
```

1.29 expbin-fast.cpp

```
ll expbin(ll a, ll b) {
  ll res = 1;
  while (b > 0) {
    if (b & 1) res = res * a;
    a = a * a;
    b >>= 1;
  }
  return res;
```

```
}
```

1.30 expmatrix.cpp

```
struct Matrix {  // Estrutura para representar uma matriz
                 // com operador de multiplicação definido
  vector<vi> m;
  Matrix(bool identify = false) {
    m.resize(MAX, vi(MAX));
    for (auto &x : m)
      for (auto &y : x) y = 0;
    for (int i = 0; i < MAX; i++) m[i][i] = identify;
  }
  Matrix(vector<vi> mat) {
    m.resize(MAX, vi(MAX));
    for (auto &x : m)
      for (auto &y : x) y = 0;
    for (int i = 0; i < MAX; i++)
      for (int j = 0; j < MAX; j++) m[i][j] = mat[i][j];
  }
  vi &operator[](int pos) { return m[pos]; }
  Matrix operator*(Matrix oth) {
    Matrix ans;
    for (int i = 0; i < MAX; i++) {
      for (int j = 0; j < MAX; j++) {
        int &sum = ans[i][j];
        for (int k = 0; k < MAX; k++) {
          sum = (sum + (m[i][k] * oth[k][j]) % P) % P;
        }
      }
    }
    return ans;
  }
};

Matrix expbin(Matrix base, int exp) {  // Exponenciação binária
  Matrix ans(true);
  while (exp) {
    if (exp & 1LL) ans = ans * base;
    base = base * base;
    exp >>= 1;
  }
  return ans;
}
```

1.31 fft-fast.cpp

```
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
struct complex_t {
  double a{0.0}, b{0.0};
  complex_t() {}
  complex_t(double na) : a{na} {}
  complex_t(double na, double nb) : a{na}, b{nb} {}
  const complex_t operator+(const complex_t &c) const {
```

```
    return complex_t(a + c.a, b + c.b);
  }
  const complex_t operator-(const complex_t &c) const {
    return complex_t(a - c.a, b - c.b);
  }
  const complex_t operator*(const complex_t &c) const {
    return complex_t(a * c.a - b * c.b, a * c.b + b * c.a);
  }
  const complex_t operator/(const int &c) const {
    return complex_t(a / c, b / c);
  }
};
using cd = complex_t;
const double PI = acos(-1);
void fft(vector<cd> &a, bool invert) {
  int n = a.size();
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1) j ^= bit;
    j ^= bit;
    if (i < j) swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
      cd w(1);
      for (int j = 0; j < len / 2; j++) {
        cd u = a[i + j], v = a[i + j + len / 2] * w;
        a[i + j] = u + v;
        a[i + j + len / 2] = u - v;
        w = w * wlen;
      }
    }
  }
  if (invert) {
    for (cd &x : a) {
      x = x / n;
    }
  }
}

vi multiply(vi const &a, vi const &b) {
  vector<cd> fa(all(a));
  vector<cd> fb(all(b));
  int n = 1;
  while (n < int(a.size() + b.size())) n <<= 1;
  fa.resize(n);
  fb.resize(n);
  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < n; i++) fa[i] = fa[i] * fb[i];
  fft(fa, true);
  vi result(n);
  for (int i = 0; i < n; i++) result[i] = round(fa[i].a);
```

```
  return result;
}
```

1.32 floyd-warshall.cpp

```
// TODO: inicializar dist[i][j] = oo pras arestas que não existem
//             dist[i][j] = w(u, v) se a aresta existe
//             dist[i][i] = 0
void floyd_warshall() {
  for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}
```

1.33 geometria.cpp

```
const ld EPS = 1e-12;

struct point {
  ld x, y;
  point(ld _x, ld _y) : x(_x), y(_y) {}
  point() {}
};

struct vet {
  ld x, y;
  vet(ld _x, ld _y) : x(_x), y(_y) {}
  vet(point a, point b) : x(b.x - a.x), y(b.y - a.y) {}
  vet() {}
  vet operator-(vet oth) {
    vet c;
    c.x = this->x - oth.x;
    c.y = this->y - oth.y;
    return c;
  }
};

struct seg {
  point a, b;
  seg(point _a, point _b) : a(_a), b(_b) {}
  seg() {}
};

// produto vetorial
ld cross(vet u, vet v) { return u.x * v.y - u.y * v.x; }

// produto escalar
ld dot(vet u, vet v) { return u.x * v.x + u.y * v.y; }

// retorna verdadeiro se dois segmentos de reta se intersectam interiormente
bool intersect(seg x, seg y) {
  vet u(x.a, x.b), v(x.b, y.a), w(x.b, y.b);
  vet _u(y.a, y.b), _v(y.b, x.a), _w(y.b, x.b);
  return (cross(u, v) * cross(u, w) < -EPS) &&
         (cross(_u, _v) * cross(_u, _w) < -EPS);
```

```
}

// distancia ponto ponto
ld dpp(point a, point b) {
  vet u(a, b);
  return sqrtl(dot(u, u));
}

// distancia ponto segmento
ld dps(point a, seg x) {
  vet u(x.a, a), v(x.a, x.b);
  if (dot(v, v) < EPS) return dpp(x.a, a);
  ld lambda = dot(u, v) / dot(v, v);
  if (lambda < -EPS) return dpp(a, x.a);
  if (lambda > 1.) return dpp(a, x.b);
  return sqrtl(dot(u, u) - lambda * lambda * dot(v, v));
}

// distancia segmento segmento
ld dss(seg x, seg y) {
  if (intersect(x, y)) return 0;
  return min(dps(x.a, y), min(dps(x.b, y), min(dps(y.a, x), dps(y.b, x))));
}
```

1.34 gosper-hack.cpp

```
void GospersHack(int n, int k, function<void(ll)> f) {
  int msk = (1 << k) - 1;
  int limit = (1 << n);
  while (msk < limit) {
    f(msk);
    int c = msk & -msk;
    int r = msk + c;
    set = (((r ^ set) >> 2) / c) | r;
  }
}
```

1.35 hadamard.cpp

```
vi hadamard_transform(const vi& a) {
  vi dp = a;
  for (int bit = 1; bit < a.size(); bit <<= 1) {
    for (int mask = 0; mask < a.size(); mask++) {
      if ((mask & bit) == 0) {
        int u = dp[mask], v = dp[mask ^ bit];
        dp[mask] = u + v;
        dp[mask ^ bit] = u - v;
      }
    }
  }
  return dp;
}

vi inverse_hadamard_transform(const vi<int>& f) {
  vi dp = f;
  for (int bit = 1; bit < f.size(); bit <<= 1) {
```

```
    for (int mask = 0; mask < f.size(); mask++) {
      if ((mask & bit) == 0) {
        int x = dp[mask], y = dp[mask ^ bit];
        dp[mask] = (x + y) / 2;
        dp[mask ^ bit] = (x - y) / 2;
      }
    }
  }
  return dp;
}

// a.size() == b.size() == 2^k
vi xor_convolution(const vi<int>& a, const vi<int>& b) {
  vi f = hadamard_transform(a);
  vi g = hadamard_transform(b);
  vi h(f.size());
  for (int i = 0; i < f.size(); i++) {
    h[i] = f[i] * g[i];
  }
  vi c = inverse_hadamard_transform(h);
  return c;
}
```

1.36 hash2d.cpp

```
char mat[MAX][MAX];       // TODO
int hs[MAX][MAX];         // TODO
int PWX[MAX], PWY[MAX];   // TODO
int n, m;                 // TODO

struct Hashing {
  static const int PX = 3731, PY = 2999, mod = 998244353;
  Hashing() {
    PWX[0] = PWY[0] = 1;
    for (int i = 0; i < n; i++) PWX[i + 1] = 1LL * PWX[i] * PX % mod;
    for (int i = 0; i < m; i++) PWY[i + 1] = 1LL * PWY[i] * PY % mod;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        hs[i + 1][j + 1] = (((((1LL) * hs[i][j + 1] * PX) % mod +
                              ((1LL) * hs[i + 1][j] * PY) % mod) %
                                 mod -
                             (((1LL) * hs[i][j] * PX) % mod) * PY) %
                                mod +
                            (mat[i][j])) %
                           mod;
        hs[i + 1][j + 1] %= mod;
        hs[i + 1][j + 1] += mod;
        hs[i + 1][j + 1] %= mod;
      }
    }
  }
  int get_hash(int x1, int y1, int x2, int y2) {  // 1-indexado inclusivo
    x1--;
    y1--;
    int dx = x2 - x1, dy = y2 - y1;
    return (((1LL * hs[x2][y2]) - (1LL * hs[x2][y1]) * PWY[dy] % mod + mod) %
```

```
                mod -
            1LL * (hs[x1][y2] - (1LL * hs[x1][y1]) * PWY[dy] % mod + mod) %
                mod * PWX[dx] % mod +
            mod) %
           mod;
  }
  int get_hash() { return get_hash(1, 1, n, m); }
};
// TODO: simplificar
```

1.37 hash.cpp

```
const int P = 1e9 + 7;
mt19937 rmg((int)chrono::steady_clock::now().time_since_epoch().count());
struct hash_str {
  vector<ll> h, p;
  hash_str(string &s) : h(s.size()), p(s.size()) {
    int n = s.size();
    h[0] = s[0];
    int b = uniform_int_distribution<int>(0, P - 1)(rmg);
    for (int i = 1; i < n; i++) {
      h[i] = (h[i - 1] * b + s[i]) % P;
    }
    p[0] = 1;
    for (int i = 1; i < n; i++) {
      p[i] = (p[i - 1] * b) % P;
    }
  }
  ll substr(int l, int r) {  // <- 4 3 2 1 0
    if (l == 0) return h[r];
    ll ans = (h[r] - h[l - 1] * p[r - l + 1]) % P;
    if (ans < 0) {
      ans += P;
    }
    return ans;
  }
};
```

1.38 hash-generalizado.cpp

```
#define to_i(ch) (ch - 'a' + 1)
const ll P = 1e9 + 7;
const int MAX_PREC = 10;

mt19937 rmg((int)chrono::steady_clock::now().time_since_epoch().count());

bool flag = false;
vi b;
vector<vi> p;

struct hash_str {
  vector<vi> h;
  hash_str(string &s) {
    int n = s.size();
    h.resize(MAX_PREC);

```

```
    if (!flag) {
      p.resize(MAX_PREC);
      b.resize(MAX_PREC);
      for (int k = 0; k < MAX_PREC; k++) {
        b[k] = uniform_int_distribution<ll>(256, int(1e9 + 7) - 1)(rmg);
        p[k].resize(n);
        p[k][0] = 1;
        for (int i = 1; i < n; i++) {
          p[k][i] = (p[k][i - 1] * b[k]) % P;
        }
      }

      flag = true;
    }
    // h[0] = s[0]
    // h[1] = s[0] * b + s[1]
    // h[2] = s[0] * b^2 + s[1] * b + s[2]
    // h[3] = s[0] * b^3 + s[1] * b^2 + s[2] * b + s[3]
    for (int k = 0; k < MAX_PREC; k++) {
      h[k].resize(n);
      h[k][0] = to_i(s[0]);
      for (int i = 1; i < n; i++) {
        h[k][i] = (h[k][i - 1] * b[k] + to_i(s[i])) % P;
      }
    }
  }
  vi substr(int l, int r) {
    vi retval;
    if (l == 0) {
      for (int k = 0; k < MAX_PREC; k++) {
        retval.emplace_back(h[k][r]);
      }
      return retval;
    }
    for (int k = 0; k < MAX_PREC; k++) {
      ll ans = (h[k][r] - h[k][l - 1] * p[k][r - l + 1]) % P;
      if (ans < 0) {
        ans += P;
      }
      retval.emplace_back(ans);
    }
    return retval;
  }

  vi f(int i, char ch) {
    int a = i - 1;
    int _b = p[0].size() - i - 1;
    vi ha = a >= 0 ? sub_hash(0, a) : vi(MAX_PREC, 0LL);
    vi hb = i < (int)p[0].size() - 1 ? sub_hash(i + 1, p[0].size() - 1)
                                     : vi(MAX_PREC, 0LL);

    vi retval(MAX_PREC);
    for (int k = 0; k < MAX_PREC; k++) {
      retval[k] = (((ha[k] * (_b >= p[k].size() - 1 ? 1 : p[k][_b + 1])) % P +
                    (to_i(ch) * p[k][_b]) % P) %
```

```
                        P +
                    hb[k]) %
                   P;
    }
    return retval;
  }
};

int who[MAX][26];

bool __cmp(vi &a, vi &b) {
  int n = a.size();
  for (int i = 0; i < n; i++) {
    if (a[i] != b[i]) return false;
  }
  return true;
}

int main() {
  cin.tie(0);
  ios_base::sync_with_stdio(0);

  string s, p;
  cin >> s >> p;

  int n = s.size();
  int m = p.size();

  hash_str hs(s), hp(p);

  map<vi, ii> tab;
  vi hash_orig = hp.sub_hash(0, m - 1);

  // Pegar todos os hashes do padrão, para cada uma das alterações
  for (int i = 0; i < m; i++) {
    for (char ch = 'a'; ch <= 'z'; ch++) {
      vi val = hp.f(i, ch);
      //        cout << val[0] << "\n";
      tab[val] = make_pair(i, p[i]);
    }
  }

  int total = 0;
  for (int i = 0; i < n - m + 1; i++) {
    vi val = hs.sub_hash(i, i + m - 1);
    int pos;
    char ch;
    if (tab.find(val) == tab.end()) {
      continue;
    }
    total += int(__cmp(val, hash_orig));

    tie(pos, ch) = tab[val];
    who[i + pos][ch - 'a']++;
  }
```

```
127
128   int q;
129   cin >> q;
130   while (q--) {
131     int i;
132     char ch;
133     cin >> i >> ch;
134     --i;
135
136     int ans = total + who[i][ch - 'a'] - who[i][s[i] - 'a'];
137     cout << ans << "\n";
138   }
139   return 0;
140 }
```

1.39 hld.cpp

```
1  vi adj[MAX];
2
3  vi sz(MAXN), h(MAXN), par(MAXN), pos(MAXN), sop(MAXN), head(MAXN), tail(MAXN);
4
5  void remove_parent(int u = 0) {
6    for (int v : adj[u]) {
7      adj[v].erase(find(all(adj[v]), u));
8      remove_parent(v);
9    }
10 }
11
12 int fill(int u = 0) {
13   for (int i = 0; i < adj[u].size(); i++) {
14     int& v = adj[u][i];
15     h[v] = h[u] + 1;
16     par[v] = u;
17     sz[u] += fill(v);
18     if (sz[adj[u][0]] < sz[v]) swap(adj[u][0], adj[u][i]);
19   }
20   return ++sz[u];
21 }
22
23 void hld(int u = 0) {
24   static int ids = 0;
25   sop[pos[u] = ids++] = u;
26   for (int v : adj[u]) {
27     head[v] = (v == adj[u][0] ? head[u] : v);
28     hld(v);
29   }
30   tail[u] = adj[u].size() ? tail[adj[u][0]] : u;
31 }
32
33 void build() {}  // TODO: Sparse table / Seg tree ...
34
35 int gethld(int u, int v) {
36   int ans = oo;  // TODO:
37   if (pos[u] > pos[v]) swap(u, v);
38   while (head[u] != head[v]) {
39     ans = min(ans, getmin(pos[head[v]], pos[v] + 1));  // TODO
```

```
    v = par[head[v]];
    if (pos[u] > pos[v]) swap(u, v);
  }
  return ans = min(ans,
                   getmin(pos[u] + 1 /* TODO: edge = 1*/, pos[v] + 1));  //
                      TODO
}
```

1.40 hungarian.cpp

```
class Solution {
 public:
  int cost[MAX][MAX];     // cost matrix
  int n, max_match;       // n workers and n jobs
  int lx[MAX], ly[MAX];   // labels of X and Y parts
  int xy[MAX];            // xy[x] - vertex that is matched with x,
  int yx[MAX];            // yx[y] - vertex that is matched with y
  bool S[MAX], T[MAX];    // sets S and T in algorithm
  int slack[MAX];         // as in the algorithm description
  int slackx[MAX];        // slackx[y] such a vertex, that
  int prev_ious[MAX];     // array for memorizing alternating p

  void init_labels() {
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    for (int x = 0; x < n; x++)
      for (int y = 0; y < n; y++) lx[x] = max(lx[x], cost[x][y]);
  }

  void update_labels() {
    int x, y;
    int delta = 99999999;     // init delta as infinity
    for (y = 0; y < n; y++)  // calculate delta using slack
      if (!T[y]) delta = min(delta, slack[y]);
    for (x = 0; x < n; x++)  // update X labels
      if (S[x]) lx[x] -= delta;
    for (y = 0; y < n; y++)  // update Y labels
      if (T[y]) ly[y] += delta;
    for (y = 0; y < n; y++)  // update slack array
      if (!T[y]) slack[y] -= delta;
  }

  void add_to_tree(int x, int prev_iousx)
  // x - current vertex,prev_iousx - vertex from X before x in the alternating
  // path, so we add edges (prev_iousx, xy[x]), (xy[x], x)
  {
    S[x] = true;                    // add x to S
    prev_ious[x] = prev_iousx;    // we need this when augmenting
    for (int y = 0; y < n; y++)  // update slacks, because we add new vertex
       to
                                    // S
      if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
        slack[y] = lx[x] + ly[y] - cost[x][y];
        slackx[y] = x;
      }
  }
```

```
46
47  void augment()  // main function of the algorithm
48  {
49    if (max_match == n) return;  // check whether matching is already perfect
50    int x, y, root;              // just counters and root vertex
51    int q[MAX], wr = 0, rd = 0;  // q - queue for bfs, wr,rd - write and read
52    // pos in queue
53    memset(S, false, sizeof(S));  // init set S
54    memset(T, false, sizeof(T));  // init set T
55    memset(prev_ious, -1,
56           sizeof(prev_ious));  // init set prev_ious - for the alternating
                                   tree
57
58    for (x = 0; x < n; x++)  // finding root of the tree
59    {
60      if (xy[x] == -1) {
61        q[wr++] = root = x;
62        prev_ious[x] = -2;
63        S[x] = true;
64        break;
65      }
66    }
67
68    for (y = 0; y < n; y++)  // initializing slack array
69    {
70      slack[y] = lx[root] + ly[y] - cost[root][y];
71      slackx[y] = root;
72    }
73
74    // second part of augment() function
75    while (true)  // main cycle
76    {
77      while (rd < wr)  // building tree with bfs cycle
78      {
79        x = q[rd++];               // current vertex from X part
80        for (y = 0; y < n; y++)  // iterate through all edges in equality
                                     graph
81          if (cost[x][y] == lx[x] + ly[y] && !T[y]) {
82            if (yx[y] == -1)
83              break;           // an exposed vertex in Y found, so
84                               // augmenting path exists!
85            T[y] = true;       // else just add y to T,
86            q[wr++] = yx[y];  // add vertex yx[y], which is matched
87            // with y, to the queue
88            add_to_tree(yx[y], x);  // add edges (x,y) and (y,yx[y]) to the
                                       tree
89          }
90        if (y < n) break;  // augmenting path found!
91      }
92      if (y < n) break;  // augmenting path found!
93
94      update_labels();  // augmenting path not found, so improve labeling
95
96      wr = rd = 0;
97      for (y = 0; y < n; y++)
```

```
98          // in this cycle we add edges that were added to the equality graph as
                a
99          // result of improving the labeling, we add edge (slackx[y], y) to the
100         // tree if and only if !T[y] && slack[y] == 0, also with this edge we
101         // add another one (y, yx[y]) or augment the matching, if y was
                exposed
102         if (!T[y] && slack[y] == 0) {
103           if (yx[y] ==
104               -1)  // exposed vertex in Y found - augmenting path exists!
105           {
106             x = slackx[y];
107             break;
108           } else {
109             T[y] = true;  // else just add y to T,
110             if (!S[yx[y]]) {
111               q[wr++] = yx[y];  // add vertex yx[y], which is matched with
112               // y, to the queue
113               add_to_tree(yx[y], slackx[y]);  // and add edges (x,y) and (y,
114               // yx[y]) to the tree
115             }
116           }
117         }
118       if (y < n) break;  // augmenting path found!
119     }
120
121     if (y < n)  // we found augmenting path!
122     {
123       max_match++;  // increment matching
124       // in this cycle we inverse edges along augmenting path
125       for (int cx = x, cy = y, ty; cx != -2; cx = prev_ious[cx], cy = ty) {
126         ty = xy[cx];
127         yx[cy] = cx;
128         xy[cx] = cy;
129       }
130       augment();  // recall function, go to step 1 of the algorithm
131     }
132   }  // end of augment() function
133
134   int hungarian() {
135     int ret = 0;     // weight of the optimal matching
136     max_match = 0;  // number of vertices in current matching
137     memset(xy, -1, sizeof(xy));
138     memset(yx, -1, sizeof(yx));
139     init_labels();  // step 0
140     augment();      // steps 1-3
141
142     for (int x = 0; x < n; x++)  // forming answer there
143       ret += cost[x][xy[x]];
144
145     return ret;
146   }
147
148   int assignmentProblem(int Arr[], int N) {
149     n = N;
150     for (int i = 0; i < n; i++)
```

```
151       for (int j = 0; j < n; j++) cost[i][j] = -1 * Arr[i * n + j];
152
153     int ans = -1 * hungarian();
154
155     return ans;
156   }
157 };
```

1.41 interpolacao.cpp

```
1  // from:
2  // https://github.com/PauloMiranda98/Competitive-Programming-Notebook/blob/54
       af0a8dcefdeb5505538a3716855db62bcdc716/code/math/lagrange.h#L30
3  typedef long double ld;
4  struct PointValue {
5    ld x, y;
6    PointValue(ld x0 = 0, ld y0 = 0) : x(x0), y(y0) {}
7  };
8  void mul(vector<ld> &A, int x0) {  // multiply A(x) by (x - x0)
9    int n = A.size();
10   A.push_back(0);
11   auto B = A;
12   for (int i = n; i >= 1; i--) {
13     A[i] = A[i - 1];
14   }
15   A[0] = 0;
16   for (int i = 0; i < n + 1; i++) A[i] -= B[i] * x0;
17 }
18 void div(vector<ld> &A, int x0) {  // multiply A(x) by (x - x0)
19   int g = (int)A.size() - 1;
20   vector<ld> aux(g);
21   for (int i = g; i >= 1; i--) {
22     aux[i - 1] = A[i];
23     A[i - 1] += x0 * aux[i - 1];
24   }
25   A = aux;
26 }
27 // Change Polynomial Representation from Point-Value to Coefficient
28 // O(n^2)
29 vector<ld> LagrangeInterpolation(vector<PointValue> vp) {
30   vector<ld> A(1, 1);
31   int n = vp.size();
32   for (int i = 0; i < n; i++) mul(A, vp[i].x);
33   vector<ld> ans(n, 0);
34   for (int i = 0; i < n; i++) {
35     ld x = vp[i].x, y = vp[i].y;
36     div(A, x);
37     ld d = 1;
38     for (int j = 0; j < n; j++) {
39       if (j != i) d *= (x - vp[j].x);
40     }
41     for (int j = 0; j < n; j++) ans[j] += A[j] * (y / d);
42     mul(A, vp[i].x);
43   }
44   return ans;
45 }
```

1.42 kmp-automata.cpp

```
int b[MAX], st[MAX][K];

void kmpp(string &p) {
  int i = 0, j = -1, n = p.size();
  b[0] = -1;
  while (i < n) {
    while (j >= 0 && p[i] != p[j]) j = b[j];
    ++i, ++j;
    b[i] = j;
  }
  for (int i = 0; i < n; i++)
    for (int ch = 0; ch < K; ch++)
      if (to_i(p[i]) == ch)
        st[i][ch] = i + 1;
      else
        st[i][ch] = b[i] == -1 ? 0 : st[b[i]][ch];
}

bool kmp(string &t, string &p) {
  int i = 0, j = 0, n = t.size(), m = p.size();
  while (i < n) {
    j = st[j][to_i(t[i])];
    ++i;
    if (j == m) {
      j = b[j];
      return true;
    }
  }
  return false;
}
```

1.43 kmp.cpp

```
int b[MAX];
void kmpp(string &p) {
  int i = 0, j = -1;
  b[0] = -1;
  int n = p.size();
  while (i < n) {
    while (j >= 0 && p[i] != p[j]) j = b[j];
    ++i, ++j;
    b[i] = j;
  }
}

bool kmp(string &t, string &p) {
  int i = 0, j = 0, n = t.size(), m = p.size();
  while (i < n) {
    while (j >= 0 && t[i] != p[j]) j = b[j];
    ++i, ++j;
    if (j == m) {
      j = b[j];
      return true;
    }
```

```
  }
  return false;
}
```

1.44 knapsack.cpp

```
int memo[MAXN][MAXM];
vi c, v;

int knap(int i, int j) {
  if (i == c.size()) return memo[i][j] = 0;
  if (memo[i][j] != -1) return memo[i][j];

  int ch1 = -oo;
  if (c[i] <= j) {
    ch1 = knap(i + 1, j - c[i]) + v[i];
  }
  int ch2 = knap(i + 1, j);

  return memo[i][j] = max(ch1, ch2);
}
```

1.45 kruskal.cpp

```
vector<iii> edges;  // {peso, u, v}

int kruskal(int n) {
  sort(all(edges));
  dsu d(n);
  int cost = 0;
  for (auto &[w, u, v] : edges)
    if (d.find(u) != d.find(v)) {
      cost += w;
      d.join(u, v);
    }
  return cost;
}
```

1.46 kuhn.cpp

```
// Kuhn
//
// Computa matching maximo em grafo bipartido
// 'n' e 'm' sao quantos vertices tem em cada particao
// chamar add(i, j) para add aresta entre o cara i
// da particao A, e o cara j da particao B
// (entao i < n, j < m)
// Para recuperar o matching, basta olhar 'ma' e 'mb'
// 'recover' recupera o min vertex cover como um par de
// {caras da particao A, caras da particao B}
//
// O(|V| * |E|)
// Na pratica, parece rodar tao rapido quanto o Dinitz

mt19937 rng((int)chrono::steady_clock::now().time_since_epoch().count());

struct kuhn {
```

```
  int n, m;
  vector<vector<int>> g;
  vector<int> vis, ma, mb;

  kuhn(int n_, int m_) : n(n_), m(m_), g(n), vis(n + m), ma(n, -1), mb(m, -1)
     {}

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

  bool dfs(int i) {
    vis[i] = 1;
    for (int j : g[i])
      if (!vis[n + j]) {
        vis[n + j] = 1;
        if (mb[j] == -1 or dfs(mb[j])) {
          ma[i] = j, mb[j] = i;
          return true;
        }
      }
    return false;
  }
  int matching() {
    int ret = 0, aum = 1;
    for (auto& i : g) shuffle(i.begin(), i.end(), rng);
    while (aum) {
      for (int j = 0; j < m; j++) vis[n + j] = 0;
      aum = 0;
      for (int i = 0; i < n; i++)
        if (ma[i] == -1 and dfs(i)) ret++, aum = 1;
    }
    return ret;
  }
};

pair<vector<int>, vector<int>> recover(kuhn& K) {
  K.matching();
  int n = K.n, m = K.m;
  for (int i = 0; i < n + m; i++) K.vis[i] = 0;
  for (int i = 0; i < n; i++)
    if (K.ma[i] == -1) K.dfs(i);
  vector<int> ca, cb;
  for (int i = 0; i < n; i++)
    if (!K.vis[i]) ca.push_back(i);
  for (int i = 0; i < m; i++)
    if (K.vis[n + i]) cb.push_back(i);
  return {ca, cb};
}
```

1.47 lca.cpp

```
int p[MAX], h[MAX], tab[LOGN][MAX];
int n;  // TODO
ll dist[MAX];

void dfs(int u, int _p) {
  p[u] = _p;
```

```
  for (auto &[v, c] : adj[u]) {
    if (v == _p) continue;
    dist[v] = dist[u] + c;
    h[v] = h[u] + 1;
    dfs(v, u);
  }
}

void build() {  // TODO
  dfs(0, 0);    // TODO
  for (int j = 0; j < n; j++) tab[0][j] = p[j];
  for (int i = 1; i < LOGN; i++)
    for (int j = 0; j < n; j++) tab[i][j] = tab[i - 1][tab[i - 1][j]];
}

int goup(int u, int k) {
  for (int i = 0; i < LOGN; i++)
    if (k & (1LL << i)) u = tab[i][u];
  return u;
}

int lca(int u, int v) {
  if (h[u] < h[v]) swap(u, v);
  u = goup(u, h[u] - h[v]);
  if (u == v) return u;
  for (int i = LOGN - 1; i >= 0; i--)
    if (tab[i][u] != tab[i][v]) {
      u = tab[i][u];
      v = tab[i][v];
    }
  return tab[0][u];
}

ll solve(int u, int v) {  // Distancia entre dois vertices com custo
  if (u > v) swap(u, v);
  if (u == 0) return dist[v];  // TODO
  return dist[u] + dist[v] - 2 * dist[lca(u, v)];
}
```

1.48 lcs.cpp

```
int lcs(string &s, string &t) {
  int n = s.size();
  int m = t.size();
  vector<vi> dp = vector<vi>(n + 1, vi(m + 1));
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
      int ch1 = dp[i - 1][j];
      int ch2 = dp[i][j - 1];
      int ch3 = s[i - 1] == t[j - 1] ? dp[i - 1][j - 1] + 1 : 0;
      dp[i][j] = max({ch1, ch2, ch3});
    }
  }
  return dp[n][m];
}
```

1.49 line.cpp

```cpp
struct line {
  int a, b, c;
  line() {}
  // a * x + b * y + c = 0
  line(int _a, int _b, int _c) : a(_a), b(_b), c(_c) {}

  line(ii x, ii y) {
    a = (x.second - y.second);
    b = (-x.first + y.first);
    c = x.first * y.second - x.second * y.first;

    int g = gcd(a, gcd(b, c));
    a /= g;
    b /= g;
    c /= g;
  }

  bool is(ii x) {
    int v1 = a * x.first;
    int v2 = b * x.second;
    return v1 + v2 == -c;
  }

  double gety(int x) { return ((-a * x - c) * (1.)) / b; }

  double getx(int y) { return ((-b * y - c) * (1.)) / a; }
};
```

1.50 lis.cpp

```cpp
vector<ll> bit(MAX, 0LL);

void add(int i, ll delta) {
  for (; i < MAX; i += i & (-i)) bit[i] = max(bit[i], delta);
}

ll get(int i) {
  if (i <= 0) return 0LL;
  ll ans = 0LL;
  for (; i > 0; i -= i & (-i)) ans = max(ans, bit[i]);
  return ans;
}

int lis(vi &vet) {
  int n = vet.size();
  vector<ii> tmp(n);
  for (int i = 0; i < n; i++) {
    tmp[i].first = vet[i];
    tmp[i].second = -(i + 1);
  }
  sort(all(tmp));
  int ans = 0;
  fill(all(bit), 0);
  for (int i = 0; i < n; i++) {
```

```
    int pos = -tmp[i].second;
    int now = get(pos - 1);
    ans = max(ans, now + 1);
    add(pos, now + 1);
  }
  return ans;
}
```

1.51 manacher.cpp

```
// manacher recebe um vetor de T e retorna o vetor com tamanho dos palindromos
// ret[2*i] = tamanho do maior palindromo centrado em i
// ret[2*i+1] = tamanho maior palindromo centrado em i e i+1
//
// Complexidades:
// manacher - O(n)
// palindrome - <O(n), O(1)>
// pal_end - O(n)

template <typename T>
vi manacher(const T& s) {
  int l = 0, r = -1, n = s.size();
  vi d1(n), d2(n);
  for (int i = 0; i < n; i++) {
    int k = i > r ? 1 : min(d1[l + r - i], r - i);
    while (i + k < n && i - k >= 0 && s[i + k] == s[i - k]) k++;
    d1[i] = k--;
    if (i + k > r) l = i - k, r = i + k;
  }
  l = 0, r = -1;
  for (int i = 0; i < n; i++) {
    int k = i > r ? 0 : min(d2[l + r - i + 1], r - i + 1);
    k++;
    while (i + k <= n && i - k >= 0 && s[i + k - 1] == s[i - k]) k++;
    d2[i] = --k;
    if (i + k - 1 > r) l = i - k, r = i + k - 1;
  }
  vi ret(2 * n - 1);
  for (int i = 0; i < n; i++) ret[2 * i] = 2 * d1[i] - 1;
  for (int i = 0; i < n - 1; i++) ret[2 * i + 1] = 2 * d2[i + 1];
  return ret;
}

// verifica se a string s[i..j] eh palindromo
template <typename T>
struct palindrome {
  vi man;

  palindrome(const T& s) : man(manacher(s)) {}
  bool query(int i, int j) { return man[i + j] >= j - i + 1; }
};

// tamanho do maior palindromo que termina em cada posicao
template <typename T>
vi pal_end(const T& s) {
  vi ret(s.size());
```

```
  palindrome<T> p(s);
  ret[0] = 1;
  for (int i = 1; i < s.size(); i++) {
    ret[i] = min(ret[i - 1] + 2, i + 1);
    while (!p.query(i - ret[i] + 1, i)) ret[i]--;
  }
  return ret;
}
// usage: auto pal = palindrome<string>(str)
//        pal.query(l, r) [l, r] 0-indexado
```

1.52 mergesort-tree.cpp

```
struct merge_sort_tree {
  struct item {
    vi vet;
    item operator+(item oth) {
      item c;
      c.vet = vi(this->vet.size() + oth.vet.size());
      for (int k = 0, i = 0, j = 0; k < c.vet.size(); k++) {
        if (j == oth.vet.size() ||
            (i < this->vet.size() && this->vet[i] < oth.vet[j])) {
          c.vet[k] = this->vet[i++];
        } else {
          c.vet[k] = oth.vet[j++];
        }
      }
      return c;
    }
  };

  int n;
  vector<item> seg;

  merge_sort_tree(vi &vet) : seg(4 * vet.size()), n(vet.size()) {
    build(vet, 0, 0, n);
  }

  void build(vi &vet, int pos, int lx, int rx) {
    if (rx - lx == 1) {
      seg[pos].vet.emplace_back(vet[lx]);
      return;
    }
    int mid = lx + (rx - lx) / 2;
    build(vet, 2 * pos + 1, lx, mid);
    build(vet, 2 * pos + 2, mid, rx);
    seg[pos] = seg[2 * pos + 1] + seg[2 * pos + 2];
  }

  int get(int l, int r, int x, int pos, int lx, int rx) {
    if (lx >= r || rx <= l) return 0;
    if (lx >= l && rx <= r) {
      if (seg[pos].vet[0] > x) return 0;
      int a = 0;                      // good
      int b = seg[pos].vet.size();  // bad
      while (a + 1 < b) {
```

```
        int mid = a + (b - a) / 2;
        if (seg[pos].vet[mid] <= x)
          a = mid;
        else
          b = mid;
      }
      return a + 1;
    }
    int mid = lx + (rx - lx) / 2;
    return get(l, r, x, 2 * pos + 1, lx, mid) +
           get(l, r, x, 2 * pos + 2, mid, rx);
  }

  // qtde <= x no intervalo [l, r)?
  int get(int l, int r, int x) { return get(l, r, x, 0, 0, n); }
};
```

1.53 min-cost-max-flow.cpp

```
// MinCostMaxFlow
//
// min_cost_flow(s, t, f) computa o par (fluxo, custo)
// com max(fluxo) <= f que tenha min(custo)
// min_cost_flow(s, t) -> Fluxo maximo de custo minimo de s pra t
// Se for um dag, da pra substituir o SPFA por uma DP pra nao
// pagar O(nm) no comeco
// Se nao tiver aresta com custo negativo, nao precisa do SPFA
//
// O(nm + f * m log n)

template <typename T>
struct mcmf {
  struct edge {
    int to, rev, flow, cap;  // para, id da reversa, fluxo, capacidade
    bool res;                // se eh reversa
    T cost;                  // custo da unidade de fluxo
    edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
    edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_)
        : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_), cost(cost_)
          {}
  };

  vector<vector<edge>> g;
  vector<int> par_idx, par;
  T inf;
  vector<T> dist;

  mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits<T>::max() / 3) {}

  void add(int u, int v, int w, T cost) {  // de u pra v com cap w e custo
     cost
    edge a = edge(v, g[v].size(), 0, w, cost, false);
    edge b = edge(u, g[u].size(), 0, 0, -cost, true);

    g[u].push_back(a);
    g[v].push_back(b);
```

```
  }

  vector<T> spfa(int s) {  // nao precisa se nao tiver custo negativo
    deque<int> q;
    vector<bool> is_inside(g.size(), 0);
    dist = vector<T>(g.size(), inf);

    dist[s] = 0;
    q.push_back(s);
    is_inside[s] = true;

    while (!q.empty()) {
      int v = q.front();
      q.pop_front();
      is_inside[v] = false;

      for (int i = 0; i < g[v].size(); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;

          if (is_inside[to]) continue;
          if (!q.empty() and dist[to] > dist[q.front()])
            q.push_back(to);
          else
            q.push_front(to);
          is_inside[to] = true;
        }
      }
    }
    return dist;
  }
  bool dijkstra(int s, int t, vector<T>& pot) {
    priority_queue<pair<T, int>, vector<pair<T, int>>, greater<>> q;
    dist = vector<T>(g.size(), inf);
    dist[s] = 0;
    q.emplace(0, s);
    while (q.size()) {
      auto [d, v] = q.top();
      q.pop();
      if (dist[v] < d) continue;
      for (int i = 0; i < g[v].size(); i++) {
        auto [to, rev, flow, cap, res, cost] = g[v][i];
        cost += pot[v] - pot[to];
        if (flow < cap and dist[v] + cost < dist[to]) {
          dist[to] = dist[v] + cost;
          q.emplace(dist[to], to);
          par_idx[to] = i, par[to] = v;
        }
      }
    }
    return dist[t] < inf;
  }

  pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
```

```
    vector<T> pot(g.size(), 0);
    pot = spfa(s);  // mudar algoritmo de caminho minimo aqui

    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
      for (int i = 0; i < g.size(); i++)
        if (dist[i] < inf) pot[i] += dist[i];

      int mn_flow = flow - f, u = t;
      while (u != s) {
        mn_flow = min(mn_flow,
                      g[par[u]][par_idx[u]].cap - g[par[u]][par_idx[u]].flow);
        u = par[u];
      }

      ret += pot[t] * mn_flow;

      u = t;
      while (u != s) {
        g[par[u]][par_idx[u]].flow += mn_flow;
        g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
        u = par[u];
      }

      f += mn_flow;
    }

    return make_pair(f, ret);
  }

  // Opcional: retorna as arestas originais por onde passa flow = cap
  vector<pair<int, int>> recover() {
    vector<pair<int, int>> used;
    for (int i = 0; i < g.size(); i++)
      for (edge e : g[i])
        if (e.flow == e.cap && !e.res) used.push_back({i, e.to});
    return used;
  }
};
```

1.54 minqueue.cpp

```
struct MinQueue {
  deque<ii> d;
  int ini, fim;

  MinQueue() {
    ini = 1;
    fim = 0;
  }

  void push(int v) {
    while (!d.empty() && d.back().first >= v) d.pop_back();
    d.push_back(ii(v, ++fim));
  }
```

```
14
15   void pop() {
16     if (!d.empty() && d.front().second == ini++) d.pop_front();
17   }
18
19   int get() { return d.front().first; }
20 };
```

1.55 mint.cpp

```
1  // Aritmetica Modular
2  //
3  // O mod tem q ser primo
4
5  template <int p>
6  struct mod_int {
7    ll expo(ll b, ll e) {
8      ll ret = 1;
9      while (e) {
10       if (e % 2) ret = ret * b % p;
11       e /= 2, b = b * b % p;
12     }
13     return ret;
14   }
15   ll inv(ll b) { return expo(b, p - 2); }
16
17   using m = mod_int;
18   int v;
19   mod_int() : v(0) {}
20   mod_int(ll v_) {
21     if (v_ >= p or v_ <= -p) v_ %= p;
22     if (v_ < 0) v_ += p;
23     v = v_;
24   }
25   m& operator+=(const m& a) {
26     v += a.v;
27     if (v >= p) v -= p;
28     return *this;
29   }
30   m& operator-=(const m& a) {
31     v -= a.v;
32     if (v < 0) v += p;
33     return *this;
34   }
35   m& operator*=(const m& a) {
36     v = v * ll(a.v) % p;
37     return *this;
38   }
39   m& operator/=(const m& a) {
40     v = v * inv(a.v) % p;
41     return *this;
42   }
43   m operator-() { return m(-v); }
44   m& operator^=(ll e) {
45     if (e < 0) {
46       v = inv(v);
```

```
      e = -e;
    }
    v = expo(v, e % (p - 1));
    return *this;
  }
  bool operator==(const m& a) { return v == a.v; }
  bool operator!=(const m& a) { return v != a.v; }

  friend istream& operator>>(istream& in, m& a) {
    ll val;
    in >> val;
    a = m(val);
    return in;
  }
  friend ostream& operator<<(ostream& out, m a) { return out << a.v; }
  friend m operator+(m a, m b) { return a += b; }
  friend m operator-(m a, m b) { return a -= b; }
  friend m operator*(m a, m b) { return a *= b; }
  friend m operator/(m a, m b) { return a /= b; }
  friend m operator^(m a, ll e) { return a ^= e; }
};

typedef mod_int<(int)1e9 + 7> mint;
```

1.56 mo.cpp

```
int B = 1;  // TODO

struct item {
  int l, r, id;
  item() {}
  item(int _l, int _r, int _id) : l(_l), r(_r), id(_id) {}
};

bool cmp(item a, item b) {
  if (a.l / B != b.l / B) return make_pair(a.l, a.r) < make_pair(b.l, b.r);
  return (a.l / B) % 2 ? a.r < b.r : a.r > b.r;
}

void add(int i) {
  // TODO
}

void rem(int i) {
  // TODO
}

vi mo(vector<item> &ev) {
  sort(all(ev), cmp);
  vi retval(ev.size());
  int _l = 0, _r = -1;
  for (auto &[l, r, id] : ev) {
    while (_l > l) add(--_l);
    while (_r < r) add(++_r);
    while (_l < l) rem(_l++);
    while (_r > r) rem(_r--);
```

```
    retval[id] = ans;
  }
  return retval;
}
```

1.57 ntt.cpp

```
const int N = 1 << 18;
const int mod = 998244353;
const int root = 3;
int lim, rev[N], w[N], wn[N], inv_lim;
void reduce(int &x) { x = (x + mod) % mod; }
int POW(int x, int y, int ans = 1) {
  for (; y; y >>= 1, x = (ll)x * x % mod)
    if (y & 1) ans = (ll)ans * x % mod;
  return ans;
}
void precompute(int len) {
  lim = wn[0] = 1;
  int s = -1;
  while (lim < len) lim <<= 1, ++s;
  for (int i = 0; i < lim; ++i) rev[i] = rev[i >> 1] >> 1 | (i & 1) << s;
  const int g = POW(root, (mod - 1) / lim);
  inv_lim = POW(lim, mod - 2);
  for (int i = 1; i < lim; ++i) wn[i] = (ll)wn[i - 1] * g % mod;
}
void ntt(vi &a, int typ) {
  for (int i = 0; i < lim; ++i)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int i = 1; i < lim; i <<= 1) {
    for (int j = 0, t = lim / i / 2; j < i; ++j) w[j] = wn[j * t];
    for (int j = 0; j < lim; j += i << 1) {
      for (int k = 0; k < i; ++k) {
        const int x = a[k + j], y = (ll)a[k + j + i] * w[k] % mod;
        reduce(a[k + j] += y - mod), reduce(a[k + j + i] = x - y);
      }
    }
  }
  if (!typ) {
    reverse(a.begin() + 1, a.begin() + lim);
    for (int i = 0; i < lim; ++i) a[i] = (ll)a[i] * inv_lim % mod;
  }
}

vi multiply(vi &f, vi &g) {
  int n = (int)f.size() + (int)g.size() - 1;
  precompute(n);
  vi a = f, b = g;
  a.resize(lim);
  b.resize(lim);
  ntt(a, 1), ntt(b, 1);
  for (int i = 0; i < lim; ++i) a[i] = (ll)a[i] * b[i] % mod;
  ntt(a, 0);
  while ((int)a.size() && a.back() == 0) a.pop_back();
  return a;
}
```

1.58 on-suffix-array.cpp

```
vi s;

inline bool leq(int a1, int a2, int b1, int b2) {
  return (a1 < b1 || (a1 == b1 && a2 <= b2));
}

inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3) {
  return (a1 < b1 || (a1 == b1 && leq(a2, a3, b2, b3)));
}

static void radixPass(int* a, int* b, int* r, int n, int K) {
  int* c = new int[K + 1];
  for (int i = 0; i <= K; i++) c[i] = 0;
  for (int i = 0; i < n; i++) c[r[a[i]]]++;
  for (int i = 0, sum = 0; i <= K; i++) {
    int t = c[i];
    c[i] = sum;
    sum += t;
  }
  for (int i = 0; i < n; i++) b[c[r[a[i]]]++] = a[i];
  delete[] c;
}

void suffixArray(int* s, int* SA, int n, int K) {
  int n0 = (n + 2) / 3, n1 = (n + 1) / 3, n2 = n / 3, n02 = n0 + n2;
  int* s12 = new int[n02 + 3];
  s12[n02] = s12[n02 + 1] = s12[n02 + 2] = 0;
  int* SA12 = new int[n02 + 3];
  SA12[n02] = SA12[n02 + 1] = SA12[n02 + 2] = 0;
  int* s0 = new int[n0];
  int* SA0 = new int[n0];
  for (int i = 0, j = 0; i < n + (n0 - n1); i++)
    if (i % 3 != 0) s12[j++] = i;
  radixPass(s12, SA12, s + 2, n02, K);
  radixPass(SA12, s12, s + 1, n02, K);
  radixPass(s12, SA12, s, n02, K);
  int name = 0, c0 = -1, c1 = -1, c2 = -1;
  for (int i = 0; i < n02; i++) {
    if (s[SA12[i]] != c0 || s[SA12[i] + 1] != c1 || s[SA12[i] + 2] != c2) {
      name++;
      c0 = s[SA12[i]];
      c1 = s[SA12[i] + 1];
      c2 = s[SA12[i] + 2];
    }
    if (SA12[i] % 3 == 1)
      s12[SA12[i] / 3] = name;
    else
      s12[SA12[i] / 3 + n0] = name;
  }
  if (name < n02) {
    suffixArray(s12, SA12, n02, name);
    for (int i = 0; i < n02; i++) s12[SA12[i]] = i + 1;
  } else
    for (int i = 0; i < n02; i++) SA12[s12[i] - 1] = i;
```

```
  for (int i = 0, j = 0; i < n02; i++)
    if (SA12[i] < n0) s0[j++] = 3 * SA12[i];
  radixPass(s0, SA0, s, n0, K);
  for (int p = 0, t = n0 - n1, k = 0; k < n; k++) {
#define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
    int i = GetI();
    int j = SA0[p];
    if (SA12[t] < n0 ? leq(s[i], s12[SA12[t] + n0], s[j], s12[j / 3])
                     : leq(s[i], s[i + 1], s12[SA12[t] - n0 + 1], s[j],
                           s[j + 1], s12[j / 3 + n0])) {
      SA[k] = i;
      t++;
      if (t == n02)
        for (k++; p < n0; p++, k++) SA[k] = SA0[p];
    } else {
      SA[k] = j;
      p++;
      if (p == n0)
        for (k++; t < n02; t++, k++) SA[k] = GetI();
    }
  }
}

void build_suf(string& str) {
  int n = str.size();
  int* _s = new int[n + 5];
  int k = 0;
  for (int i = 0; i < n; i++) {
    _s[i] = (int)str[i];
    k = max(k, _s[i]);
  }
  _s[n] = _s[n + 1] = _s[n + 2] = 0;
  int* SA = new int[n + 5];
  suffixArray(_s, SA, n, k);

  s.resize(n + 1);

  int ma = 0;  // TODO
  for (int i = 0; i < n; i++) {
    s[i + 1] = SA[i];
    ma = max(ma, SA[i]);
  }
  s[0] = ma + 1;
}

int main() {
  string str;
  cin >> str;
  build_suf(str);

  int n = str.size();
  for (int i = 0; i <= n; i++) {
    cout << s[i] << " ";
  }
  cout << "\n";
```

```
  return 0;
}
```

1.59 palindromic-tree.cpp

```
struct ptree {
  struct node {
    int length, link;
    char pc = '\0';
    int p = -1;
    map<char, int> to;
    node(int length, int link) : length(length), link(link) {}
  };
  vector<node> nodes;
  int current;

  ptree() : current(1) {
    nodes.push_back(node(-1, 0));
    nodes.push_back(node(0, 0));
  }
  void add(int i, string& s) {
    int parent = nodes[current].length == i ? nodes[current].link : current;
    while (s[i - nodes[parent].length - 1] != s[i]) parent = nodes[parent].
       link;

    if (nodes[parent].to.find(s[i]) != nodes[parent].to.end()) {
      current = nodes[parent].to[s[i]];
    } else {
      int link = nodes[parent].link;
      while (s[i - nodes[link].length - 1] != s[i]) link = nodes[link].link;
      link = max(1, nodes[link].to[s[i]]);
      current = nodes[parent].to[s[i]] = nodes.size();
      nodes.push_back(node(nodes[parent].length + 2, link));
      nodes[current].pc = s[i];
      nodes[current].p = parent;
    }
  }
  void insert(string& s) {
    current = 1;
    for (int i = 0; i < int(s.size()); i++) add(i, s);
  }
};
```

1.60 pbs.cpp

```
s.insert(1);
s.insert(2);
s.insert(4);
s.insert(8);
s.insert(16);

cout << *s.find_by_order(1) << endl;              // 2
cout << *s.find_by_order(2) << endl;              // 4
cout << *s.find_by_order(4) << endl;              // 16
cout << (end(s) == s.find_by_order(6)) << endl;   // true

```

```
cout << s.order_of_key(-5) << endl;   // 0
cout << s.order_of_key(1) << endl;    // 0
cout << s.order_of_key(3) << endl;    // 2
cout << s.order_of_key(4) << endl;    // 2
cout << s.order_of_key(400) << endl;  // 5
```

1.61 pollard-rho.cpp

```
// Complexidades (considerando mul constante):
// rho - esperado O(n^(1/4)) no pior caso
// fact - esperado menos que O(n^(1/4) log(n)) no pior caso

ll mul(ll a, ll b, ll m) {
  ll ret = a * b - ll((long double)1 / m * a * b + 0.5) * m;
  return ret < 0 ? ret + m : ret;
}

ll pow(ll x, ll y, ll m) {
  if (!y) return 1;
  ll ans = pow(mul(x, x, m), y / 2, m);
  return y % 2 ? mul(x, ans, m) : ans;
}

bool prime(ll n) {
  if (n < 2) return 0;
  if (n <= 3) return 1;
  if (n % 2 == 0) return 0;

  ll r = __builtin_ctzll(n - 1), d = n >> r;
  for (int a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
    ll x = pow(a, d, n);
    if (x == 1 or x == n - 1 or a % n == 0) continue;

    for (int j = 0; j < r - 1; j++) {
      x = mul(x, x, n);
      if (x == n - 1) break;
    }
    if (x != n - 1) return 0;
  }
  return 1;
}

ll rho(ll n) {
  if (n == 1 or prime(n)) return n;
  auto f = [n](ll x) { return mul(x, x, n) + 1; };

  ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
  while (t % 40 != 0 or gcd(prd, n) == 1) {
    if (x == y) x = ++x0, y = f(x);
    q = mul(prd, abs(x - y), n);
    if (q != 0) prd = q;
    x = f(x), y = f(f(y)), t++;
  }
  return gcd(prd, n);
}

```

```cpp
vector<ll> fact(ll n) {
  if (n == 1) return {};
  if (prime(n)) return {n};
  ll d = rho(n);
  vector<ll> l = fact(d), r = fact(n / d);
  l.insert(l.end(), all(r));
  return l;
}
```

1.62 prefix-sum2d.cpp

```cpp
int getbit(int lx, int ly, int rx, int ry) {  // entrada - fechado
  rx++;                                       // aberto
  ry++;
  return pref[rx][ry] - pref[rx][ly] - pref[lx][ry] + pref[lx][ly];
}

void build() {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      pref[i + 1][j + 1] =
          pref[i + 1][j] + pref[i][j + 1] - pref[i][j] + mat[i][j];
    }
  }
}
```

1.63 prim.cpp

```cpp
// TODO
ll dist[MAX];
bool vis[MAX];

ll prim(int s = 0) {
  priority_queue<ii, vector<ii>, greater<ii>> p;
  for (auto &x : dist) x = oo;
  for (auto &x : vis) x = false;

  ll retval = 0;
  p.emplace(dist[s] = 0, s);
  while (!p.empty()) {
    int u, val;
    tie(val, u) = p.top();
    p.pop();
    if (vis[u]) continue;
    vis[u] = true;
    retval += dist[u];
    for (auto &[v, cost] : adj[u]) {
      if (!vis[v] && dist[v] > cost) {
        p.emplace(dist[v] = cost, v);
      }
    }
  }
  return retval;
}
```

1.64 reroot.cpp

```cpp
void fill(int u = 0, int p = -1) {
  for (auto &v : adj[u]) {
    if (v == p) continue;
    fill(v, u);
    // TODO
  }
}

void reroot(int new_root, int old_root) {  // TODO
}

void solve(int u = 0, int p = -1) {
  // TODO
  for (auto &v : adj[u]) {
    if (v == p) continue;
    reroot(v, u);
    solve(v, u);
    reroot(u, v);
  }
}
```

1.65 rotate90.cpp

```cpp
void rotate90(vector<vi> &mat) {
  int n = mat.size();
  vector<vi> tmp = mat;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      mat[j][n - i - 1] = tmp[i][j];
    }
  }
}
```

1.66 rotate.cpp

```cpp
// TODO: circular rotation?

int rotateLeft(int x, int d) { return (x << d) | (x >> (32 - d)) }

int rotateRight(int x, int d) { return (x >> d) | (x << (32 - d)) }
```

1.67 scc.cpp

```cpp
vector<vi> adj, adj_rev;
set<int> adj_scc[MAX];
int vis[MAX], raizes[MAX];
vi order, nodos_raiz;
int components[MAX], sz_comp = 0;

void dfs1(int u, vector<vi> &adj) {
  if (vis[u]) return;
  vis[u] = 1;
  for (auto &v : adj[u]) {
    dfs1(v, adj);
  }
  order.push_back(u);
}
```

```cpp

void dfs2(int u, vector<vi> &adj_rev) {
  if (vis[u]) return;
  vis[u] = 1;
  components[sz_comp++] = u;
  for (auto &v : adj_rev[u]) {
    dfs2(v, adj_rev);
  }
}

void scc(vector<vi> &adj) {
  int n = adj.size();
  adj_rev.resize(n);
  for (auto &u : adj_rev) u.clear();
  for (int u = 0; u < n; u++) {
    for (auto &v : adj[u]) {
      adj_rev[v].push_back(u);
    }
  }
  memset(vis, 0, sizeof(vis));
  order.clear();
  for (int u = 0; u < n; u++) {
    dfs1(u, adj);
  }

  memset(vis, 0, sizeof(vis));
  reverse(all(order));
  nodos_raiz.clear();
  for (auto &u : order) {
    if (!vis[u]) {
      sz_comp = 0;
      dfs2(u, adj_rev);
      int raiz = components[0];
      for (int i = 0; i < sz_comp; i++) {
        int v = components[i];
        raizes[v] = raiz;
      }
      nodos_raiz.push_back(raiz);
    }
  }
  for (auto &u : adj_scc) {  // TODO
    u.clear();
  }
  for (int u = 0; u < n; u++) {
    for (auto &v : adj[u]) {
      int _u = raizes[u], _v = raizes[v];
      if (_u != _v) {
        adj_scc[_u].insert(_v);
      }
    }
  }
}
```

1.68 segbeats.cpp

```cpp
// a[i] = max(a[i], x); l <= i <= r
```

```
// a[l] + a[l + 1] + ... + a[r]

struct item {
  bool isValid = false;
  int max2 = -oo, max1 = -oo;
  int cnt = 0;
  ll sum = 0;

  item operator+(item oth) {
    item c = item();
    if (this->max1 > oth.max1) {
      c.max1 = this->max1;
      c.cnt = this->cnt;
      c.max2 = max(this->max2, oth.max1);
    } else if (this->max1 == oth.max1) {
      c.max1 = this->max1;
      c.cnt = this->cnt + oth.cnt;
      c.max2 = max(this->max2, oth.max2);
    } else if (this->max1 < oth.max1) {
      c.max1 = oth.max1;
      c.cnt = oth.cnt;
      c.max2 = max(this->max1, oth.max2);
    }
    c.sum = this->sum + oth.sum;
    return c;
  }
};

item seg[4 * MAX];
int tamseg;

void initseg(int n) {
  tamseg = 1;
  while (tamseg < n) tamseg *= 2;
}

void buildseg(vi &a, int pos, int lx, int rx) {
  if (rx - lx == 1) {
    if (lx < (int)a.size()) {
      seg[pos].max1 = seg[pos].sum = a[lx];
      seg[pos].cnt = 1;
    }
    return;
  }
  int mid = lx + (rx - lx) / 2;
  buildseg(a, 2 * pos + 1, lx, mid);
  buildseg(a, 2 * pos + 2, mid, rx);
  seg[pos] = seg[2 * pos + 1] + seg[2 * pos + 2];
}

void push(int pos, int lx, int rx) {
  if (rx - lx == 1) return;
  if (seg[pos].isValid) {
    int val = seg[pos].max1;
    for (int dx = 1; dx <= 2; dx++) {
```

```
      if (seg[2 * pos + dx].max1 > val) {
        seg[2 * pos + dx].sum -=
            (1LL * (seg[2 * pos + dx].max1 - val)) * seg[2 * pos + dx].cnt;
        seg[2 * pos + dx].max1 = val;
        seg[2 * pos + dx].isValid = true;
      }
    }
    seg[pos].isValid = false;
  }
}

void setseg(int l, int r, int v, int pos, int lx, int rx) {
  push(pos, lx, rx);
  if (lx >= r || rx <= l || seg[pos].max1 <= v) return;
  if (lx >= l && rx <= r && seg[pos].max2 < v) {
    assert(seg[pos].max1 >= v);
    seg[pos].sum -= (1LL * (seg[pos].max1 - v)) * (seg[pos].cnt);
    seg[pos].max1 = v;
    // seg[pos].max2 = -oo;
    seg[pos].isValid = true;
    return;
  }
  int mid = lx + (rx - lx) / 2;
  setseg(l, r, v, 2 * pos + 1, lx, mid);
  setseg(l, r, v, 2 * pos + 2, mid, rx);
  seg[pos] = seg[2 * pos + 1] + seg[2 * pos + 2];
}

ll getseg(int l, int r, int pos, int lx, int rx) {
  push(pos, lx, rx);
  if (lx >= r || rx <= l) return 0;
  if (lx >= l && rx <= r) return seg[pos].sum;
  int mid = lx + (rx - lx) / 2;
  return getseg(l, r, 2 * pos + 1, lx, mid) +
         getseg(l, r, 2 * pos + 2, mid, rx);
}
```

1.69 segment-tree-2d.cpp

```
struct item {
  int e = -1, d = -1, raiz = -1;
  int val = 0;
};

int tamseg = 1 << 19;

// TODO: copiar aqui o codigo da segment_tree_din.cpp

ll getseg(ll y1, ll x1, ll y2, ll x2, int pos, ll lx, ll rx) {
  if (lx >= x2 || rx <= x1) return 0LL;
  if (pos == -1) return 0LL;
  if (lx >= x1 && rx <= x2) {
    return getseg(y1, y2, seg[pos].raiz, 0, tamseg);
  }
  ll mid = lx + (rx - lx) / 2;
  ll x = getseg(y1, x1, y2, x2, seg[pos].e, lx, mid);
```

```
  ll y = getseg(y1, x1, y2, x2, seg[pos].d, mid, rx);
  return x + y;
}

void addseg(ll lin, ll col, int v, int pos, ll lx, ll rx) {
  if (rx - lx == 1) {
    gexist(seg[pos].raiz);
    addseg(lin, v, seg[pos].raiz, 0, tamseg);
    return;
  }
  ll mid = lx + (rx - lx) / 2;
  if (col < mid) {
    gexist(seg[pos].e);
    addseg(lin, col, v, seg[pos].e, lx, mid);
  } else {
    gexist(seg[pos].d);
    addseg(lin, col, v, seg[pos].d, mid, rx);
  }
  gexist(seg[pos].raiz);
  addseg(lin, v, seg[pos].raiz, 0, tamseg);
}
```

1.70 segment-tree-din.cpp

```
struct item {
  int e = -1, d = -1, val = 0;
};

void _merge(item &a, item &b, item &c) { a.val = b.val + c.val; }

int tamseg = 1e9 + 7;

struct Segtree {
  vector<item> seg;
  item neutro;

  int _create() {
    seg.emplace_back();
    return seg.size() - 1;
  }

  Segtree() { _create(); }

  ll getseg(ll l, ll r, int pos, ll lx, ll rx) {
    if (lx >= r || rx <= l) return 0LL;
    if (pos == -1) return 0LL;
    if (lx >= l && rx <= r) {
      return seg[pos].val;
    }
    ll mid = lx + (rx - lx) / 2;
    ll x = getseg(l, r, seg[pos].e, lx, mid);
    ll y = getseg(l, r, seg[pos].d, mid, rx);
    return x + y;
  }

  int get(int l, int r) { return getseg(l, r, 0, -tamseg, tamseg); }
```

```
33
34    void addseg(ll i, int v, int pos, ll lx, ll rx) {
35      if (rx - lx == 1) {
36        seg[pos].val += v;
37        return;
38      }
39      ll mid = lx + (rx - lx) / 2;
40      if (i < mid) {
41        if (seg[pos].e == -1) {
42          int tmp = _create();
43          seg[pos].e = tmp;
44        }
45        addseg(i, v, seg[pos].e, lx, mid);
46      } else {
47        if (seg[pos].d == -1) {
48          int tmp = _create();
49          seg[pos].d = tmp;
50        }
51        addseg(i, v, seg[pos].d, mid, rx);
52      }
53      int e = seg[pos].e;
54      int d = seg[pos].d;
55      _merge(seg[pos], e == -1 ? neutro : seg[e], d == -1 ? neutro : seg[d]);
56    }
57
58    void set(int i, int v) { addseg(i, v, 0, -tamseg, tamseg); }
59  };
```

1.71 segment-tree-din-lazy.cpp

```
1  // TODO: melhorar esse código BGSHOOT
2
3  struct item {
4    int e = -1, d = -1;
5    bool isValid = false;
6    // TODO:
7
8    void merge() {
9      int e = seg[pos].e;
10     int d = seg[pos].d;
11     if (e != -1 && d != -1) {
12       // TODO:
13     } else if (seg[pos].e != -1 || seg[pos].d != -1) {
14       // TODO:
15     }
16   }
17 };
18
19 vector<item> seg;
20 item neutro;
21
22 int _create() {
23   seg.emplace_back();
24   return seg.size() - 1;
25 }
26
```

```
27 void initseg() { _create(); }  // TODO: chamar antes de usar
28
29 void push(int pos, int lx, int rx) {
30   if (rx - lx == 1) return;
31
32   if (seg[pos].isValid) {
33     if (seg[pos].e == -1) {
34       int tmp = _create();
35       seg[pos].e = tmp;
36     }
37     if (seg[pos].d == -1) {
38       int tmp = _create();
39       seg[pos].d = tmp;
40     }
41     // TODO:
42     seg[seg[pos].e].isValid = true;
43
44     // TODO:
45     seg[seg[pos].d].isValid = true;
46
47     seg[pos].isValid = false;
48   }
49 }
50
51 void assign(ll l, ll r, int v, int pos, ll lx, ll rx) {
52   if (lx >= r || rx <= l) return;
53   if (pos == -1) return;
54
55   push(pos, lx, rx);
56
57   if (lx >= l && rx <= r) {
58     seg[pos].dono = v;
59     seg[pos].isValid = true;
60     return;
61   }
62   ll mid = lx + (rx - lx) / 2;
63   if (seg[pos].e == -1 && !(lx >= r || mid <= l)) {
64     int tmp = _create();
65     seg[pos].e = tmp;
66   }
67   if (seg[pos].d == -1 && !(mid >= r || rx <= l)) {
68     int tmp = _create();
69     seg[pos].d = tmp;
70   }
71   assign(l, r, v, seg[pos].e, lx, mid);
72   assign(l, r, v, seg[pos].d, mid, rx);
73   seg[pos].merge();
74 }
```

1.72 segment-tree-lazy.cpp

```
1 struct item {
2   bool isValid = false;
3   // TODO
4   item operator+(item oth) {
5     item c = item();
```

```
    // TODO
    return c;
  }
};

item seg[4 * MAX];

void buildseg(vi &a, int pos, int lx, int rx) {
  if (rx - lx == 1) {
    if (lx < (int)a.size()) {
      // TODO
    }
    return;
  }
  int mid = lx + (rx - lx) / 2;
  buildseg(a, 2 * pos + 1, lx, mid);
  buildseg(a, 2 * pos + 2, mid, rx);
  seg[pos] = seg[2 * pos + 1] + seg[2 * pos + 2];
}

void push(int pos, int lx, int rx) {
  if (rx - lx == 1) return;
  if (seg[pos].isValid) {
    for (int dx = 1; dx <= 2; dx++) {
      // TODO
      seg[2 * pos + dx].isValid = true;
    }
    seg[pos].isValid = false;
  }
}

void setseg(int l, int r, int v, int pos, int lx, int rx) {
  push(pos, lx, rx);
  if (lx >= r || rx <= l) return;
  if (lx >= l && rx <= r) {
    // TODO
    seg[pos].isValid = true;
    return;
  }
  int mid = lx + (rx - lx) / 2;
  setseg(l, r, v, 2 * pos + 1, lx, mid);
  setseg(l, r, v, 2 * pos + 2, mid, rx);
  seg[pos] = seg[2 * pos + 1] + seg[2 * pos + 2];
}

item getseg(int l, int r, int pos, int lx, int rx) {
  push(pos, lx, rx);
  if (lx >= r || rx <= l) return item();
  if (lx >= l && rx <= r) return seg[pos];
  int mid = lx + (rx - lx) / 2;
  return getseg(l, r, 2 * pos + 1, lx, mid) +
         getseg(l, r, 2 * pos + 2, mid, rx);
}
```

1.73 segment-tree-persistente.cpp

```cpp
struct item {
  int e = 0, d = 0, val = 0;
};

void _merge(item &a, item &b, item &c) { a.val = b.val + c.val; }

vector<item> seg;
item neutro;
vi raiz;

int _create() {
  seg.emplace_back();
  return seg.size() - 1;
}

void initseg(int n) {  // TODO: chamar antes de usar
  raiz.resize(n);
  _create();  // neutro
  int tmp = _create();
  raiz[0] = tmp;  // raiz
}

void buildseg(vi &vet, int pos, ll lx, ll rx) {
  if (rx - lx == 1) {
    if (lx < vet.size()) {
      seg[pos].val = vet[lx];
    }
    return;
  }
  int e = _create();
  int d = _create();
  seg[pos].e = e;
  seg[pos].d = d;

  ll mid = lx + (rx - lx) / 2;
  buildseg(vet, seg[pos].e, lx, mid);
  buildseg(vet, seg[pos].d, mid, rx);
  _merge(seg[pos], seg[e], seg[d]);
}

ll getseg(ll l, ll r, int pos, ll lx, ll rx) {
  if (lx >= r || rx <= l) return 0LL;
  if (lx >= l && rx <= r) {
    return seg[pos].val;
  }
  ll mid = lx + (rx - lx) / 2;
  ll x = getseg(l, r, seg[pos].e, lx, mid);
  ll y = getseg(l, r, seg[pos].d, mid, rx);
  return x + y;
}

int addseg(ll i, int v, int pos, ll lx, ll rx) {
  int new_pos = _create();
  seg[new_pos] = seg[pos];

```

```
  if (rx - lx == 1) {
    seg[new_pos].val += v;
    return new_pos;
  }

  ll mid = lx + (rx - lx) / 2;
  if (i < mid) {
    int tmp = addseg(i, v, seg[pos].e, lx, mid);
    seg[new_pos].e = tmp;
  } else {
    int tmp = addseg(i, v, seg[pos].d, mid, rx);
    seg[new_pos].d = tmp;
  }
  int e = seg[new_pos].e;
  int d = seg[new_pos].d;
  _merge(seg[new_pos], seg[e], seg[d]);
  return new_pos;
}

void imprimir(int pos, int ident = 0) {
  if (pos == 0) return;
  for (int i = 0; i < ident; i++) {
    cout << " ";
  }
  cout << seg[pos].val;

  bool is_leaf = (!seg[pos].e && !seg[pos].d);
  if (is_leaf) cout << " <-- ";
  cout << " @" << pos << "\n";
  imprimir(seg[pos].e, ident + 1);
  imprimir(seg[pos].d, ident + 1);
}
```

1.74 segment-tree-sum.cpp

```
struct item {
  item operator+(item oth) {
    item c;
    // TODO
    return c;
  }
};
item seg[4 * MAX];

void buildseg(vi &vet, int pos, int lx, int rx) {
  if (rx - lx == 1) {
    if (lx < vet.size()) {
      // TODO:
    }
    return;
  }
  int mid = lx + (rx - lx) / 2;
  buildseg(vet, 2 * pos + 1, lx, mid);
  buildseg(vet, 2 * pos + 2, mid, rx);
  seg[pos] = seg[2 * pos + 1] + seg[2 * pos + 2];
}
```

```
22
23 void setseg(int i, int v, int pos, int lx, int rx) {
24   if (rx - lx == 1) {
25     // TODO:
26     return;
27   }
28   int mid = lx + (rx - lx) / 2;
29   if (i < mid)
30     setseg(i, v, 2 * pos + 1, lx, mid);
31   else
32     setseg(i, v, 2 * pos + 2, mid, rx);
33   seg[pos] = seg[2 * pos + 1] + seg[2 * pos + 2];
34 }
35
36 item getseg(int l, int r, int pos, int lx, int rx) {
37   if (lx >= r || rx <= l) return item();
38   if (lx >= l && rx <= r) return seg[pos];
39   int mid = lx + (rx - lx) / 2;
40   return getseg(l, r, 2 * pos + 1, lx, mid) +
41          getseg(l, r, 2 * pos + 2, mid, rx);
42 }
```

1.75 segment-tree-sum-it.cpp

```
1  template <typename T>
2  struct segtree {
3    int n;
4    vector<T> seg;
5    segtree(int _n) : n(_n), seg(2 * _n) {}
6    T get(int l, int r) {
7      T left = T(), right = T();
8      for (l += n, r += n; l < r; l /= 2, r /= 2) {
9        if (l & 1) left = left + seg[l++];
10       if (r & 1) right = seg[--r] + right;
11     }
12     return left + right;
13   }
14   void set(int i, T v) {
15     seg[i += n] = v;
16     while (i >>= 1) seg[i] = seg[i * 2] + seg[i * 2 + 1];
17   }
18 };
19 struct item {
20   item operator+(item oth) {}
21 };
```

1.76 sieve.cpp

```
1  vi p(MAXS + 1, 1), d(MAXS + 1), primos;
2  void sieve(int n = MAXS) {
3    p[0] = p[1] = 0;
4    for (int i = 2; i <= n; i++) {
5      if (p[i]) {
6        primos.emplace_back(i);
7        d[i] = i;
8        for (int j = i * i; j <= n; j += i) {
```

```
        p[j] = 0;
        d[j] = i;
      }
    }
  }
}
```

1.77 small-to-large2.cpp

```
set<int>* small(int u, int p = 0) {
  set<int>* now = new set<int>(all(toadd[u]));
  for (auto& v : adj[u]) {
    if (v == p) continue;
    set<int>* oth = small(v, u);
    if (oth->size() > now->size()) swap(oth, now);
    (*now).insert(all((*oth)));
  }
  for (auto i : torem[u]) now->erase(i);
  ans[u] = now->size();
  return now;
}
```

1.78 small-to-large.cpp

```
bool big[MAX];
int sz[MAX];

void dfs_sz(int u, int p = -1) {
  sz[u] = 1;
  for (auto &v : adj[u]) {
    if (v == p) continue;
    dfs_sz(v, u);
    sz[u] += sz[v];
  }
}

void add(int x) {  // TODO:
}

void rem(int x) {  // TODO:
}

void add(int u, int p, int x) {
  if (x == 1)
    add(u);
  else
    rem(u);

  for (auto &v : adj[u]) {
    if (v == p || big[v]) continue;
    add(v, u, x);
  }
}

void small(int u, int p = -1, bool keep = 0) {
  int bigchild = -1, mx = -1;
```

```
  for (auto &v : adj[u])
    if (v != p && mx < sz[v]) mx = sz[v], bigchild = v;
  for (auto &v : adj[u]) {
    if (v == p || v == bigchild) continue;
    small(v, u, 0);
  }
  if (bigchild != -1) small(bigchild, u, 1), big[bigchild] = 1;
  add(u, p, 1);

  // solve

  if (bigchild != -1) big[bigchild] = 0;
  if (keep == 0) add(u, p, -1);
}
```

1.79 sqrt-blocks.cpp

```
// TODO: B
auto f = [&](int l, int r, function<void(int i)> apply_elem,
             function<void(int i)> apply_block) {  // [l, r]  0-indexed
  int bl = l / B, br = r / B;
  if (bl == br) {
    for (int i = l; i <= r; i++) apply_elem(i);
    return;
  }
  for (int i = l; i <= (bl + 1) * B - 1; i++) apply_elem(i);
  for (int bi = bl + 1; bi <= br - 1; bi++) apply_block(bi);
  for (int i = br * B; i <= r; i++) apply_elem(i);
};
```

1.80 sqrt-decomposition-on-trees.cpp

```
int B = ceil(sqrt(h_mx)) + gen(1, 100);
for (int i = 0; i < h_mx; i += B) {
  int mi = -1;
  for (int j = 0; j < B; j++) {
    int id = i + j;
    if (!by_height[id].empty()) {
      if (mi == -1 || (by_height[mi].size() > by_height[id].size())) {
        mi = id;
      }
    }
  }
  if (mi != -1) {
    for (auto &u : by_height[mi]) {
      for (auto &v : by_height[mi]) {
        if (u <= v) {
          tab[ii(u, v)] = solve(u, v);
        }
      }
    }
    for (auto &u : by_height[mi]) {
      special[u] = true;
    }
  }
}
```

1.81 suffix-array.cpp

```cpp
vi s, c, lcp;

void count_sort(vi &s, vi &c) {
  int n = s.size();
  vi cnt(n), pos(n), s_new(n);
  for (auto &x : c) cnt[x]++;

  pos[0] = 0;
  for (int i = 1; i < n; i++) pos[i] = pos[i - 1] + cnt[i - 1];

  for (auto &x : s) {
    s_new[pos[c[x]]] = x;
    ++pos[c[x]];
  }
  s = s_new;
}

void build_suf(string &str) {
  str += "$";
  int n = str.size();
  c.resize(n);
  s.resize(n);

  // Construir r^0
  {
    vector<pair<char, int>> tmp(n);
    for (int i = 0; i < n; i++) {
      tmp[i].first = str[i];
      tmp[i].second = i;
    }

    sort(all(tmp));
    for (int i = 0; i < n; i++) s[i] = tmp[i].second;

    c[s[0]] = 0;
    for (int i = 1; i < n; i++)
      c[s[i]] = c[s[i - 1]] + (int)(tmp[i].first != tmp[i - 1].first);
  }

  int k = 0;
  while ((1 << k) < n) {
    // Ja esta ordenada pela segunda metade
    for (auto &x : s) x = ((x - (1 << k)) % n + n) % n;

    // Ordenar a primeira metade
    count_sort(s, c);

    // Novas classes de equivalência
    vi c_new(n);
    c_new[s[0]] = 0;
    for (int i = 1; i < n; i++) {
      ii prev = {c[s[i - 1]], c[(s[i - 1] + (1 << k)) % n]};
      ii cur = {c[s[i]], c[(s[i] + (1 << k)) % n]};

```

```
      c_new[s[i]] = c_new[s[i - 1]];
      if (cur != prev) c_new[s[i]]++;
    }
    c = c_new;
    ++k;
  }
}

void build_lcp(string &str) {
  int n = str.size();
  lcp.resize(n);
  int k = 0;
  for (int i = 0; i < n - 1; i++) {
    int pi = c[i];
    int prev = s[pi - 1];
    while (str[i + k] == str[prev + k]) k++;
    lcp[pi] = k;
    k = max(k - 1, 0);
  }
}

void echo_suf(string str) {
  int n = str.size();
  for (int i = 0; i < n; i++) {
    cout << "s[" << i << "]: " << s[i] << ", lcp: " << lcp[i] << ", ";
    cout << str.substr(s[i], n - s[i]) << "\n";
  }
}
```

1.82 suffix-automaton.cpp

```
struct node {
  int len, link;
  map<char, int> nxt;
};

node st[MAX * 2];
int sz, last;

void sa_init() {  // TODO: Chamar antes de usar
  st[0].len = 0;
  st[0].link = -1;
  sz = last = 0;
  sz++;
}

void sa_extend(char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  int p = last;
  while (p != -1 && st[p].nxt.find(c) == st[p].nxt.end()) {
    st[p].nxt[c] = cur;
    p = st[p].link;
  }
  if (p == -1) {
    st[cur].link = 0;
```

```
  } else {
    int q = st[p].nxt[c];
    if (st[p].len + 1 == st[q].len) {
      st[cur].link = q;
    } else {
      int clone = sz++;
      st[clone].len = st[p].len + 1;
      st[clone].nxt = st[q].nxt;
      st[clone].link = st[q].link;
      while (p != -1 && st[p].nxt[c] == q) {
        st[p].nxt[c] = clone;
        p = st[p].link;
      }
      st[q].link = st[cur].link = clone;
    }
  }
  last = cur;
}

bool substr(string &str) {
  int q = 0;
  int n = str.size();
  for (int i = 0; i < n; i++) {
    char c = str[i];
    if (st[q].nxt.find(c) == st[q].nxt.end()) {
      return false;
    }
    q = st[q].nxt[c];
  }
  return true;
}
```

1.83 ternaria.cpp

```

double f(double x) {  // TODO
}

double ts(double l, double r) {
  const int MAXIT = 500;  // TODO
  for (int tt = 0; tt < MAXIT; tt++) {
    double dx = (r - l) / 3.0;
    double x1 = l + dx;
    double x2 = r - dx;
    if (f(x1) < f(x2))
      l = x1;
    else
      r = x2;
  }
  return l;  // TODO
}
```

1.84 ternaria-int.cpp

```

int f(int x) {}
```

```
3
4  int ts(int l, int r) {
5    int ans = oo;
6    while (l <= r) {
7      int dx = (r - l) / 3.0;
8      int x1 = l + dx;
9      int x2 = r - dx;
10     int f1 = f(x1);
11     int f2 = f(x2);
12     if (f1 > f2) {
13       ans = f2;
14       l = x1 + 1;
15     } else {
16       ans = f1;
17       r = x2 - 1;
18     }
19   }
20   return ans;
21 }
```

1.85 toposort.cpp

```
1  vi adj[MAX];
2  stack<int> stk;
3  int vis[MAX];
4
5  void dfs(int u) {
6    if (vis[u]) return;
7    vis[u] = 1;
8    for (auto &v : adj[u]) dfs(v);
9    stk.push(u);
10 }
11
12 vi toposort(int n) {
13   memset(vis, 0, sizeof(vis));
14   for (int i = 0; i < n; i++) dfs(i);
15   vi retval;
16   while (!stk.empty()) {
17     retval.push_back(stk.top());
18     stk.pop();
19   }
20   return retval;
21 }
```

1.86 trie-bits.cpp

```
1  const int SZ = 60, K = 2;
2  struct TrieBits {
3    struct node {
4      int nxt[K], term = 0, occ = 0;
5      node() { memset(nxt, -1, sizeof(nxt)); }
6    };
7
8    vector<node> trie;
9
10   TrieBits() { trie.emplace_back(); }
```

```

  void add(int x) {
    int u = 0;
    for (int i = 0; i < SZ; i++) {
      int c = ((1LL << (SZ - i - 1)) & x) > 0;
      if (trie[u].nxt[c] == -1) {
        trie[u].nxt[c] = trie.size();
        trie.emplace_back();
      }
      u = trie[u].nxt[c];
      trie[u].occ++;
    }
    trie[u].term = 1;
  }

  void rem(int x) {
    for (int i = 0, u = 0; i < SZ; i++) {
      int c = ((1LL << (SZ - i - 1)) & x) > 0;
      int prev = u;
      u = trie[u].nxt[c];
      trie[u].occ--;
      if (trie[u].occ == 0) {
        trie[prev].nxt[c] = -1;
      }
    }
  }

  int solve(int x, int u = 0, int h = (SZ - 1)) {
    if (h == -1) return 0;
    int bit = (x & (1LL << h)) > 0;
    if (trie[u].nxt[bit ^ 1] != -1) {
      return solve(x, trie[u].nxt[bit ^ 1], h - 1) | (1LL << h);
    } else if (trie[u].nxt[bit] != -1) {
      return solve(x, trie[u].nxt[bit], h - 1);
    } else {
      return 0;
    }
  }
};
```

1.87 trie.cpp

```
#define to_i(ch) (ch - 'a')
#define K 26
struct node {
  int nxt[K], term = 0;
  node() { memset(nxt, -1, sizeof(nxt)); }
};
vector<node> trie;  // TODO: criar a raiz
void ins(string &w) {
  int n = w.size();
  int u = 0;
  for (int i = 0; i < n; i++) {
    int c = to_i(w[i]);
    if (trie[u].nxt[c] == -1) {
      trie[u].nxt[c] = trie.size();
```

```
      trie.emplace_back();
    }
    u = trie[u].nxt[c];
  }
  trie[u].term = 1;
}
```

1.88 vertex-cover.cpp

```
// Vertex cover
//
// Encontra o tamanho do vertex cover minimo
// Da pra alterar facil pra achar os vertices
// Parece rodar com < 2 s pra N = 90
//
// O(n * 1.38^n)

namespace cover {
const int MAX = 96;
vector<int> g[MAX];
bitset<MAX> bs[MAX];
int n;

void add(int i, int j) {
  if (i == j) return;
  n = max({n, i + 1, j + 1});
  bs[i][j] = bs[j][i] = 1;
}

int rec(bitset<MAX> m) {
  int ans = 0;
  for (int x = 0; x < n; x++)
    if (m[x]) {
      bitset<MAX> comp;
      function<void(int)> dfs = [&](int i) {
        comp[i] = 1, m[i] = 0;
        for (int j : g[i])
          if (m[j]) dfs(j);
      };
      dfs(x);

      int ma, deg = -1, cyc = 1;
      for (int i = 0; i < n; i++)
        if (comp[i]) {
          int d = (bs[i] & comp).count();
          if (d <= 1) cyc = 0;
          if (d > deg) deg = d, ma = i;
        }
      if (deg <= 2) {  // caminho ou ciclo
        ans += (comp.count() + cyc) / 2;
        continue;
      }
      comp[ma] = 0;

      // ou ta no cover, ou nao ta no cover
      ans += min(1 + rec(comp), deg + rec(comp & ~bs[ma]));
```

```
    }
  return ans;
}
int solve() {
  bitset<MAX> m;
  for (int i = 0; i < n; i++) {
    m[i] = 1;
    for (int j = 0; j < n; j++)
      if (bs[i][j]) g[i].push_back(j);
  }
  return rec(m);
}
}  // namespace cover
```

1.89 virtual-tree.cpp

```
// TODO: lca, distancia entre dois vertices com custo (?), tempo de entrada da
// DFS
int virtual_tree(vi vet) {
  auto cmp = [&](int i, int j) { return in[i] < in[j]; };
  sort(all(vet), cmp);
  for (int i = vet.size() - 1; i; i--) vet.push_back(lca(vet[i - 1], vet[i]));
  sort(all(vet), cmp);
  vet.erase(unique(all(vet)), vet.end());
  for (int i = 0; i < vet.size(); i++) virt[vet[i]].clear();
  for (int i = 1; i < vet.size(); i++) {
    int parent = lca(vet[i - 1], vet[i]);
    int d = dist(parent, vet[i]);
    virt[parent].emplace_back(vet[i], d);
  }
  return vet[0];
}
```

1.90 welzl.cpp

```
const double EPS = 1e-15;

struct point {
  double x, y;

  point(double _x, double _y) : x(_x), y(_y) {}

  point() {}
};

struct circle {
  point c;
  double r;

  circle(point _c, double _r) : c(_c), r(_r) {}

  circle() {}
};

double dist(const point &a, const point &b) {
  double dx = a.x - b.x;
```

```
  double dy = a.y - b.y;
  return sqrtl(dx * dx + dy * dy);
}

bool lte(double a, double b) { return b - a >= -EPS; }

bool is_inside(const circle &c, const point &p) {
  return lte(dist(c.c, p), c.r);
}

circle circle_from(const point &A, const point &B) {
  point c = {(A.x + B.x) / 2.0L, (A.y + B.y) / 2.0L};
  return {c, dist(c, A)};
}

// A partir da equação (x - a)^2 + (y - b)^2 = r^2
// Encontrar a, b, resolvendo um sistema linear usandos os pontos dados.
circle circle_from(point &p1, point &p2, point &p3) {
  double a = 2 * (-p1.x + p2.x);
  double b = 2 * (-p1.y + p2.y);
  double c = 2 * (-p2.x + p3.x);
  double d = 2 * (-p2.y + p3.y);
  double r1 = -(p1.x * p1.x - p2.x * p2.x + p1.y * p1.y - p2.y * p2.y);
  double r2 = -(p2.x * p2.x - p3.x * p3.x + p2.y * p2.y - p3.y * p3.y);
  double det = a * d - b * c;
  double cx = (d * r1 - b * r2) / det;
  double cy = (-c * r1 + a * r2) / det;

  point center{cx, cy};
  double radius = dist(center, p1);
  return {center, radius};
}

circle min_circle_trivial(vector<point> &P) {
  if (P.empty()) {
    return {{0, 0}, 0};
  } else if (P.size() == 1) {
    return {P[0], 0};
  } else if (P.size() == 2) {
    return circle_from(P[0], P[1]);
  } else {
    return circle_from(P[0], P[1], P[2]);
  }
}

mt19937 rnd((int)chrono::steady_clock::now().time_since_epoch().count());

int gen(int a, int b) {
  uniform_int_distribution<int> dist(a, b);
  return dist(rnd);
}

circle welzl(vector<point> &pontos, vector<point> r, int n) {
  if (n == 0 || r.size() == 3) {
    return min_circle_trivial(r);
```

```
  }
  // Escolher um ponto aleatoriamente
  int idx = gen(0, n - 1);
  point p = pontos[idx];

  // Coloca no fim e desconsidera
  swap(pontos[idx], pontos[n - 1]);

  circle d = welzl(pontos, r, n - 1);
  if (is_inside(d, p)) {
    return d;
  }
  r.push_back(p);
  return welzl(pontos, r, n - 1);
}

circle welzl(vector<point> &pontos) { return welzl(pontos, {}, pontos.size());
    }
```

1.91 zfunction.cpp

```
vi zfunction(string &s) {
  int n = s.size();
  vi z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

2 Problemas

2.1 area-union-rec.cpp

```
// TODO: simplificar codigo
#define MAX 30005

struct SegLazy {
  struct item {
    bool isValid = false;
    int lazy = 0;
    int mi = 0, qtde = 0;

    item operator+(item oth) {
      item c = item();
      if (this->mi == oth.mi) {
        c.mi = this->mi;
        c.qtde = this->qtde + oth.qtde;
      } else if (this->mi < oth.mi) {
        c.mi = this->mi;
        c.qtde = this->qtde;
      } else {
        c.mi = oth.mi;
        c.qtde = oth.qtde;
```

```
    }
    return c;
  }
};

void merge(item &c, item a, item oth) {
  c.isValid = false;
  c.lazy = 0;
  if (a.mi == oth.mi) {
    c.mi = a.mi;
    c.qtde = a.qtde + oth.qtde;
  } else if (a.mi < oth.mi) {
    c.mi = a.mi;
    c.qtde = a.qtde;
  } else {
    c.mi = oth.mi;
    c.qtde = oth.qtde;
  }
}

item seg[4 * MAX];
int tamseg;

void buildseg(int pos, int lx, int rx) {
  if (rx - lx == 1) {
    seg[pos].isValid = false;
    seg[pos].lazy = 0;
    seg[pos].mi = 0;
    seg[pos].qtde = 1;
    return;
  }
  int mid = lx + (rx - lx) / 2;
  buildseg(2 * pos + 1, lx, mid);
  buildseg(2 * pos + 2, mid, rx);
  merge(seg[pos], seg[2 * pos + 1], seg[2 * pos + 2]);
}

void push(int pos, int lx, int rx) {
  if (rx - lx == 1) return;
  if (seg[pos].isValid) {
    int lazy = seg[pos].lazy;
    for (int dx = 1; dx <= 2; dx++) {
      seg[2 * pos + dx].mi += lazy;
      seg[2 * pos + dx].lazy += lazy;
      seg[2 * pos + dx].isValid = true;
    }
    seg[pos].lazy = 0;
    seg[pos].isValid = false;
  }
}

void setseg(int l, int r, int v, int pos, int lx, int rx) {
  push(pos, lx, rx);
  if (lx >= r || rx <= l) return;
  if (lx >= l && rx <= r) {
```

```
76        seg[pos].mi += v;
77        seg[pos].lazy = v;
78        seg[pos].isValid = true;
79        return;
80      }
81      int mid = lx + (rx - lx) / 2;
82      setseg(l, r, v, 2 * pos + 1, lx, mid);
83      setseg(l, r, v, 2 * pos + 2, mid, rx);
84      merge(seg[pos], seg[2 * pos + 1], seg[2 * pos + 2]);
85    }
86
87    void set(int l, int r, int v) { setseg(l, r, v, 0, 0, tamseg); }
88
89    item getseg(int l, int r, int pos, int lx, int rx) {
90      push(pos, lx, rx);
91      if (lx >= r || rx <= l) return item();
92      if (lx >= l && rx <= r) return seg[pos];
93      int mid = lx + (rx - lx) / 2;
94      return getseg(l, r, 2 * pos + 1, lx, mid) +
95             getseg(l, r, 2 * pos + 2, mid, rx);
96    }
97    int get(int l, int r) {
98      item bah = getseg(l, r, 0, 0, tamseg);
99      int qtde = bah.qtde;
100     int mi = bah.mi;
101     return (r - l) - (mi == 0 ? qtde : 0);
102   }
103   void initseg(int n) {
104     tamseg = 1;
105     while (tamseg < n) tamseg *= 2;
106     buildseg(0, 0, tamseg);
107   }
108 };
109
110 SegLazy seg;
111
112 void add(int l, int r, int v) { seg.set(l, r, v); }
113
114 int get(int l, int r) { return seg.get(l, r); }
115
116 ll solve(vector<iiii> &rec) {
117   int n = rec.size();
118   vector<iii> ev;
119   int i = 0;
120   int maxn = 0;
121   for (auto &[lx, ly, rx, ry] : rec) {
122     ev.emplace_back(lx, 0, i);
123     ev.emplace_back(rx, 1, i);
124     maxn = max(maxn, ry + 1);
125     ++i;
126   }
127   int offset = 10;
128   seg.initseg(maxn + 20);
129   sort(all(ev));
130   int prev = -1e9 - 7;
```

```
  ll ans = 0;
  for (auto &[_, op, id] : ev) {
    auto [lx, ly, rx, ry] = rec[id];
    if (op == 0) {
      if (prev != (-1e9 - 7)) {
        int qtde = get(-5 + offset, maxn + offset);
        ans += ((ll)(lx - prev)) * qtde;
      }
      add(ly + offset, ry + offset, 1);
      prev = lx;
    } else {
      int qtde = get(-5 + offset, maxn + offset);
      ans += ((ll)(rx - prev)) * qtde;
      add(ly + offset, ry + offset, -1);
      prev = rx;
    }
  }
  return ans;
}
```

2.2 dynamic-tree-dp.cpp

```
// Resolve o problema: https://www.acmicpc.net/problem/13515
// DP em árvore com alteração (HLD + Segment tree)

struct item {
  int color, sz, pref, is_empty = 1;
  item operator+(item oth) {
    if (this->is_empty) return oth;
    if (oth.is_empty) return *this;
    return {this->color, this->sz + oth.sz,
            this->sz == this->pref && this->color == oth.color
                ? this->pref + oth.pref
                : this->pref,
            0};
  }
};

// TODO: adicionar código de HLD

segtree<int> seg_color;
segtree<item> seg_dp;
vector<vi> light;
vi color;

void updateDP(int u) {
  seg_dp.set(pos[u],
             {color[u], light[u][color[u]] + 1, light[u][color[u]] + 1, 0});
}
void build(int n) {
  color = vi(n);
  remove_parent();
  fill();
  hld(head[0] = 0);
  seg_color = segtree<int>(n);
  for (int i = 0; i < n; i++) seg_color.set(i, color[sop[i]]);
```

```
35
36    light = vector<vi>(n, vi(2));
37    seg_dp = segtree<item>(n);
38    for (int u = 1; u < n; u++)
39      if (head[u] == u) {
40        light[par[u]][color[u]] += sz[u];
41      }
42    for (int i = 0; i < n; ++i) updateDP(i);
43  }
44
45  void toggle(int u) {
46    bool has_changed = 0;
47    while (head[u] != 0) {
48      light[par[head[u]]][color[head[u]]] -=
49          seg_dp.get(pos[head[u]], pos[tail[u]] + 1).pref;
50      if (!has_changed) {
51        has_changed = 1;
52        color[u] ^= 1;
53        seg_color.set(pos[u], color[u]);
54      }
55      updateDP(u);
56      light[par[head[u]]][color[head[u]]] +=
57          seg_dp.get(pos[head[u]], pos[tail[u]] + 1).pref;
58      u = par[head[u]];
59    }
60    if (!has_changed) {
61      has_changed = 1;
62      color[u] ^= 1;
63      seg_color.set(pos[u], color[u]);
64    }
65    updateDP(u);
66  }
67
68  int solve(int u) {
69    int c = color[u];
70    while (head[u] != 0 && color[par[head[u]]] == c &&
71           (seg_color.get(pos[head[u]], pos[u] + 1) ==
72            (c * (h[u] - h[head[u]] + 1))))
73      u = par[head[u]];
74    int l = 0;
75    int r = h[u] - h[head[u]] + 1;
76    while (r - l > 1) {
77      int mid = l + (r - l) / 2;
78      if (seg_color.get(pos[u] - mid, pos[u] + 1) == (c * (mid + 1)))
79        l = mid;
80      else
81        r = mid;
82    }
83    u = sop[pos[u] - l];
84    return seg_dp.get(pos[u], pos[tail[u]] + 1).pref;
85  }
```

2.3 extremos-inteiros.cpp

```
1  // Devolve a quantidade de coordenadas inteiras entre dois pontos
2  // excluindo os próprios pontos
```

```cpp
int extremos_inteiros(int ax, int ay, int bx, int by){
    if (ax == bx) return abs(ay - by) - 1;
    if (ay == by) return abs(ax - bx) - 1;
    return __gcd(abs(ax - bx), abs(ay - by)) - 1;
}
```

2.4 igual-alternado.cpp

```cpp
vi diff, pref1, pref2;

// tem algum caractere diferente?
bool has_diff(int l, int r) {  // [l, r)
  return (oth[r - 1] - oth[l]) > 0;
}

// todos os caracteres são iguais?
bool is_equal(int l, int r) { // [l, r)
  return !has_diff(l, r);
}

// a string é alternada e.g. "abababb"
bool alternate(int l, int r) {  // [l, r)
  if ((r - l) <= 2) {
    return str[l] != str[l + 1];
  }
  --r;
  --r;
  if ((pref1[r] - pref1[l]) || (pref2[r] - pref2[l])) {
    return false;
  }
  return str[l] != str[l + 1];
}

void build(string str) {  // TODO
  int n = str.size();
  diff = vi(n + 1);
  for (int i = 0; i + 1 < n; i++) {
    diff[i + 1] = diff[i] + (str[i] != str[i + 1]);
  }

  vi even(n);
  for (int i = 0; i + 2 < n; i += 2) {
    even[i] = str[i] != str[i + 2];
  }

  vi odd(n);
  for (int i = 1; i + 2 < n; i += 2) {
    odd[i] = str[i] != str[i + 2];
  }

  pref1 = vi(n + 1);
  pref2 = vi(n + 1);
  partial_sum(all(even), pref1.begin() + 1);
  partial_sum(all(odd), pref2.begin() + 1);
}
```

2.5 knapsack-plus-independent-set-in-tree.cpp

```cpp
// problema da mochila combinado com conjunto independente em uma árvore
const int MAX = 303;
int n, x;
int w[MAX], v[MAX], sz[MAX];
vi adj[MAX];

void hld(int u, int p) {
  adj[u].erase(remove(all(adj[u]), p), adj[u].end());
  sz[u] = 1;
  for (auto &v : adj[u]) {
    hld(v, u);
    sz[u] += sz[v];
  }
  sort(all(adj[u]), [&](int u, int v) { return sz[u] > sz[v]; });
}

pair<vi, vi> dfs(int u, const vi &dp) {
  auto d = dp, e = dp;
  if (!adj[u].empty()) {
    tie(d, e) = dfs(adj[u][0], dp);
    for (int i = 1; i < adj[u].size(); i++) {
      d = dfs(adj[u][i], d).first;
      e = dfs(adj[u][i], e).second;
    }
  }
  for (int i = x; i >= 0; i--) {
    e[i] = d[i];
    if (i >= w[u]) d[i] = max(d[i], e[i - w[u]] + v[u]);
  }
  return {d, e};
}


signed main(){
    cin.tie(0)->sync_with_stdio(0);

    cin >> n >> x;
    for (int i = 0; i < n; i++)
        cin >> w[i] >> v[i];

    for (int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        --u, --v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    hld(0, 0);
    auto [d, ignore] = dfs(0, vi(x + 1));
    cout << d[x] << "\n";
}
```

2.6 kobus.cpp

```cpp
const int MOD = 1e9 + 7;

vi are[1123];
int pd(int u, int p, int k, bool in);

vi memoaux[1123][1123];

int pdaux(int u, int p, int i, int k){
    if (i == are[u].size()){
        if (k == 0) return 1;
        else return 0;
    }
    if (are[u][i] == p){
        return pdaux(u, p, i + 1, k);
    }
    int &resp = memoaux[u][k][i];
    if (resp == -1){
        resp = 0;
        for (int j=0; j <= k; j++){
            resp += pd(are[u][i], u, j, 1) * pdaux(u, p, i + 1, k - j);
            resp %= MOD;
        }
    }
    return resp;
}

int memo[1123][1123][2];

int pd(int u, int p, int k, bool in){
    if (k == 0) return 1;
    if (k < 0) return 0;
    int &resp = memo[u][k][in];
    if (resp == -1){
        resp = 0;
        if (in == 0){
            // caso 0, não preciso começar a subárvore aqui
            // posso não começar
            for (auto &x: are[u]){
                if (x != p){
                    resp += pd(x,u, k, 0);
                }
            }
        }
        // começo
        resp += pdaux(u, p,  0, k - 1);
    }
    return resp;
}

int main(){
    int n, k;
    cin >> n >> k;
    for (int i=0; i < n - 1; i++){
        int a, b;
        cin >> a >> b;
```

```
        a--, b--;
        are[a].push_back(b);
        are[b].push_back(a);
    }
    for (int i=0; i < n; i++){
        for (int j=0; j <= k; j++){
            memo[i][j][0] = memo[i][j][1] = -1;
            memoaux[i][j] = vi(are[i].size(), -1);
        }
    }
    cout << pd(0, 0, k, 0) << "\n";
}
```

2.7 kth-node-path.cpp

```
// TODO: copy lca code
int kth(int u, int v, int k) { // k is 0-indexed
  int d = solve(u, v);  // distance between two nodes
  if (d < k) return -1;
  if (k <= solve(u, lca(u, v))) {
    return goup(u, k);
  } else {
    return goup(v, d - k);
  }
}
```

2.8 max-distance-node-tree.cpp

```
vector<ii> adj[MAX];
int fst[MAX], sd[MAX];
int c[MAX];   // maior caminho considerando a subarvore enraizada em u
int cp[MAX];  // maior caminho indo para o pai
int right_border[MAX]; // auxiliar para recuperar a resposta
int up[MAX]; // auxiliar para recuperar a resposta

int ans[MAX], ansu[MAX]; // ans[v]  := distancia mais longe de v,
                         // ansu[v] := nó que está mais distante

int dfs(int u = 0, int p = -1) {
  fst[u] = sd[u] = -1;
  right_border[u] = u;
  int retval = 0;
  for (auto &[v, _] : adj[u]) {
    if (v == p) continue;
    int r = dfs(v, u);
    retval = max(retval, 1 + r);
    if (fst[u] == -1 || (c[fst[u]] < r)) {
      right_border[u] = right_border[v];
      sd[u] = fst[u];
      fst[u] = v;
    } else if (sd[u] == -1 || c[sd[u]] < r) {
      sd[u] = v;
    }
  }
  c[u] = retval;
  return retval;
```

```
}

void dfs2(int u, int p = -1) {
  ans[u] = c[u];
  cp[u] = 0;
  int goup = -1;
  up[u] = u;
  if (p != -1) {
    if (fst[p] != u) {
      goup = right_border[fst[p]];
      cp[u] = 2 + c[fst[p]];
    } else if (sd[p] != -1) {
      goup = right_border[sd[p]];
      cp[u] = 2 + c[sd[p]];
    }
    if (cp[u] < 1 + cp[p]) {
      goup = up[p];
    }
    up[u] = goup;
    cp[u] = max(cp[u], 1 + cp[p]);
  }
  int go = right_border[u];
  if (ans[u] < cp[u]) {
    go = goup;
  }
  ansu[u] = go;
  ans[u] = max(ans[u], cp[u]);
  for (auto &[v, _] : adj[u]) {
    if (v == p) continue;
    dfs2(v, u);
  }
}
```

2.9 max-xor-over-all-subsets.cpp

```
// Given a set S of size 1 <= n <= 10^5 with elements 0 <= ai < 2^20. What is
// the maximum possible xor of the elements of some subset of S?
const int LOG_A = 64;
int basis[LOG_A];

void insertVector(int mask) {
  for (int i = LOG_A - 1; i >= 0; i--) {
    if ((mask & 1 << i) == 0) continue;

    if (!basis[i]) {
      basis[i] = mask;
      return;
    }
    mask ^= basis[i];
  }
}

int maxXor() {
  int ans = 0;

  for (int i = LOG_A - 1; i >= 0; i--) {
```

```
    if (!basis[i]) continue;

    if (ans & 1 << i) continue;

    ans ^= basis[i];
  }
  return ans;
}
```

2.10 mediana-dinamica.cpp

```
struct median {
  multiset<int> l, r;
  int lsum = 0, rsum = 0;
  void balance() {
    int sz = l.size() + r.size();
    int target = (sz + 1) / 2;
    while (l.size() < target) {
      auto it = r.begin();
      l.emplace(*it);
      lsum += *it;
      rsum -= *it;
      r.erase(it);
    }
    while (l.size() > target) {
      auto it = --l.end();
      lsum -= *it;
      rsum += *it;
      r.emplace(*it);
      l.erase(it);
    }
  }
  void add(int x) {
    if (l.empty() || x < *l.rbegin()) {
      l.emplace(x);
      lsum += x;
    } else {
      rsum += x;
      r.emplace(x);
    }
    balance();
  }
  int get() { return *l.rbegin(); }
  int solve() {
    int med = *l.rbegin();
    return med * l.size() - lsum + rsum - med * r.size();
  }
};
```

2.11 next-greater-element.cpp

```
stack<ii> stk;
vi nxt_greater(vi &vet){
    int n = vet.size();
    vi nxt(n);
    for (int i=0; i < n; i++){
```

```
        while (!stk.empty() && stk.top().first < vet[i]){
            nxt[stk.top().second] = i;
            stk.pop();
        }
        stk.emplace(vet[i], i);
    }
    while (!stk.empty()){
        nxt[stk.top().second] = n;
        stk.pop();
    }
    return nxt;
}
```

2.12 sampling-points-shift.cpp

```
// Um polinômio f(x) de grau menor que N está oculto são dados N valores $f(0), f(1), ... , f(N-1) e dois inteiros c, M.
// Calcule f(c + k) (mod 998244353) para k = 0, 1, ... , M - 1.
vector<mint> sampling_points_shift(const vector<mint>& f, mint c, int m) {
  int n = f.size();
  vector<mint> fat(n + m), a(n + m), b(n + m), s(n + m), res(m);
  fat[0] = 1;
  for (int i = 1; i < n + m; i++) fat[i] = fat[i - 1] * i;
  for (int i = 0; i < n + m; i++) {
    if (i < n) {
      int sign = ((n - 1) - i) & 1 ? -1 : 1;
      a[i] = (sign * f[i]) / (fat[(n - 1) - i] * fat[i]);
    }
    b[i] = ((c - n + i + 1) == 0) ? 1 : 1 / (c - n + i + 1);
  }
  s[0] = 1;
  for (int i = 1; i < n + m; i++)
    s[i] = s[i - 1] * max((int)(c - n + i).val(), (int)1);
  auto retval = convolution(a, b);
  for (int k = 0; k < m; k++) {
    int x = -1;
    if ((c + k).val() < n) {
      x = (c + k).val();
      if (x + modint::mod() < n) continue;// small mod, two zero case
    }
    if (x == -1) res[k] = retval[n + k - 1] * s[n + k] / s[k]; // non-zero case
    else res[k] = f[x]; // zero-case
  }
  return res;
}
```

2.13 taylor-shift.cpp

```
// Um polinômio f(x) = a0 x^0 + a1 x^1 + ... + a_{N-1} x^{N-1} e um inteiro c é dado. Compute uma sequência b0, b1, ... , b_{N-1} satisfazendo f(x + c) = b0 + b1 x^1 + ... + x_{N-1}x^{N-1}, imprima cada elemento modulo 998244353.
// https://judge.yosupo.jp/problem/polynomial_taylor_shift

vector<mint> taylor_shift(vector<mint> &a, int c){
```

```
    int n = a.size();
    vector<mint> fat(n);
    fat[0] = 1;
    for (int i=1; i < n; i++){
        fat[i] = fat[i - 1] * i;
    }
    vector<mint> A(n), B(n);
    for (int i=0; i < n; i++){
        A[i] = a[i] * fat[i];
        B[i] = expbin(c, i) / fat[i];
    }
    reverse(all(A));
    auto retval = convolution(A, B);
    vector<mint> C(n);
    for (int i=0; i < n; i++){
        C[i] = retval[n - i - 1] / fat[i];
    }
    return C;
}
```

3 Essencial

3.1 Template

```cpp
#include <bits/stdc++.h>

#include <ext/pb_ds/assoc_container.hpp> // opcional
#include <ext/pb_ds/detail/standard_policies.hpp> // opcional
#include <ext/pb_ds/tree_policy.hpp>

#define int long long
#define all(x) x.begin(), x.end()

using namespace __gnu_pbds; // opcional
using namespace std;

template <class T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag
    , tree_order_statistics_node_update>; // opcional

template <class T> using ordered_multiset = tree<T, null_type, less_equal<T>,
    rb_tree_tag, tree_order_statistics_node_update>; // opcional

typedef long long ll;
typedef pair<int, int> ii;
typedef tuple<int, int, int> iii;
typedef tuple<int, int, int, int> iiii;
typedef vector<int> vi;
const ll oo = 1987654321987654321;

template <class It>
void db(It b, It e) { // opcional
  for (auto it = b; it != e; it++) cout << *it << ' ';
  cout << endl;
}

signed main() {
    cin.tie(0)->sync_with_stdio(0);

}
```

3.2 Makefile

```
CXX = g++

# fast
CXXFLAGS = -Wall -Wextra -O2 --static -std=c++17

# debug
CXXFLAGS = -fsanitize=address,undefined -fno-omit-frame-pointer -g -Wall -
    Wshadow -std=c++17 -Wno-unused-result -Wno-sign-compare -Wno-char-
    subscripts
```

3.3 stress.sh

```
#!/bin/bash
gen=
sol=
brute=
for ((i=1; ; i++)) do
    ./$gen $i > in
    ./$sol < in > out
    ./$brute < in > out2
    if (! cmp -s out out2) then
        echo "--> entrada:"; cat in
        echo "--> saida obtida:"; cat out
        echo "--> saida esperada:";cat out2
        break;
    fi
    echo $i
done
```

3.4 gen.cpp

```
#include <bits/stdc++.h>
using namespace std;

mt19937 rnd;
int gen(int a, int b) {
  uniform_int_distribution<int> dist(a, b);
  return dist(rnd);
}

signed main(signed argc, char *argv[]) {
  cin.tie(0)->sync_with_stdio(0);
  if (argc < 2) {
    cout << "uso errado\n";
    exit(1);
  }
  rnd = mt19937(atoi(argv[1]));


}
```