Contents

1 Contest

1.1 bash config

```
#copy first argument to clipborad ! ONLY WORK ON
    XORG !
alias clip="xclip -sel clip"

# compile the $1 parameter, if a $2 is provided
# the name will be the the binary output, if
# none is provided the binary name will be
# 'a.out'
comp() {
  echo ">> COMPILING $1 <<" 1>&2

  if [ $# -gt 1 ]; then
    outfile="${2}"
  else
    outfile="a.out"
  fi

  time g++ -std=c++20 \
    -O2 \
    -g3 \
    -Wall \
    -fsanitize=address,undefined \
    -fno-sanitize-recover \
    -D LOCAL \
    -o "${outfile}" \
    "$1"

  if [ $? -ne 0 ]; then
    echo ">> FAILED <<" 1>&2
    return 1
  fi
  echo ">> DONE << " 1>&2
}

# run the binary given in $1, if none is
# given it will try to run the 'a.out'
# binary
run() {
  to_run=./a.out
  if [ -n "$1" ]; then
    to_run="$1"
  fi
  time $to_run
}

# just comp and run your cpp file
# accpets <in1 >out and everything else
comprun() {
  comp "$1" "a" && run ./a ${@:2}
}

testall() {
  comp "$1" generator
  comp "$2" brute
  comp "$3" main

  input_counter=1

  while true; do
    echo "$input_counter"
    run ./generator >input
    run ./main <input >main_output.txt
    run ./brute <input >brute_output.txt

    diff brute_output.txt main_output.txt
    if [ $? -ne 0 ]; then
      echo "Outputs differ at input $input_counter"
      echo "Brute file output:"
      cat brute_output.txt
      echo "Main file output:"
      cat main_output.txt
      echo "input used: "
      cat input
      break
    fi
    ((input_counter++))
  done
}

touch_macro() {
  cat "$1"/template.cpp >>"$2"
  cat "$1"/run.cpp >>"$2"
  cp "$1"/debug.cpp .
}

# Creates a contest with hame $2
# Copies the macro and debug file from $1
# Already creates files a...z .cpp and .py
prepare_contest() {
  mkdir "$2"
  cd "$2"

  for i in {a..z}; do
    touch_macro $1 $i.cpp
  done
}
```

1.2 debug

```
template <typename T>
concept Printable = requires(T t) {
  {
    std::cout << t
  } -> std::same_as<std::ostream &>;
};
template <Printable T>
void __print(const T &x) {
  cerr << x;
}
template <size_t T>
void __print(const bitset<T> &x) {
  cerr << x;
}
template <typename A, typename B>
void __print(const pair<A, B> &p);
template <typename... A>
void __print(const tuple<A...> &t);
template <typename T>
void __print(stack<T> s);
template <typename T>
void __print(queue<T> q);
template <typename T, typename... U>
void __print(priority_queue<T, U...> q);
template <typename A>
void __print(const A &x) {
  bool first = true;
  cerr << '{';
  for (const auto &i : x) {
    cerr << (first ? "" : ","), __print(i);
    first = false;
  }
  cerr << '}';
}
template <typename A, typename B>
void __print(const pair<A, B> &p) {
  cerr << '(';
  __print(p.first);
  cerr << ',';
  __print(p.second);
  cerr << ')';
}
template <typename... A>
void __print(const tuple<A...> &t) {
  bool first = true;
  cerr << '(';
  apply(
      [&first](const auto &...args) {
        ((cerr << (first ? "" : ","),
          __print(args), first = false),
         ...);
      },
      t);
  cerr << ')';
}
template <typename T>
void __print(stack<T> s) {
  vector<T> debugVector;
  while (!s.empty()) {
    T t = s.top();
    debugVector.push_back(t);
    s.pop();
  }
  reverse(debugVector.begin(), debugVector.end());
  __print(debugVector);
}
template <typename T>
void __print(queue<T> q) {
  vector<T> debugVector;
  while (!q.empty()) {
    T t = q.front();
    debugVector.push_back(t);
    q.pop();
  }
  __print(debugVector);
}
template <typename T, typename... U>
void __print(priority_queue<T, U...> q) {
  vector<T> debugVector;
  while (!q.empty()) {
    T t = q.top();
    debugVector.push_back(t);
    q.pop();
  }
  __print(debugVector);
}
void _print() { cerr << "]\n"; }
template <typename Head, typename... Tail>
void _print(const Head &H, const Tail &...T) {
  __print(H);
  if (sizeof...(T)) cerr << ", ";
  _print(T...);
}

#define dbg(x...)                 \
  cerr << "[" << #x << "] = ["; \
  _print(x)
```

1.3 run

```
void run();

int32_t main() {
#ifndef LOCAL
  fastio;
#endif
  int T = 1;

  cin >> T;

  rep(t, 0, T) {
    dbg(t);
```

```
    run();
  }
}
void run() {}
```

1.4 short-template

```
#include <bits/stdc++.h>
using namespace std;
#define fastio                         \
  ios_base::sync_with_stdio(0); \
  cin.tie(0);
void run() {}
int32_t main(void) {
  fastio;
  int t;
  t = 1;
  // cin >> t;
  while (t--) run();
}
```

1.5 template

```
#include <bits/stdc++.h>
using namespace std;
#ifdef LOCAL
#include "debug.cpp"
#else
#define dbg(...)
#endif
#define endl '\n'
#define fastio                         \
  ios_base::sync_with_stdio(0); \
  cin.tie(0);
#define int long long
#define all(j) j.begin(), j.end()
#define rall(j) j.rbegin(), j.rend()
#define len(j) (int)j.size()
#define rep(i, a, b)                                  \
  for (common_type_t<decltype(a), decltype(b)> \
          i = (a);                                      \
       i < (b); i++)
#define rrep(i, a, b)                                 \
  for (common_type_t<decltype(a), decltype(b)> \
          i = (a);                                      \
       i > (b); i--)
#define trav(a,x) for (auto& a : x)
#define pb push_back
#define pf push_front
#define ppb pop_back
#define ppf pop_front
#define eb emplace_back
#define lb lower_bound
#define ub upper_bound
#define emp emplace
#define ins insert
#define divc(a,b) ((a)+(b)-1ll)/(b)
using str = string;
using ll = long long;
using ull = unsigned long long;
using ld = long double;
using vll = vector<ll>;
using pll = pair<ll, ll>;
using vll2d = vector<vll>;
using vi = vector<int>;
using vi2d = vector<vi>;
using pii = pair<int, int>;
using vpii = vector<pii>;
using vc = vector<char>;
using vs = vector<str>;
template <typename T>
using pqmn =
    priority_queue<T, vector<T>, greater<T>>;
template <typename T>
using pqmx = priority_queue<T, vector<T>>;
template <typename T, typename U>
inline bool chmax(T &a, U const &b) {
  return (a < b ? a = b, 1 : 0);
}
template <typename T, typename U>
inline bool chmin(T &a, U const &b) {
  return (a > b ? a = b, 1 : 0);
}
```

1.6 vim config

```
set sta nu rnu sc cindent
set ts=2 sw=2
set bg=dark ruler clipboard=unnamed,unnamedplus,
    timeoutlen=100
colorscheme default
syntax on
" Takes the hash of the selected text and put
" in the vim clipboard
function! HashSelectedText()
    " Yank the selected text to the unnamed
    register
    normal! gvy
    " Use the system() function to call sha256sum
    with the yanked text
    let l:hash = system('echo ' . shellescape(@@) .
     ' | sha256sum')
    " Yank the hash into Vim's unnamed register
    let @" = l:hash
endfunction
```

2 Data Structures

2.1 SQRT decomposition

2.1.1 two-sequence-queries

```
using ll = long long;
const ll MOD = 998244353;
inline ll sum(const ll a, const ll b) {
  return (a + b) % MOD;
}
ll sub(const ll a, const ll b) {
  return (a - b + MOD) % MOD;
}
inline ll mul(const ll a, const ll b) {
  return (a * b) % MOD;
}
struct SqrtDecomposition {
  struct t_sqrt {
    int l, r;
    ll x, y;
    ll prod;
    ll sum_as, sum_bs;
    t_sqrt() {
      l = numeric_limits<int>::max();
      r = numeric_limits<int>::min();
      x = y = prod = sum_as = sum_bs = 0;
    };
  };
  int sqrtLen;
  vector<t_sqrt> blocks;
  vector<ll> as, bs;
  SqrtDecomposition(const vector<ll> &as_,
                    const vector<ll> &bs_) {
    int n = as_.size();
    sqrtLen = (int)sqrt(n + .0) + 1;
    blocks.resize(sqrtLen + 6.66);
    as = as_;
    bs = bs_;
    for (int i = 0; i < n; i++) {
      auto &bi = blocks[i / sqrtLen];
      bi.l = min(bi.l, i);
      bi.r = max(bi.r, i);
      bi.sum_as = sum(bi.sum_as, as[i]);
      bi.sum_bs = sum(bi.sum_bs, bs[i]);
      bi.prod = sum(bi.prod, mul(as[i], bs[i]));
    }
  }
  // adds x to a[i], and y to b[i], in range [l,
  // r]
  void update(int l, int r, ll x, ll y) {
    auto apply1 = [&](int idx, ll x,
                      ll y) -> void {
      auto &block = blocks[idx / sqrtLen];
      block.prod =
          sub(block.prod, mul(as[idx], bs[idx]));
      block.sum_as = sub(block.sum_as, as[idx]);
      block.sum_bs = sub(block.sum_bs, bs[idx]);
      as[idx] = sum(as[idx], x);
      bs[idx] = sum(bs[idx], y);
      block.prod =
          sum(block.prod, as[idx] * bs[idx]);
      block.sum_as = sum(block.sum_as, as[idx]);
      block.sum_bs = sum(block.sum_bs, bs[idx]);
    };
    auto apply2 = [&](int idx, ll x,
                      ll y) -> void {
      blocks[idx].x = sum(blocks[idx].x, x);
      blocks[idx].y = sum(blocks[idx].y, y);
    };
    int cl = l / sqrtLen, cr = r / sqrtLen;
    if (cl == cr) {
      for (int i = l; i <= r; i++) {
        apply1(i, x, y);
      }
    } else {
      for (int i = l; i <= (cl + 1) * sqrtLen - 1;
           i++) {
        apply1(i, x, y);
      }
      for (int i = cl + 1; i <= cr - 1; i++) {
        apply2(i, x, y);
      }
      for (int i = cr * sqrtLen; i <= r; i++) {
        apply1(i, x, y);
      }
    }
```

```
  }
  // sum of a[i]*b[i] in range [l  r]
  ll query(int l, int r) {
    auto eval1 = [&](int idx) -> ll {
      auto &block = blocks[idx / sqrtLen];
      return mul(sum(as[idx], +block.x),
                 sum(bs[idx], block.y));
    };
    auto eval2 = [&](int idx) -> ll {
      auto &block = blocks[idx];
      ll ret = 0;
      ret =
          sum(ret,
              mul(mul(block.x, block.y),
                  sum(sub(block.r, block.l), 1)));
      ret = sum(ret, block.prod);
      ret = sum(ret, block.y * block.sum_as);
      ret = sum(ret, block.x * block.sum_bs);
      return ret;
    };
    ll ret = 0;
    int cl = l / sqrtLen, cr = r / sqrtLen;
    if (cl == cr) {
      for (int i = l; i <= r; i++) {
        ret = sum(ret, eval1(i));
      }
    } else {
      for (int i = l; i <= (cl + 1) * sqrtLen - 1;
           i++) {
        ret = sum(eval1(i), ret);
      }
      for (int i = cl + 1; i <= cr - 1; i++) {
        ret = sum(ret, eval2(i));
      }
      for (int i = cr * sqrtLen; i <= r; i++) {
        ret = sum(ret, eval1(i));
      }
    }
    return ret;
  }
};
```

2.2 Segment tree (dynamic)

2.2.1 Range Max Query Point Max Assignment

Description: Answers range queries in ranges until 10^9 (maybe more)
Time: Query and update $O(n \cdot \log n)$

```
struct node;
node *newNode();
struct node {
  node *left, *right;
  int lv, rv;
  ll val;
  node() : left(NULL), right(NULL), val(-oo) {}
  inline void init(int l, int r) {
    lv = l;
    rv = r;
  }
  inline void extend() {
    if (!left) {
      int m = (lv + rv) / 2;
      left = newNode();
      right = newNode();
      left->init(lv, m);
      right->init(m + 1, rv);
    }
  }
  ll query(int l, int r) {
    if (r < lv || rv < l) {
      return 0;
    }
    if (l <= lv && rv <= r) {
      return val;
    }
    extend();
    return max(left->query(l, r),
               right->query(l, r));
  }
  void update(int p, ll newVal) {
    if (lv == rv) {
      val = max(val, newVal);
      return;
    }
    extend();
    (p <= left->rv ? left : right)
        ->update(p, newVal);
    val = max(left->val, right->val);
  }
};
const int BUFFSZ(1e7);
node *newNode() {
  static int bufSize = BUFFSZ;
  static node buf[(int)BUFFSZ];
  assert(bufSize);
  return &buf[--bufSize];
}
struct SegTree {
  int n;
  node *root;
  SegTree(int _n) : n(_n) {
    root = newNode();
    root->init(0, n);
  }
  ll query(int l, int r) {
    return root->query(l, r);
  }
  void update(int p, ll v) { root->update(p, v); }
};
```

2.2.2 Range Sum Query Point Sum Update

Description: Answers range queries in ranges until 10^9 (maybe more)
Time: Query and update in $O(n \cdot \log n)$

```
struct node;
node *newNode();
struct node {
  node *left, *right;
  int lv, rv;
  ll val;
  node() : left(NULL), right(NULL), val(0) {}
  inline void init(int l, int r) {
    lv = l;
    rv = r;
  }
  inline void extend() {
    if (!left) {
      int m = (rv - lv) / 2 + lv;
      left = newNode();
      right = newNode();
      left->init(lv, m);
      right->init(m + 1, rv);
    }
  }
  ll query(int l, int r) {
    if (r < lv || rv < l) {
      return 0;
    }
    if (l <= lv && rv <= r) {
      return val;
    }
    extend();
    return left->query(l, r) + right->query(l, r);
  }
  void update(int p, ll newVal) {
    if (lv == rv) {
      val += newVal;
      return;
    }
    extend();
    (p <= left->rv ? left : right)
        ->update(p, newVal);
    val = left->val + right->val;
  }
};
const int BUFFSZ(1.3e7);
node *newNode() {
  static int bufSize = BUFFSZ;
  static node buf[(int)BUFFSZ];
  // assert(bufSize);
  return &buf[--bufSize];
}
struct SegTree {
  int n;
  node *root;
  SegTree(int _n) : n(_n) {
    root = newNode();
    root->init(0, n);
  }
  ll query(int l, int r) {
    return root->query(l, r);
  }
  void update(int p, ll v) { root->update(p, v); }
};
```

2.3 Segment tree point update range query

2.3.1 Query GCD (bottom up)

```
using ll = long long;
struct Node {
  ll value;
  bool undef;
  Node()
      : value(1), undef(1){};  // Neutral element
  Node(ll v) : value(v), undef(0){};
};
inline Node combine(const Node &nl,
                    const Node &nr) {
  if (nl.undef) return nr;
  if (nr.undef) return nl;
  Node m;
  m.value = gcd(nl.value, nr.value);
```

```
  m.undef = false;
  return m;
}
template <typename T = Node, auto F = combine>
struct SegTree {
  int n;
  vector<T> st;
  SegTree(int _n) : n(_n), st(n << 1) {}
  void assign(int p, const T &k) {
    for (st[p += n] = k; p >>= 1;)
      st[p] = F(st[p << 1], st[p << 1 | 1]);
  }
  T query(int l, int r) {
    T ansl, ansr;
    for (l += n, r += n + 1; l < r;
         l >>= 1, r >>= 1) {
      if (l & 1) ansl = F(ansl, st[l++]);
      if (r & 1) ansr = F(st[--r], ansr);
    }
    return F(ansl, ansr);
  }
};
```

2.3.2 Query hash (top down)

```
const ll MOD = 1'000'000'009;
const ll P = 31;
const int MAXN = 2'000'000;
ll pows[MAXN + 1];
void computepows() {
  pows[0] = 1;
  for (int i = 1; i <= MAXN; i++) {
    pows[i] = (pows[i - 1] * P) % MOD;
  }
}
struct Node {
  ll hash;
  Node() : hash(-1){};  // Neutral element
  Node(ll v) : hash(v){};
};
inline Node combine(Node &vl, Node &vr, int nl,
                    int nr, int ql, int qr) {
  if (vl.hash == -1) return vr;
  if (vr.hash == -1) return vl;
  Node vm;
  int nm = midpoint(nl, nr);
  int lsize = min(nm, qr) - max(nl, ql) + 1;
  vm.hash = (vl.hash +
             ((vr.hash * pows[lsize]) % MOD)) %
            MOD;
  return vm;
}
template <typename T = Node, auto F = combine>
struct SegTree {
  int n;
  vector<T> st;
  SegTree(int n) : n(n), st(n << 2) {}
  void assign(int p, const T &v) {
    assign(1, 0, n - 1, p, v);
  }
  void assign(int node, int l, int r, int p,
              const T &v) {
    if (l == r) {
      st[node] = v;
      return;
    }
    int m = midpoint(l, r);
    if (p <= m)
      assign(node << 1, l, m, p, v);
    else
      assign(node << 1 | 1, m + 1, r, p, v);
    st[node] = F(st[node << 1], st[node << 1 | 1],
                 l, r, l, r);
  }
  inline T query(int l, int r) {
    return query(1, 0, n - 1, l, r);
  }
  inline T query(int node, int nl, int nr, int l,
                 int r) const {
    if (r < nl or nr < l) return T();
    if (l <= nl and nr <= r) return st[node];
    int m = midpoint(nl, nr);
    auto a = query(node << 1, nl, m, l, r);
    auto b =
        query(node << 1 | 1, m + 1, nr, l, r);
    return F(a, b, nl, nr, l, r);
  }
};
```

2.3.3 Query max subarray sum (bottom up)

```
struct Node {
  ll tot, suf, pref, best;
  // Neutral element
  Node()
      : tot(-oo),
        suf(-oo),
        pref(-oo),
        best(-oo) {}  // Neutral element
  // for assign
  Node(ll x) {
    tot = x, suf = x, pref = x,
    best = max(0ll, x);
  }
};
Node combine(Node &nl, Node &nr) {
  if (nl.tot == -oo) return nr;
  if (nr.tot == -oo) return nl;
  Node m;
  m.tot = nl.tot + nr.tot;
  m.pref = max({nl.pref, nl.tot + nr.pref});
  m.suf = max({nr.suf, nr.tot + nl.suf});
  m.best =
      max({nl.best, nr.best, nl.suf + nr.pref});
  return m;
}
```

2.3.4 Query min (bottom up)

```
struct Node {
  ll value;
  Node()
      : value(numeric_limits<
              ll>::max()){};  // Neutral element
  Node(ll v) : value(v){};
};
Node combine(Node &l, Node &r) {
  Node m;
  m.value = min(l.value, r.value);
  return m;
}
template <typename T = Node, auto F = combine>
struct SegTree {
  int n;
  vector<T> st;
  SegTree(int _n) : n(_n), st(n << 1) {}
  void assign(int p, const T &k) {
    for (st[p += n] = k; p >>= 1;)
      st[p] = F(st[p << 1], st[p << 1 | 1]);
  }
  T query(int l, int r) {
    T ansl = T(), ansr = T();
    for (l += n, r += n + 1; l < r;
         l >>= 1, r >>= 1) {
      if (l & 1) ansl = F(ansl, st[l++]);
      if (r & 1) ansr = F(st[--r], ansr);
    }
    return F(ansl, ansr);
  }
};
```

2.3.5 Query sum (bottom up)

```
struct Node {
  ll value;
  Node() : value(0){};  // Neutral element
  Node(ll v) : value(v){};
};
inline Node combine(const Node &nl,
                    const Node &nr) {
  Node m;
  m.value = nl.value + nr.value;
  return m;
}
struct SegTree {
  int n;
  vector<Node> st;
  SegTree(int _n) : n(_n), st(n << 1) {}
  void assign(int p, const Node &k) {
    for (st[p += n] = k; p >>= 1;)
      st[p] = combine(st[p << 1], st[p << 1 | 1]);
  }
  Node query(int l, int r) {
    Node ansl = Node(), ansr = Node();
    for (l += n, r += n + 1; l < r;
         l >>= 1, r >>= 1) {
      if (l & 1) ansl = combine(ansl, st[l++]);
      if (r & 1) ansr = combine(st[--r], ansr);
    }
    return combine(ansl, ansr);
  }
};
```

2.4 Segment tree range update range query

2.4.1 Arithmetic progression sum update, query sum

Description: Makes arithmetic progression updates in range and sum queries.
Usage: Considering $PA(A, R) = [A + R, A + 2R, A + 3R, ...]$
- **update_set(l, r, A, R):** sets [l, r] to PA(A, R)

- **update_add(l, r, A, R):** sum PA(A, R) in [l, r]
- **query(l, r):** sum in range [l, r]

Time: build $O(N)$, updates and queries $O(logN)$

```
const ll oo = 1e18;
struct SegTree {
  struct Data {
    ll sum;
    ll set_a, set_r, add_a, add_r;
    Data()
        : sum(0),
          set_a(oo),
          set_r(0),
          add_a(0),
          add_r(0) {}
  };
  int n;
  vector<Data> seg;
  SegTree(int n_)
      : n(n_), seg(vector<Data>(4 * n)) {}
  void prop(int p, int l, int r) {
    int sz = r - l + 1;
    ll &sum = seg[p].sum, &set_a = seg[p].set_a,
       &set_r = seg[p].set_r,
       &add_a = seg[p].add_a,
       &add_r = seg[p].add_r;
    if (set_a != oo) {
      set_a += add_a, set_r += add_r;
      sum =
          set_a * sz + set_r * sz * (sz + 1) / 2;
      if (l != r) {
        int m = (l + r) / 2;
        seg[2 * p].set_a = set_a;
        seg[2 * p].set_r = set_r;
        seg[2 * p].add_a = seg[2 * p].add_r = 0;
        seg[2 * p + 1].set_a =
            set_a + set_r * (m - l + 1);
        seg[2 * p + 1].set_r = set_r;
        seg[2 * p + 1].add_a =
            seg[2 * p + 1].add_r = 0;
      }
      set_a = oo, set_r = 0;
      add_a = add_r = 0;
    } else if (add_a or add_r) {
      sum +=
          add_a * sz + add_r * sz * (sz + 1) / 2;
      if (l != r) {
        int m = (l + r) / 2;
        seg[2 * p].add_a += add_a;
        seg[2 * p].add_r += add_r;
        seg[2 * p + 1].add_a +=
            add_a + add_r * (m - l + 1);
        seg[2 * p + 1].add_r += add_r;
      }
      add_a = add_r = 0;
    }
  }
  int inter(pii a, pii b) {
    if (a.first > b.first) swap(a, b);
    return max(
        0, min(a.second, b.second) - b.first + 1);
  }
  ll set(int a, int b, ll aa, ll rr, int p, int l,
         int r) {
    prop(p, l, r);
    if (b < l or r < a) return seg[p].sum;
    if (a <= l and r <= b) {
      seg[p].set_a = aa;
      seg[p].set_r = rr;
      prop(p, l, r);
      return seg[p].sum;
    }
    int m = (l + r) / 2;
    int tam_l = inter({l, m}, {a, b});
    return seg[p].sum =
               set(a, b, aa, rr, 2 * p, l, m) +
               set(a, b, aa + rr * tam_l, rr,
                   2 * p + 1, m + 1, r);
  }
  void update_set(int l, int r, ll aa, ll rr) {
    set(l, r, aa, rr, 1, 0, n - 1);
  }
  ll add(int a, int b, ll aa, ll rr, int p, int l,
         int r) {
    prop(p, l, r);
    if (b < l or r < a) return seg[p].sum;
    if (a <= l and r <= b) {
      seg[p].add_a += aa;
      seg[p].add_r += rr;
      prop(p, l, r);
      return seg[p].sum;
    }
    int m = (l + r) / 2;
    int tam_l = inter({l, m}, {a, b});
    return seg[p].sum =
               add(a, b, aa, rr, 2 * p, l, m) +
               add(a, b, aa + rr * tam_l, rr,
                   2 * p + 1, m + 1, r);
  }
  void update_add(int l, int r, ll aa, ll rr) {
    add(l, r, aa, rr, 1, 0, n - 1);
  }
  ll query(int a, int b, int p, int l, int r) {
    prop(p, l, r);
    if (b < l or r < a) return 0;
    if (a <= l and r <= b) return seg[p].sum;
    int m = (l + r) / 2;
    return query(a, b, 2 * p, l, m) +
           query(a, b, 2 * p + 1, m + 1, r);
  }
  ll query(int l, int r) {
    return query(l, r, 1, 0, n - 1);
  }
};
```

2.4.2 Increment update query min & max (bottom up)

```
using SegT = ll;
struct QueryT {
  SegT mx, mn;
  QueryT()
      : mx(numeric_limits<SegT>::min()),
        mn(numeric_limits<SegT>::max()) {}
  QueryT(SegT _v) : mx(_v), mn(_v) {}
};
inline QueryT combine(QueryT ln, QueryT rn,
                      pii lr1, pii lr2) {
  chmax(ln.mx, rn.mx);
  chmin(ln.mn, rn.mn);
  return ln;
}
using LazyT = SegT;
inline QueryT applyLazyInQuery(QueryT q, LazyT l,
                               pii lr) {
  if (q.mx == QueryT().mx) q.mx = SegT();
  if (q.mn == QueryT().mn) q.mn = SegT();
  q.mx += l, q.mn += l;
  return q;
}
inline LazyT applyLazyInLazy(LazyT a, LazyT b) {
  return a + b;
}
using UpdateT = SegT;
inline QueryT applyUpdateInQuery(QueryT q,
                                 UpdateT u,
                                 pii lr) {
  if (q.mx == QueryT().mx) q.mx = SegT();
  if (q.mn == QueryT().mn) q.mn = SegT();
  q.mx += u, q.mn += u;
  return q;
}
inline LazyT applyUpdateInLazy(LazyT l, UpdateT u,
                               pii lr) {
  return l + u;
}
template <typename Qt = QueryT,
          typename Lt = LazyT,
          typename Ut = UpdateT, auto C = combine,
          auto ALQ = applyLazyInQuery,
          auto ALL = applyLazyInLazy,
          auto AUQ = applyUpdateInQuery,
          auto AUL = applyUpdateInLazy>
struct LazySegmentTree {
  int n, h;
  vector<Qt> ts;
  vector<Lt> ds;
  vector<pii> lrs;
  LazySegmentTree(int _n)
      : n(_n),
        h(sizeof(int) * 8 - __builtin_clz(n)),
        ts(n << 1),
        ds(n),
        lrs(n << 1) {
    rep(i, 0, n) lrs[i + n] = {i, i};
    rrep(i, n - 1, 0) {
      lrs[i] = {lrs[i << 1].first,
                lrs[i << 1 | 1].second};
    }
  }
  LazySegmentTree(const vector<Qt> &xs)
      : LazySegmentTree(len(xs)) {
    copy(all(xs), ts.begin() + n);
    rep(i, 0, n) lrs[i + n] = {i, i};
    rrep(i, n - 1, 0) {
      ts[i] = C(ts[i << 1], ts[i << 1 | 1],
                lrs[i << 1], lrs[i << 1 | 1]);
    }
  }
  void set(int p, Qt v) {
    ts[p + n] = v;
    build(p + n);
  }
  void upd(int l, int r, Ut v) {
    l += n, r += n + 1;
    int l0 = l, r0 = r;
    for (; l < r; l >>= 1, r >>= 1) {
```

```
    if (l & 1) apply(l++, v);
    if (r & 1) apply(--r, v);
  }
  build(l0), build(r0 - 1);
}
Qt qry(int l, int r) {
  l += n, r += n + 1;
  push(l), push(r - 1);
  Qt resl = Qt(), resr = Qt();
  pii lr1 = {l, l}, lr2 = {r, r};
  for (; l < r; l >>= 1, r >>= 1) {
    if (l & 1)
      resl = C(resl, ts[l], lr1, lrs[l]), l++;
    if (r & 1)
      r--, resr = C(ts[r], resr, lrs[r], lr2);
  }
  return C(resl, resr, lr1, lr2);
}
void build(int p) {
  while (p > 1) {
    p >>= 1;
    ts[p] = ALQ(C(ts[p << 1], ts[p << 1 | 1],
                  lrs[p << 1], lrs[p << 1 | 1]),
                ds[p], lrs[p]);
  }
}
void push(int p) {
  rrep(s, h, 0) {
    int i = p >> s;
    if (ds[i] != Lt()) {
      apply(i << 1, ds[i]),
          apply(i << 1 | 1, ds[i]);
      ds[i] = Lt();
    }
  }
}
inline void apply(int p, Ut v) {
  ts[p] = AUQ(ts[p], v, lrs[p]);
  if (p < n) ds[p] = AUL(ds[p], v, lrs[p]);
}
};
```

2.4.3 Increment update sum query (top down)

```
struct Lnode {
  ll v;
  bool assign;
  Lnode() : v(), assign() {}  // Neutral element
  Lnode(ll _v, bool a = 0) : v(_v), assign(a){};
};
using Qnode = ll;
using Unode = Lnode;
struct LSegTree {
  int n, ql, qr;
  vector<Qnode> st;
  vector<Lnode> lz;
  /*-----------------------------------------*/
  Qnode merge(Qnode lv, Qnode rv, int nl,
              int nr) {
    return lv + rv;
  }
  void prop(int i, int l, int r) {
    if (lz[i].assign) {
      st[i] = lz[i].v * (r - l + 1);
      if (l != r) lz[tol(i)] = lz[tor(i)] = lz[i];
    } else {
      st[i] += lz[i].v * (r - l + 1);
      if (l != r)
        lz[tol(i)].v += lz[i].v,
            lz[tor(i)].v += lz[i].v;
    }
    lz[i] = Lnode();
  }
  void applyV(int i, Unode v) {
    if (v.assign) {
      lz[i] = v;
    } else {
      lz[i].v += v.v;
    }
  }
  /*-----------------------------------------*/
  LSegTree() {}
  LSegTree(int _n)
      : n(_n), st(_n << 2), lz(_n << 2) {}
  bool disjoint(int l, int r) {
    return qr < l or r < ql;
  }
  bool contains(int l, int r) {
    return ql <= l and r <= qr;
  }
  int tol(int i) { return i << 1; }
  int tor(int i) { return i << 1 | 1; }
  void build(vector<Qnode> &v) {
    build(v, 1, 0, n - 1);
  }
  void build(vector<Qnode> &v, int i, int l,
             int r) {
    if (l == r) {
      st[i] = v[l];
      return;
    }
    int m = midpoint(l, r);
    build(v, tol(i), l, m);
    build(v, tor(i), m + 1, r);
    st[i] = merge(st[tol(i)], st[tor(i)], l, r);
  }
  void upd(int l, int r, Unode v) {
    ql = l, qr = r;
    upd(1, 0, n - 1, v);
  }
  void upd(int i, int l, int r, Unode v) {
    prop(i, l, r);
    if (disjoint(l, r)) return;
    if (contains(l, r)) {
      applyV(i, v);
      prop(i, l, r);
      return;
    }
    int m = midpoint(l, r);
    upd(tol(i), l, m, v);
    upd(tor(i), m + 1, r, v);
    st[i] = merge(st[tol(i)], st[tor(i)], l, r);
  }
  Qnode qry(int l, int r) {
    ql = l, qr = r;
    return qry(1, 0, n - 1);
  }
  Qnode qry(int i, int l, int r) {
    prop(i, l, r);
    if (disjoint(l, r)) return Qnode();
    if (contains(l, r)) return st[i];
    int m = midpoint(l, r);
    return merge(qry(tol(i), l, m),
                 qry(tor(i), m + 1, r), l, r);
  }
};
```

2.5 Bitree 2D

Description: Given a 2D array you can increment an arbitrary position, and also query the subsum of a subgrid
Time: Update and query in $O(logN^2)$

```
struct Bit2d {
  int n;
  vll2d bit;
  Bit2d(int ni) : n(ni), bit(n + 1, vll(n + 1)) {}
  Bit2d(int ni, vll2d &xs)
      : n(ni), bit(n + 1, vll(n + 1)) {
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        update(i, j, xs[i][j]);
      }
    }
  }
  void update(int x, int y, ll val) {
    for (; x <= n; x += (x & (-x))) {
      for (int i = y; i <= n; i += (i & (-i))) {
        bit[x][i] += val;
      }
    }
  }
  ll sum(int x, int y) {
    ll ans = 0;
    for (int i = x; i; i -= (i & (-i))) {
      for (int j = y; j; j -= (j & (-j))) {
        ans += bit[i][j];
      }
    }
    return ans;
  }
  ll query(int x1, int y1, int x2, int y2) {
    return sum(x2, y2) - sum(x2, y1 - 1) -
           sum(x1 - 1, y2) + sum(x1 - 1, y1 - 1);
  }
};
```

2.6 Convex Hull Trick / Line Container

Description: Container where you can add lines of the form $mx + b$, and query the maximum value at point x.
Usage: $insert\_line(m, b)$ inserts the line $m \cdot x + b$ in the container.
$eval(x)$ find the highest value among all lines in the point x.
Time: Eval and insert in $O(\log N)$

```
const ll LLINF = 1e18;
const ll is_query = -LLINF;
struct Line {
  ll m, b;
  mutable function<const Line *()> succ;
  bool operator<(const Line &rhs) const {
    if (rhs.b != is_query) return m < rhs.m;
    const Line *s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
struct Cht : public multiset<Line> {  // maintain
                                      // max m*x+b
```

```
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end())
      return y->m == x->m && y->b <= x->b;
    return (ld)(x->b - y->b) * (z->m - y->m) >=
           (ld)(y->b - z->b) * (y->m - x->m);
  }
  void insert_line(
      ll m,
      ll b) {  // min -> insert (-m,-b) -> -eval()
    auto y = insert({m, b});
    y->succ = [=] {
      return next(y) == end() ? 0 : &*next(y);
    };
    if (bad(y)) {
      erase(y);
      return;
    }
    while (next(y) != end() && bad(next(y)))
      erase(next(y));
    while (y != begin() && bad(prev(y)))
      erase(prev(y));
  }
  ll eval(ll x) {
    auto l = *lower_bound((Line){x, is_query});
    return l.m * x + l.b;
  }
};
```

2.7 DSU / UFDS

Usage: You may discomment the commented parts to find online which nodes belong to each set, it makes the *union_set* method cost $O(log^2)$ instead $O(A)$

```
struct DSU {
  vi ps, sz;
  // vector<unordered_set<int>> sts;
  DSU(int N)
      : ps(N + 1),
        sz(N, 1) /*, sts(N) */
  {
    iota(ps.begin(), ps.end(), 0);
    // for (int i = 0; i < N; i++)
    // sts[i].insert(i);
  }
  int find_set(int x) {
    return ps[x] == x ? x
                      : ps[x] = find_set(ps[x]);
  }
  int size(int u) { return sz[find_set(u)]; }
  bool same_set(int x, int y) {
    return find_set(x) == find_set(y);
  }
  void union_set(int x, int y) {
    if (same_set(x, y)) return;
    int px = find_set(x);
    int py = find_set(y);
    if (sz[px] < sz[py]) swap(px, py);
    ps[py] = px;
    sz[px] += sz[py];
    // sts[px].merge(sts[py]);
  }
};
```

2.8 Lichao Tree (dynamic)

Description: Lichao Tree that creates the nodes dynamically, allowing to query and update from range $[MAXL, MAXR]$
Usage:

- $query(x)$: find the highest point among all lines in the structure
- $add(a, b)$: add a line of form $y = ax + b$ in the structure
- $addSegment(a, b, l, r)$: add a line segment of form $y = ax + b$ which covers from range $[l, r]$

Time: $O(\log N)$

```
template <typename T = ll, T MAXL = 0,
          T MAXR = 1 '000' 000'001>
struct LiChaoTree {
  static const T inf =
      -numeric_limits<T>::max() / 2;
  bool first_best(T a, T b) { return a > b; }
  T get_best(T a, T b) {
    return first_best(a, b) ? a : b;
  }
  struct line {
    T m, b;
    T operator()(T x) { return m * x + b; }
  };
  struct node {
    line li;
    node *left, *right;
    node(line _li = {0, inf})
        : li(_li),
          left(nullptr),
          right(nullptr) {}
    ~node() {
      delete left;
      delete right;
    }
  };
  node *root;
  LiChaoTree(line li = {0, inf})
      : root(new node(li)) {}
  ~LiChaoTree() { delete root; }
  T query(T x, node *cur, T l, T r) {
    if (cur == nullptr) return inf;
    if (x < l or x > r) return inf;
    T mid = midpoint(l, r);
    T ans = cur->li(x);
    ans = get_best(ans,
                   query(x, cur->left, l, mid));
    ans = get_best(
        ans, query(x, cur->right, mid + 1, r));
    return ans;
  }
  T query(T x) {
    return query(x, root, MAXL, MAXR);
  }
  void add(line li, node *&cur, T l, T r) {
    if (cur == nullptr) {
      cur = new node(li);
      return;
    }
    T mid = midpoint(l, r);
    if (first_best(li(mid), cur->li(mid)))
      swap(li, cur->li);
    if (first_best(li(l), cur->li(l)))
      add(li, cur->left, l, mid);
    if (first_best(li(r), cur->li(r)))
      add(li, cur->right, mid + 1, r);
  }
  void add(T m, T b) {
    add({m, b}, root, MAXL, MAXR);
  }
  void addSegment(line li, node *&cur, T l, T r,
                  T lseg, T rseg) {
    if (r < lseg || l > rseg) return;
    if (cur == nullptr) cur = new node;
    if (lseg <= l && r <= rseg) {
      add(li, cur, l, r);
      return;
    }
    T mid = midpoint(l, r);
    if (l != r) {
      addSegment(li, cur->left, l, mid, lseg,
                 rseg);
      addSegment(li, cur->right, mid + 1, r, lseg,
                 rseg);
    }
  }
  void addSegment(T a, T b, T l, T r) {
    addSegment({a, b}, root, MAXL, MAXR, l, r);
  }
};
```

2.9 Merge sort tree

Description: Like a segment tree but each node stores the ordered subsegment it represents.
Usage:

- $inrange(l, r, a, b)$: counts the number of positions i, $l \leq i \leq r$ such that $a \leq x_i \leq b$.

Time: Build $O(N \log N^2)$, inrange $O(\log N^2)$
Memory: $O(n \log N)$

```
template <class T>
struct MergeSortTree {
  int n;
  vector<vector<T>> st;
  MergeSortTree(vector<T> &xs)
      : n(len(xs)), st(n << 1) {
    rep(i, 0, n) st[i + n] = vector<T>({xs[i]});
    rrep(i, n - 1, 0) {
      st[i].resize(len(st[i << 1]) +
                   len(st[i << 1 | 1]));
      merge(all(st[i << 1]), all(st[i << 1 | 1]),
            st[i].begin());
    }
  }
  int count(int i, T a, T b) {
    return upper_bound(all(st[i]), b) -
           lower_bound(all(st[i]), a);
  }
  int inrange(int l, int r, T a, T b) {
    int ans = 0;
    for (l += n, r += n + 1; l < r;
         l >>= 1, r >>= 1) {
      if (l & 1) ans += count(l++, a, b);
      if (r & 1) ans += count(--r, a, b);
    }
    return ans;
  }
};
```

2.10 Mex with update

Description: This DS allows you to mantain an array of elments, insert, and remove, and query the MEX at any time.
Usage:

- $Mex(mxsz)$: Initialize the DS, $mxsz$ must be the maximum number of elements that the structure may have.
- $add(x)$: just adds one copy of x.
- $rmv(x)$: just remove a copy of x.
- $operator()$: returns the MEX.

Time:

- $Mex(mxsz)$: $O(\log mxsz)$
- $add(x)$: $O(\log mxsz)$
- $rmv(x)$: $O(\log mxsz)$
- $operator()$: $O(1)$

```
struct Mex {
  int mx_sz;
  vi hs;
  set<int> st;
  Mex(int _mx_sz) : mx_sz(_mx_sz), hs(mx_sz + 1) {
    auto it = st.begin();
    rep(i, 0, mx_sz + 1) it = st.insert(it, i);
  }
  void add(int x) {
    if (x > mx_sz) return;
    if (!hs[x]++) st.erase(x);
  }
  void rmv(int x) {
    if (x > mx_sz) return;
    if (!--hs[x]) st.emplace(x);
  }
  int operator()() const { return *st.begin(); }
  /*
    Optional, you can just create with size
    len(xs) add N elements :D
  */
  Mex(const vi &xs, int _mx_sz = -1)
      : Mex(~_mx_sz ? _mx_sz : len(xs)) {
    for (auto xi : xs) add(xi);
  }
};
```

2.11 Orderd Set (GNU PBDS)

Usage: If you need an ordered **multi** set you may add an id to each value. Using greater_equal, or less_equal is considered undefined behavior.

- **order_of_key (k)**: Number of items strictly smaller/greater than k .
- **find_by_order(k)**: K-th element in a set (counting from zero).

Time: Both $O(\log N)$
Warning: Is 2 or 3 times slower then a regular set/map.

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
using ordered_set =
    tree<T, null_type, less<T>, rb_tree_tag,
         tree_order_statistics_node_update>;
```

2.12 Prefix Sum 2D

Description: Given an 2D array with N lines and M columns, find the sum of the subarray that have the left upper corner at $(x1, y1)$ and right bottom corner at $(x2, y2)$.
Time: Build $O(N \cdot M)$, Query $O(1)$.

```
template <typename T>
struct psum2d {
  vector<vector<T>> s;
  vector<vector<T>> psum;
  psum2d(vector<vector<T>> &grid, int n, int m)
      : s(n + 1, vector<T>(m + 1)),
        psum(n + 1, vector<T>(m + 1)) {
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= m; j++) {
        s[i][j] =
            s[i][j - 1] + grid[i - 1][j - 1];
        psum[i][j] = psum[i - 1][j] + s[i][j];
      }
  }
  T query(int x1, int y1, int x2, int y2) {
    T ans = psum[x2 + 1][y2 + 1] + psum[x1][y1];
    ans -= psum[x2 + 1][y1] + psum[x1][y2 + 1];
    return ans;
  }
};
```

2.13 Venice Set

Description: A container that you can insert q copies of element e, increment every element in the contaiver in x, query which is the best element and it's quantity and also remove k copies of the greatest element.
Time:

- add elment $O(\log N)$
- remove $O(\log N)$
- update: $O(1)$
- query $O(1)$

```
template <typename T = ll>
struct VeniceSet {
  using T2 = pair<T, ll>;
  priority_queue<T2, vector<T2>, greater<T2>> pq;
  T acc;
  VeniceSet() : acc() {}
  void add_element(const T &e, const ll q) {
    pq.emplace(e - acc, q);
  }
  void update_all(const T &x) { acc += x; }
  T2 best() {
    auto ret = pq.top();
    ret.first += acc;
    return ret;
  }
  void pop() { pq.pop(); }
  void pop_k(int k) {
    auto [e, q] = pq.top();
    pq.pop();
    q -= k;
    if (q) pq.emplace(e, q);
  }
};
```

2.14 Wavelet tree

```
using ll = long long;
template <typename T>
struct WaveletTree {
  struct Node {
    T lo, hi;
    int left_child, right_child;
    vector<int> pcnt;
    vector<ll> psum;
    Node(int lo_, int hi_)
        : lo(lo_),
          hi(hi_),
          left_child(0),
          right_child(0),
          pcnt(),
          psum() {}
  };
  vector<Node> nodes;
  WaveletTree(vector<T> v) {
    nodes.reserve(2 * v.size());
    auto [mn, mx] =
        minmax_element(v.begin(), v.end());
    auto build = [&](auto &&self, Node &node,
                     auto from, auto to) {
      if (node.lo == node.hi or from >= to)
        return;
      auto mid = midpoint(node.lo, node.hi);
      auto f = [&mid](T x) { return x <= mid; };
      node.pcnt.reserve(to - from + 1);
      node.pcnt.push_back(0);
      node.psum.reserve(to - from + 1);
      node.psum.push_back(0);
      T left_upper = node.lo,
        right_lower = node.hi;
      for (auto it = from; it != to; it++) {
        auto value = f(*it);
        node.pcnt.push_back(node.pcnt.back() +
                            value);
        node.psum.push_back(node.psum.back() +
                            *it);
        if (value)
          left_upper = max(left_upper, *it);
        else
          right_lower = min(right_lower, *it);
      }
      auto pivot = stable_partition(from, to, f);
      node.left_child =
          make_node(node.lo, left_upper);
      self(self, nodes[node.left_child], from,
           pivot);
      node.right_child =
          make_node(right_lower, node.hi);
      self(self, nodes[node.right_child], pivot,
           to);
    };
    build(build, nodes[make_node(*mn, *mx)],
          v.begin(), v.end());
  }
  T kth_element(int L, int R, int K) const {
    auto f = [&](auto &&self, const Node &node,
                 int l, int r, int k) -> T {
      if (l > r) return 0;
      if (node.lo == node.hi) return node.lo;
      int lb = node.pcnt[l],
          rb = node.pcnt[r + 1],
          left_size = rb - lb;
      return (left_size > k
                  ? self(self,
                         nodes[node.left_child],
                         lb, rb - 1, k)
                  : self(self,
                         nodes[node.right_child],
```

```
                          l - lb, r - rb,
                          k - left_size));
    };
    return f(f, nodes[0], L, R, K);
  }
  pair<int, ll> count_and_sum_in_range(
      int L, int R, T a, T b) const {
    auto f = [&](auto &&self, const Node &node,
                 int l, int r) -> pair<int, ll> {
      if (l > r or node.lo > b or node.hi < a)
        return {0, 0};
      if (a <= node.lo and node.hi <= b)
        return {r - l + 1,
                (node.lo == node.hi
                     ? (r - l + 1ll) * node.lo
                     : node.psum[r + 1] -
                           node.psum[l])};
      int lb = node.pcnt[l],
          rb = node.pcnt[r + 1];
      auto [left_cnt, left_sum] =
          self(self, nodes[node.left_child], lb,
               rb - 1);
      auto [right_cnt, right_sum] =
          self(self, nodes[node.right_child],
               l - lb, r - rb);
      return {left_cnt + right_cnt,
              left_sum + right_sum};
    };
    return f(f, nodes[0], L, R);
  }
  inline int count_in_range(int L, int R, T a,
                            T b) const {
    return count_and_sum_in_range(L, R, a, b)
        .first;
  }
  inline ll sum_in_range(int L, int R, T a,
                         T b) const {
    return count_and_sum_in_range(L, R, a, b)
        .second;
  }
 private:
  int make_node(T lo, T hi) {
    int id = (int)nodes.size();
    nodes.emplace_back(lo, hi);
    return id;
  }
};
```

3 Dynamic Programming

3.1 Binary Knapsack (bottom up)

Description: Given the points each element have, and it repespective cost, computes the maximum points we can get if we can ignore/choose an element, in such way that the sum of costs don't exceed the maximum cost allowed.
Time: $O(N * W)$
Warning: The vectors VS and WS starts at one, so it need an empty value at index 0.

```
const int MAXN(1 '000), MAXCOST(1' 000 * 20);
ll dp[MAXN + 1][MAXCOST + 1];
bool ps[MAXN + 1][MAXCOST + 1];
pair<ll, vi> knapsack(const vll &points,
                      const vi &costs,
                      int maxCost) {
  int n = len(points) -
          1;  // ELEMENTS START AT INDEX 1 !
  for (int m = 0; m <= maxCost; m++) {
    dp[0][m] = 0;
  }
  for (int i = 1; i <= n; i++) {
    dp[i][0] = dp[i - 1][0] +
               (costs[i] == 0) * points[i];
    ps[i][0] = costs[i] == 0;
  }
  for (int i = 1; i <= n; i++) {
    for (int m = 1; m <= maxCost; m++) {
      dp[i][m] = dp[i - 1][m], ps[i][m] = 0;
      int w = costs[i];
      ll v = points[i];
      if (w <= m and
          dp[i - 1][m - w] + v > dp[i][m]) {
        dp[i][m] = dp[i - 1][m - w] + v,
        ps[i][m] = 1;
      }
    }
  }
  vi is;
  for (int i = n, m = maxCost; i >= 1; --i) {
    if (ps[i][m]) {
      is.emplace_back(i);
      m -= costs[i];
    }
  }
  return {dp[n][maxCost], is};
}
```

3.2 Edit Distance

Time: $O(N * M)$

```
int edit_distance(const string &a,
                  const string &b) {
  int n = a.size();
  int m = b.size();
  vector<vi> dp(n + 1, vi(m + 1, 0));
  int ADD = 1, DEL = 1, CHG = 1;
  for (int i = 0; i <= n; ++i) {
    dp[i][0] = i * DEL;
  }
  for (int i = 1; i <= m; ++i) {
    dp[0][i] = ADD * i;
  }
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
      int add = dp[i][j - 1] + ADD;
      int del = dp[i - 1][j] + DEL;
      int chg =
          dp[i - 1][j - 1] +
          (a[i - 1] == b[j - 1] ? 0 : 1) * CHG;
      dp[i][j] = min({add, del, chg});
    }
  }
  return dp[n][m];
}
```

3.3 Knapsack

Description: Finds the maximum score you can achieve, given that you have N items, each item has a *cost*, a *point* and a *quantity*, you can spent at most *maxcost* and buy each item the maximum quantity it has.
Time: $O(n \cdot maxcost \cdot \log maxqtd)$
Memory: $O(maxcost)$.

```
ll knapsack(const vi &weight, const vll &value,
            const vi &qtd, int maxCost) {
  vi costs;
  vll values;
  for (int i = 0; i < len(weight); i++) {
    ll q = qtd[i];
    for (ll x = 1; x <= q; q -= x, x <<= 1) {
      costs.eb(x * weight[i]);
      values.eb(x * value[i]);
    }
    if (q) {
      costs.eb(q * weight[i]);
      values.eb(q * value[i]);
    }
  }
  vll dp(maxCost + 1);
  for (int i = 0; i < len(values); i++) {
    for (int j = maxCost; j > 0; j--) {
      if (j >= costs[i])
        dp[j] = max(dp[j],
                    values[i] + dp[j - costs[i]]);
    }
  }
  return dp[maxCost];
}
```

3.4 Longest Increasing Subsequence

Description: Find the pair (sz, psx) where sz is the size of the longest subsequence and psx is a vector where psx_i tells the size of the longest increase subsequence that ends at position i. get_idx just tells which indices could be in the longest increasing subsequence.
Time: $O(n \log n)$

```
template <typename T>
pair<int, vi> lis(const vector<T> &xs, int n) {
  vector<T> dp(n + 1, numeric_limits<T>::max());
  dp[0] = numeric_limits<T>::min();
  int sz = 0;
  vi psx(n);
  rep(i, 0, n) {
    int pos =
        lower_bound(all(dp), xs[i]) - dp.begin();
    sz = max(sz, pos);
    dp[pos] = xs[i];
    psx[i] = pos;
  }
  return {sz, psx};
}
template <typename T>
vi get_idx(vector<T> xs) {
  int n = xs.size();
  auto [sz1, psx1] = lis(xs, n);
  transform(rall(xs), xs.begin(),
            [](T x) { return -x; });
  auto [sz2, psx2] = lis(xs, n);
  vi ans;
```

```
  rep(i, 0, n) {
    int l = psx1[i];
    int r = psx2[n - i - 1];
    if (l + r - 1 == sz1) ans.eb(i);
  }
  return ans;
}
```

3.5 Monery sum

Description: Find every possible sum using the given values only once.

```
set<int> money_sum(const vi &xs) {
  using vc = vector<char>;
  using vvc = vector<vc>;
  int _m = accumulate(all(xs), 0);
  int _n = xs.size();
  vvc _dp(_n + 1, vc(_m + 1, 0));
  set<int> _ans;
  _dp[0][xs[0]] = 1;
  for (int i = 1; i < _n; ++i) {
    for (int j = 0; j <= _m; ++j) {
      if (j == 0 or _dp[i - 1][j]) {
        _dp[i][j + xs[i]] = 1;
        _dp[i][j] = 1;
      }
    }
  }
  for (int i = 0; i < _n; ++i)
    for (int j = 0; j <= _m; ++j)
      if (_dp[i][j]) _ans.insert(j);
  return _ans;
}
```

3.6 Travelling Salesman Problem

Time: $O(N^2 \cdot 2^N)$
Memory: $O(N^2 \cdot 2^N)$

```
vll2d dist;
vll memo;
int tsp(int i, int mask, int N) {
  if (mask == (1 << N) - 1) return dist[i][0];
  if (memo[i][mask] != -1) return memo[i][mask];
  int ans = INT_MAX << 1;
  for (int j = 0; j < N; ++j) {
    if (mask & (1 << j)) continue;
    auto t =
        tsp(j, mask | (1 << j), N) + dist[i][j];
    ans = min(ans, t);
  }
  return memo[i][mask] = ans;
}
```

4 Extras

4.1 Binary to gray

```
string binToGray(string bin) {
  string gray(bin.size(), '0');
  int n = bin.size() - 1;
  gray[0] = bin[0];
  for (int i = 1; i <= n; i++) {
    gray[i] = '0' + (bin[i - 1] == '1') ^
              (bin[i] == '1');
  }
  return gray;
}
```

4.2 Get permutation cycles

Description: Receives a permutation [0, n-1] and return a vector 2D with each cycle.

```
vll2d getPermutationCicles(const vll &ps) {
  ll n = len(ps);
  vector<char> visited(n);
  vector<vll> cicles;
  rep(i, 0, n) {
    if (visited[i]) continue;
    vll cicle;
    ll pos = i;
    while (!visited[pos]) {
      cicle.pb(pos);
      visited[pos] = true;
      pos = ps[pos];
    }
    cicles.push_back(vll(all(cicle)));
  }
  return cicles;
}
```

4.3 Max & Min Check

Description: Returns the min/max value in range [l, r] that satisfies the lambda function check, if there is no such value the max/min possible value for that type will be returned.
Time: $O(\log l - r + 1)$

```
template <typename T>
T maxCheck(T l, T r, function<bool(T)> check) {
  T best = numeric_limits<T>::min();
  while (l <= r) {
    T m = midpoint(l, r);
    if (check(m))
      chmax(best, m), l = m + 1;
    else
      r = m - 1;
  }
  return best;
}
template <typename T>
T minCheck(T l, T r, function<bool(T)> check) {
  T best = numeric_limits<T>::max();
  while (l <= r) {
    T m = midpoint(l, r);
    if (check(m))
      chmin(best, m), r = m - 1;
    else
      l = m + 1;
  }
  return best;
}
```

4.4 Mo's algorithm

```
template <typename T, typename Tans>
struct Mo {
  struct Query {
    int l, r, idx, block;
    Query(int _l, int _r, int _idx, int _block)
        : l(_l),
          r(_r),
          idx(_idx),
          block(_block) {}
    bool operator<(const Query &q) const {
      if (block != q.block)
        return block < q.block;
      return (block & 1 ? (r < q.r) : (r > q.r));
    }
  };
  vector<T> vs;
  vector<Query> qs;
  const int block_size;
  Mo(const vector<T> &a)
      : vs(a),
        block_size((int)ceil(sqrt(a.size()))) {}
  void add_query(int l, int r) {
    qs.emplace_back(l, r, qs.size(),
                    l / block_size);
  }
  auto solve() {
    // get answer return type
    vector<Tans> answers(qs.size());
    sort(all(qs));
    int cur_l = 0, cur_r = -1;
    for (auto q : qs) {
      while (cur_l > q.l) add(--cur_l);
      while (cur_r < q.r) add(++cur_r);
      while (cur_l < q.l) remove(cur_l++);
      while (cur_r > q.r) remove(cur_r--);
      answers[q.idx] = get_answer();
    }
    return answers;
  }
 private:
  // add value at idx from data structure
  inline void add(int idx) {}
  // remove value at idx from data structure
  inline void remove(int idx) {}
  // extract current answer of the data structure
  inline Tans get_answer() {}
};
```

4.5 __int128t stream

```
void print(__int128 x) {
  if (x < 0) {
    cout << '-';
    x = -x;
  }
  if (x > 9) print(x / 10);
  cout << (char)((x % 10) + '0');
}
__int128 read() {
  string s;
  cin >> s;
  __int128 x = 0;
  for (auto c : s) {
```

```
    if (c != '-') x += c - '0';
    x *= 10;
  }
  x /= 10;
  if (s[0] == '-') x = -x;
  return x;
}
```

5 Geometry

5.1 Check if a point belong to line segment

```
// Verifica se o ponto P pertence ao segmento de
// reta AB
const ld EPS = 1e-9;
template <typename T>
struct Point {
  T x, y;
  Point(T _x, T _y) : x(_x), y(_y) {}
};
template <typename T>
bool equals(const T a, const T b) {
  if (is_floating_point<T>) {
    return fabsl(a - b) <= EPS;
  }
  return a == b;
}
/*
   Verify if the segment AB contains point P
 */
template <typename T>
bool contains(const Point<T> &A,
              const Point<T> &B,
              const Point<T> &P) {
  auto xmin = min(A.x, B.x);
  auto xmax = max(A.x, B.x);
  auto ymin = min(A.y, B.y);
  auto ymax = max(A.y, B.y);
  if (P.x < xmin || P.x > xmax || P.y < ymin ||
      P.y > ymax)
    return false;
  return equals((P.y - A.y) * (B.x - A.x),
                (P.x - A.x) * (B.y - A.y));
}
```

5.2 Check if point is inside triangle

```
struct point {
  int x, y;
  int id;
  point operator-(const point &o) const {
    return {x - o.x, y - o.y};
  }
  int operator^(const point &o) const {
    return x * o.y - y * o.x;
  }
};
/*
    Verify the direction that the point
  _e_ is in relation to the vector
  formed by the points a->b
  -1 = right
  0 = collinear
  1 = left
 */
int ccw(point a, point b, point e) {
  int tmp = (b - a) ^ (e - a);
  return (tmp > 0) - (tmp < 0);
}
/*
  Verify if the point e
  is inside the triangle formed by
  the points t1, t2, t3
 */
bool inside_triangle(point t1, point t2, point t3,
                     point e) {
  int x = ccw(t1, t2, e);
  int y = ccw(t2, t3, e);
  int z = ccw(t3, t1, e);
  return !((x == 1 or y == 1 or z == 1) and
           (x == -1 or y == -1 or z == -1));
}
```

5.3 Convex hull

```
struct pt {
  double x, y;
  int id;
};
int orientation(pt a, pt b, pt c) {
  double v = a.x * (b.y - c.y) +
             b.x * (c.y - a.y) +
             c.x * (a.y - b.y);
  if (v < 0) return -1;  // clockwise
  if (v > 0) return +1;  // counter-clockwise
  return 0;
}
bool cw(pt a, pt b, pt c,
        bool include_collinear) {
  int o = orientation(a, b, c);
  return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) {
  return orientation(a, b, c) == 0;
}
void convex_hull(vector<pt> &pts,
                 bool include_collinear = false) {
  pt p0 = *min_element(all(pts), [](pt a, pt b) {
    return make_pair(a.y, a.x) <
           make_pair(b.y, b.x);
  });
  sort(all(pts), [&p0](const pt &a, const pt &b) {
    int o = orientation(p0, a, b);
    if (o == 0)
      return (p0.x - a.x) * (p0.x - a.x) +
                 (p0.y - a.y) * (p0.y - a.y) <
             (p0.x - b.x) * (p0.x - b.x) +
                 (p0.y - b.y) * (p0.y - b.y);
    return o < 0;
  });
  if (include_collinear) {
    int i = len(pts) - 1;
    while (i >= 0 &&
           collinear(p0, pts[i], pts.back()))
      i--;
    reverse(pts.begin() + i + 1, pts.end());
  }
  vector<pt> st;
  for (int i = 0; i < len(pts); i++) {
    while (st.size() > 1 &&
           !cw(st[len(st) - 2], st.back(), pts[i],
               include_collinear))
      st.pop_back();
    st.push_back(pts[i]);
  }
  pts = st;
}
```

5.4 Polygon lattice points

```
ll cross(ll x1, ll y1, ll x2, ll y2) {
  return x1 * y2 - x2 * y1;
}
ll polygonArea(vector<pll> &pts) {
  ll ats = 0;
  for (int i = 2; i < len(pts); i++)
    ats +=
        cross(pts[i].first - pts[0].first,
              pts[i].second - pts[0].second,
              pts[i - 1].first - pts[0].first,
              pts[i - 1].second - pts[0].second);
  return abs(ats / 2ll);
}
ll boundary(vector<pll> &pts) {
  ll ats = pts.size();
  for (int i = 0; i < len(pts); i++) {
    ll deltax = (pts[i].first -
                 pts[(i + 1) % pts.size()].first);
    ll deltay =
        (pts[i].second -
         pts[(i + 1) % pts.size()].second);
    ats += abs(__gcd(deltax, deltay)) - 1;
  }
  return ats;
}
pll latticePoints(vector<pll> &pts) {
  ll bounds = boundary(pts);
  ll area = polygonArea(pts);
  ll inside = area + 1ll - bounds / 2ll;
  return {inside, bounds};
}
```

5.5 Segment intersection

```
using ld = long double;
template <typename T = ld>
struct Point {
  T x, y;
  bool is_port;
};
template <typename T = ld>
bool operator==(const Point<T> &a,
                const Point<T> &b) {
  return a.x == b.x and a.y == b.y;
}
template <typename T = ld>
struct Segment {
  Point<T> p1, p2;
};
```

```cpp
template <typename T>
int orientation(Point<T> p, Point<T> q,
                Point<T> r) {
  int val = (q.y - p.y) * (r.x - q.x) -
            (q.x - p.x) * (r.y - q.y);
  // TODO: if it's a float must use other way to
  // compare
  if (val == 0)
    return 0;  // colinear
  else if (val > 0)
    return 1;  // clockwise
  else
    return 2;  // counterclockwise
}
template <typename T>
bool do_segment_intersect(Segment<T> s1,
                          Segment<T> s2) {
  int o1 = orientation(s1.p1, s1.p2, s2.p1);
  int o2 = orientation(s1.p1, s1.p2, s2.p2);

  int o3 = orientation(s2.p1, s2.p2, s1.p1);
  int o4 = orientation(s2.p1, s2.p2, s1.p2);

  return (o1 != o2 and o3 != o4) or
         (o1 == 0 and o3 == 0) or
         (o2 == 0 and o4 == 0);
}
```

6 Graphs

6.1 Heavy-Light Decomposition (point update)

6.1.1 Maximum number on path

```cpp
struct Node {
  ll value;
  Node()
      : value(numeric_limits<
              ll>::min()){};  // Neutral
                              // element
  Node(ll v) : value(v){};
};
Node combine(Node l, Node r) {
  Node m;
  m.value = max(l.value, r.value);
  return m;
}
template <typename T = Node, auto F = combine>
struct SegTree {
  int n;
  vector<T> st;
  SegTree(int _n) : n(_n), st(n << 1) {}

  void set(int p, const T &k) {
    for (st[p += n] = k; p >>= 1;)
      st[p] = F(st[p << 1], st[p << 1 | 1]);
  }

  T query(int l, int r) {
    T ansl, ansr;
    for (l += n, r += n + 1; l < r;
         l >>= 1, r >>= 1) {
      if (l & 1) ansl = F(ansl, st[l++]);
      if (r & 1) ansr = F(st[--r], ansr);
    }
    return F(ansl, ansr);
  }
};
template <typename SegT = Node,
          auto SegOp = combine>
struct HeavyLightDecomposition {
  int n;
  vi ps, ds, sz, heavy, head, pos;
  SegTree<SegT, SegOp> seg;

  HeavyLightDecomposition(const vi2d &g,
                          const vector<SegT> &v,
                          int root = 0)
      : n(len(g)), seg(n) {
    ps = ds = sz = heavy = head = pos = vi(n, -1);

    auto dfs = [&](auto &&self, int u) -> void {
      sz[u] = 1;
      int mx = 0;
      for (auto x : g[u])
        if (x != ps[u]) {
          ps[x] = u;
          ds[x] = ds[u] + 1;
          self(self, x);
          sz[u] += sz[x];
          if (sz[x] > mx)
            mx = sz[x], heavy[u] = x;
        }
    };

    dfs(dfs, root);

    for (int i = 0, cur = 0; i < n; i++) {
      if (ps[i] == -1 or heavy[ps[i]] != i)
        for (int j = i; j != -1; j = heavy[j]) {
          head[j] = i;
          pos[j] = cur++;
        }
    }

    rep(i, 0, n) seg.set(pos[i], v[i]);
  }

  vector<pii> disjoint_ranges(int u, int v) {
    vector<pii> ret;
    for (; head[u] != head[v]; v = ps[head[v]]) {
      if (ds[head[u]] > ds[head[v]]) swap(u, v);

      ret.eb(pos[head[v]], pos[v]);
    }
    if (ds[u] > ds[v]) swap(u, v);
    ret.eb(pos[u], pos[v]);
    return ret;
  }

  SegT query_path(int u, int v) {
    SegT res;
    for (auto [l, r] : disjoint_ranges(u, v)) {
      res = SegOp(res, seg.query(l, r));
    }
    return res;
  }

  SegT query_subtree(int u) const {
    return seg.query(pos[u], pos[u] + sz[u] - 1);
  }

  void set(int u, SegT x) { seg.set(pos[u], x); }
};
```

6.2 2-SAT

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

Usage: Negated variables are represented by bit-inversions ($\tilde{x}$).

Returns true iff it is solvable ts.values[0..N-1] holds the assigned values to the vars.

Time: $O(N + E)$, where N is the number of boolean variables, and E is the number of clauses.

```cpp
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values;  // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2 * n) {}

  int addVar() {  // (optional)
    gr.eb();
    gr.eb();
    return N++;
  }

  void either(int f, int j) {
    f = max(2 * f, -1 - 2 * f);
    j = max(2 * j, -1 - 2 * j);
    gr[f].pb(j ^ 1);
    gr[j].pb(f ^ 1);
  }

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

  void atMostOne(const vi &li) {  // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i, 2, sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z;
  int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x;
    z.pb(i);
    for (int e : gr[i])
      if (!comp[e])
        low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back();
        z.pop_back();
        comp[x] = low;
        if (values[x >> 1] == -1)
          values[x >> 1] = x & 1;
      } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2 * N, 0);
    comp = val;
    rep(i, 0, 2 * N) if (!comp[i]) dfs(i);
    rep(i, 0, N) if (comp[2 * i] ==
                     comp[2 * i + 1]) return 0;
    return 1;
  }
};
```

6.3 BFS-01

Description: Similar to a Dijkstra given a weighted graph finds the distance from source s to every other node.
Time: $O(V + E)$
Warning: Applicable only when the weight of the edges $\in \{0, x\}$

```
vector<pair<ll, int>> adj[maxn];
ll dists[maxn];
int s, n;
void bfs_01() {
  fill(dists, dists + n, oo);
  dist[s] = 0;
  deque<int> q;
  q.emplace_back(s);
  while (not q.empty()) {
    auto u = q.front();
    q.pop_front();
    for (auto [v, w] : adj[u]) {
      if (dist[v] <= dist[u] + w) continue;
      dist[v] = dist[u] + w;
      w ? q.emplace_back(v) : q.emplace_front(v);
    }
  }
}
```

6.4 Bellman ford

Description: Find shortest path from a single source to all other nodes. Can detect negative cycles.
Time: $O(V \cdot E)$

```
bool bellman_ford(
    const vector<vector<pair<int, ll>>> &g, int s,
    vector<ll> &dist) {
  int n = (int)g.size();
  dist.assign(n, LLONG_MAX);
  vector<int> count(n);
  vector<char> in_queue(n);
  queue<int> q;
  dist[s] = 0;
  q.push(s);
  in_queue[s] = true;
  while (not q.empty()) {
    int cur = q.front();
    q.pop();
    in_queue[cur] = false;
    for (auto [to, w] : g[cur]) {
      if (dist[cur] + w < dist[to]) {
        dist[to] = dist[cur] + w;
        if (not in_queue[to]) {
          q.push(to);
          in_queue[to] = true;
          count[to]++;
          if (count[to] > n) return false;
        }
      }
    }
  }
  return true;
}
```

6.5 Bellman-Ford (find negative cycle)

Description: Given a directed graph find a negative cycle by running n iterations, and if the last one produces a relaxation than there is a cycle.
Time: $O(V \cdot E)$

```
const ll oo = 2500 * 1e9;
using graph = vector<vector<pair<int, ll>>>;
vi negative_cycle(graph &g, int n) {
  vll d(n, oo);
  vi p(n, -1);
  int x = -1;
  d[0] = 0;
  for (int i = 0; i < n; i++) {
    x = -1;
    for (int u = 0; u < n; u++) {
      for (auto &[v, l] : g[u]) {
        if (d[u] + l < d[v]) {
          d[v] = d[u] + l;
          p[v] = u;
          x = v;
        }
      }
    }
  }
  if (x == -1)
    return {};
  else {
    for (int i = 0; i < n; i++) x = p[x];
    vi cycle;
    for (int v = x;; v = p[v]) {
      cycle.eb(v);
      if (v == x and len(cycle) > 1) break;
    }
    reverse(all(cycle));
    return cycle;
  }
}
```

6.6 Biconnected Components

Description: Build a vector of vectors, where the i-th vector correspond to the nodes of the i-th, biconnected component, a biconnected component is a subset of nodes and edges in which there is no cut point, also exist at least two distinct routes in vertex between any two vertex in the same biconnected component.
Time: $O(N + M)$

```
const int maxn(5'00'000);
int tin[maxn], stck[maxn], bcc_cnt, n, top = 0,
                                       timer = 1;
vector<int> g[maxn], nodes[maxn];
int tarjan(int u, int p = -1) {
  int lowu = tin[u] = timer++;
  int son_cnt = 0;
  stck[++top] = u;
  for (auto v : g[u]) {
    if (!tin[v]) {
      son_cnt++;
      int lowx = tarjan(v, u);
      lowu = min(lowu, lowx);
      if (lowx >= tin[u]) {
        while (top != -1 && stck[top + 1] != v)
          nodes[bcc_cnt].emplace_back(
              stck[top--]);
        nodes[bcc_cnt++].emplace_back(u);
      }
    } else {
      lowu = min(lowu, tin[v]);
    }
  }
  if (p == -1 && son_cnt == 0) {
    nodes[bcc_cnt++].emplace_back(u);
  }
  return lowu;
}
void build_bccs() {
  timer = 1;
  top = -1;
  memset(tin, 0, sizeof(int) * n);
  for (int i = 0; i < n; i++) nodes[i] = {};
  bcc_cnt = 0;
  for (int u = 0; u < n; u++)
    if (!tin[u]) tarjan(u);
}
```

6.7 Binary Lifting/Jumping

Description: Given a function/successor grpah answers queries of the form which is the node after k moves starting from u.
Time: Build $O(N \cdot MAXLOG2)$, Query $O(MAXLOG2)$.

```
const int MAXN(2e5), MAXLOG2(30);
int bl[MAXN][MAXLOG2 + 1];
int N;
int jump(int u, ll k) {
  for (int i = 0; i <= MAXLOG2; i++) {
    if (k & (1ll << i)) u = bl[u][i];
  }
  return u;
}
void build() {
  for (int i = 1; i <= MAXLOG2; i++) {
    for (int j = 0; j < N; j++) {
      bl[j][i] = bl[bl[j][i - 1]][i - 1];
    }
  }
}
```

6.8 Block-Cut tree

```
struct block_cut_tree {
  int n;
  vector<int> id, is_cutpoint, tin, low, stk;
  vector<vector<int>> comps, tree;
  block_cut_tree(vector<vector<int>> &g)
      : n(g.size()),
        id(n),
        is_cutpoint(n),
        tin(n),
        low(n) {
    // build comps
    for (int i = 0; i < n; i++) {
      if (!tin[i]) {
        int timer = 0;
        dfs(i, -1, timer, g);
      }
    }
    int node_id = 0;
    for (int u = 0; u < n; u++) {
      if (is_cutpoint[u]) {
        id[u] = node_id++;
        tree.push_back({});
```

```
      }
    }
    for (auto &comp : comps) {
      int node = node_id++;
      tree.push_back({});
      for (int u : comp) {
        if (!is_cutpoint[u]) {
          id[u] = node;
        } else {
          tree[node].emplace_back(id[u]);
          tree[id[u]].emplace_back(node);
        }
      }
    }
  }
  void dfs(int u, int p, int &timer,
           vector<vector<int>> &g) {
    tin[u] = low[u] = ++timer;
    stk.emplace_back(u);
    for (auto v : g[u]) {
      if (v == p) continue;
      if (!tin[v]) {
        dfs(v, u, timer, g);
        low[u] = min(low[u], low[v]);
        if (low[v] >= tin[u]) {
          is_cutpoint[u] =
              (tin[u] > 1 or tin[v] > 2);
          comps.push_back({u});
          while (comps.back().back() != v) {
            comps.back().emplace_back(stk.back());
            stk.pop_back();
          }
        }
      } else
        low[u] = min(low[u], tin[v]);
    }
  }
};
```

6.9 D'Escopo-Pape

Description: Is a single source shortest path that works faster than Dijkstra's algorithm and the Bellman-Ford algorithm in most cases, and will also work for negative edges. However not for negative cycles. There exists cases where it runs in exponential time.
Usage: Returns a pair containing two vectors, the first one with the distance from s to every other node, and another one with the ancestor of each node, note that the ancestor of s is -1

```
using Edge = pair<ll, int>;
using Adj = vector<vector<Edge>>;
pair<vll, vi> desopo_pape(int s, int n,
                          const Adj &adj) {
  vll ds(n, LLONG_MAX), ps(n, -1);
  ds[s] = 0;
  vi ms(n, 2);
  deque<int> q;
  q.eb(s);
  while (len(q)) {
    int u = q.front();
    q.pop_front();
    ms[u] = 0;
    for (auto [w, v] : adj[u]) {
      if (chmin(ds[v], w + ds[u])) {
        ps[v] = u;
        if (ms[v] == 2)
          ms[v] = 1, q.pb(v);
        else if (ms[v] == 0)
          ms[v] = 1, q.pf(v);
      }
    }
  }
  return {ds, ps};
}
```

6.10 Dijkstra

```
const int MAXN = 1'00'000;
const ll MAXW = 1'000'000ll;
constexpr ll OO = MAXW * MAXN + 1;
using Edge = pair<ll, int>;  // { weigth, node}
using Adj = vector<vector<Edge>>;
template <typename T>
using min_heap =
    priority_queue<T, vector<T>, greater<T>>;
pair<vll, vi> dijkstra(const Adj &g, int s) {
  int n = len(g);
  min_heap<Edge> pq;
  vll ds(n, OO);
  vi ps(n, -1);
  pq.emp(0, s);
  ds[s] = 0;
  while (len(pq)) {
    auto [du, u] = pq.top();
    pq.pop();
    if (ds[u] < du) continue;
    for (auto [w, v] : g[u]) {
      ll ndv = du + w;
      if (chmin(ds[v], ndv)) {
        ps[v] = u;
        pq.emp(ndv, v);
      }
    }
    return {ds, ps};
  }
  // optional !
  vi recover_path(int source, int ending,
                  const vi &ps) {
    if (ps[ending] == -1) return {};
    int cur = ending;
    vi ans;
    while (cur != -1) {
      ans.eb(cur);
      cur = ps[cur];
    }
    reverse(all(ans));
    return ans;
  }
```

6.11 Dijkstra (K-shortest pahts)

```
const ll oo = 1e9 * 1e5 + 1;
using adj = vector<vector<pll>>;
vector<priority_queue<ll>> dijkstra(
    const vector<vector<pll>> &g, int n, int s,
    int k) {
  priority_queue<pll, vector<pll>, greater<pll>>
      pq;
  vector<priority_queue<ll>> dist(n);
  dist[0].emplace(0);
  pq.emplace(0, s);
  while (!pq.empty()) {
    auto [d1, v] = pq.top();
    pq.pop();
    if (not dist[v].empty() and
        dist[v].top() < d1)
      continue;
    for (auto [d2, u] : g[v]) {
      if (len(dist[u]) < k) {
        pq.emplace(d2 + d1, u);
        dist[u].emplace(d2 + d1);
      } else {
        if (dist[u].top() > d1 + d2) {
          dist[u].pop();
          dist[u].emplace(d1 + d2);
          pq.emplace(d2 + d1, u);
        }
      }
    }
  }
  return dist;
}
```

6.12 Extra Edges to Make Digraph Fully Strongly Connected

Description: Given a directed graph G find the necessary edges to add to make the graph a single strongly connected component.
Time: $O(N+M)$
Memory: $O(N)$

```
struct SCC {
  int n, num_sccs;
  vi2d adj;
  vi scc_id;
  SCC(int _n)
      : n(_n),
        num_sccs(0),
        adj(n),
        scc_id(n, -1) {}
  SCC(const vi2d &_adj) : SCC(len(_adj)) {
    adj = _adj;
    find_sccs();
  }
  void add_edge(int u, int v) { adj[u].eb(v); }
  void find_sccs() {
    int timer = 1;
    vi tin(n), st;
    st.reserve(n);
    function<int(int)> dfs = [&](int u) -> int {
      int low = tin[u] = timer++, siz = len(st);
      st.eb(u);
      for (int v : adj[u])
        if (scc_id[v] < 0)
          low =
              min(low, tin[v] ? tin[v] : dfs(v));
      if (tin[u] == low) {
        rep(i, siz, len(st)) scc_id[st[i]] =
            num_sccs;
        st.resize(siz);
        num_sccs++;
      }
      return low;
    };
    for (int i = 0; i < n; i++)
```

```cpp
        if (!tin[i]) dfs(i);
    }
};
vector<array<int, 2>> extra_edges(
        const vi2d &adj) {
    SCC scc(adj);
    auto scc_id = scc.scc_id;
    auto num_sccs = scc.num_sccs;
    if (num_sccs == 1) return {};
    int n = len(adj);
    vi2d scc_adj(num_sccs);
    vi zero_in(num_sccs, 1);
    rep(u, 0, n) {
        for (int v : adj[u]) {
            if (scc_id[u] == scc_id[v]) continue;
            scc_adj[scc_id[u]].eb(scc_id[v]);
            zero_in[scc_id[v]] = 0;
        }
    }
    int random_source =
            max_element(all(zero_in)) - zero_in.begin();
    vi vis(num_sccs);
    function<int(int)> dfs = [&](int u) {
        if (empty(scc_adj[u])) return u;
        for (int v : scc_adj[u])
            if (!vis[v]) {
                vis[v] = 1;
                int zero_out = dfs(v);
                if (zero_out != -1) return zero_out;
            }
        return (int)-1;
    };
    vector<array<int, 2>> edges;
    vi in_unused;
    rep(i, 0, num_sccs) {
        if (zero_in[i]) {
            vis[i] = 1;
            int zero_out = dfs(i);
            if (zero_out != -1)
                edges.push_back({zero_out, i});
            else
                in_unused.push_back(i);
        }
    }
    rep(i, 1, len(edges)) {
        swap(edges[i][0], edges[i - 1][0]);
    }
    rep(i, 0, num_sccs) {
        if (scc_adj[i].empty() && !vis[i]) {
            if (!in_unused.empty()) {
                edges.push_back({i, in_unused.back()});
                in_unused.pop_back();
            } else {
                edges.push_back({i, random_source});
            }
        }
    }
    for (int u : in_unused) edges.push_back({0, u});
    vi to_node(num_sccs);
    rep(i, 0, n) to_node[scc_id[i]] = i;
    for (auto &[u, v] : edges)
        u = to_node[u], v = to_node[v];
    return edges;
}
```

6.13 Find Articulation/Cut Points

Description: Given an **undirected** graph find it's articulation points.
Time: $O(N + M)$
Warning: A vertex u can be an articulation point if and only if has at least 2 adjascent vertex

```cpp
const int MAXN(100);
int N;
vi2d G;
int timer;
int tin[MAXN], low[MAXN];
set<int> cpoints;
int dfs(int u, int p = -1) {
    int cnt = 0;
    low[u] = tin[u] = timer++;
    for (auto v : G[u]) {
        if (not tin[v]) {
            cnt++;
            dfs(v, u);
            if (low[v] >= tin[u]) cpoints.insert(u);
            low[u] = min(low[u], low[v]);
        } else if (v != p)
            low[u] = min(low[u], tin[v]);
    }
    return cnt;
}
void getCutPoints() {
    memset(low, 0, sizeof(low));
    memset(tin, 0, sizeof(tin));
    cpoints.clear();
    timer = 1;
    for (int i = 0; i < N; i++) {
        if (tin[i]) continue;
        int cnt = dfs(i);
        if (cnt == 1) cpoints.erase(i);
    }
}
```

6.14 Find Bridge-Tree components

Usage: $label2CC(u, p)$ finds the 2-edge connected component of every node.
Time: $O(n + m)$

```cpp
const int maxn(3 '00' 000);
int tin[maxn], compId[maxn], qtdComps;
vi g[maxn], stck;
int n;
int dfs(int u, int p = -1) {
    int low = tin[u] = len(stck);
    stck.emplace_back(u);
    bool multEdge = false;
    for (auto v : g[u]) {
        if (v == p and !multEdge) {
            multEdge = 1;
            continue;
        }
        low = min(low,
                  tin[v] == -1 ? dfs(v, u) : tin[v]);
    }
    if (low == tin[u]) {
        for (int i = tin[u]; i < len(stck); i++)
            compId[stck[i]] = qtdComps;
        stck.resize(tin[u]);
        qtdComps++;
    }
    return low;
}
void label2CC() {
    memset(compId, -1, sizeof(int) * n);
    memset(tin, -1, sizeof(int) * n);
    stck.reserve(n);
    for (int i = 0; i < n; i++) {
        if (tin[i] == -1) dfs(i);
    }
}
```

6.15 Find Bridges

Description: Find every bridge in a **undirected** connected graph.
Warning: Remember to read the graph as pair where the second is the id of the edge !

```cpp
Time : $O(N + M) $ const int MAXN(10000),
              MAXM(100000);
int N, M, clk, tin[MAXN], low[MAXN],
        isBridge[MAXM];
vector<pii> G[MAXN];
void dfs(int u, int p = -1) {
    tin[u] = low[u] = clk++;
    for (auto [v, i] : G[u]) {
        if (v == p) continue;
        if (tin[v]) {
            low[u] = min(low[u], tin[v]);
        } else {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > tin[u]) {
                isBridge[i] = 1;
            }
        }
    }
}
void findBridges() {
    fill(tin, tin + N, 0);
    fill(low, low + N, 0);
    fill(isBridge, isBridge + M, 0);
    clk = 1;
    for (int i = 0; i < N; i++) {
        if (!tin[i]) dfs(i);
    }
}
```

6.16 Find Centroid

Description: Given a tree (don't forget to make it 'undirected'), find it's centroids.
@Time : $O(V)$

```cpp
void dfs(int u, int p, int n, vi2d &g, vi &sz,
         vi &centroid) {
    sz[u] = 1;
    bool iscentroid = true;
    for (auto v : g[u])
        if (v != p) {
            dfs(v, u, n, g, sz, centroid);
            if (sz[v] > n / 2) iscentroid = false;
            sz[u] += sz[v];
        }
```

```
  if (n - sz[u] > n / 2) iscentroid = false;
  if (iscentroid) centroid.eb(u);
}
vi getCentroid(vi2d &g, int n) {
  vi centroid;
  vi sz(n);
  dfs(0, -1, n, g, sz, centroid);
  return centroid;
}
```

6.17 Find bridges (online)

```
// O((n+m)*log(n))
struct BridgeFinder {
  // 2ecc = 2 edge conected component
  // cc = conected component
  vector<int> parent, dsu_2ecc, dsu_cc,
      dsu_cc_size;
  int bridges, lca_iteration;
  vector<int> last_visit;
  BridgeFinder(int n)
      : parent(n, -1),
        dsu_2ecc(n),
        dsu_cc(n),
        dsu_cc_size(n, 1),
        bridges(0),
        lca_iteration(0),
        last_visit(n) {
    for (int i = 0; i < n; i++) {
      dsu_2ecc[i] = i;
      dsu_cc[i] = i;
    }
  }
  int find_2ecc(int v) {
    if (v == -1) return -1;
    return dsu_2ecc[v] == v
               ? v
               : dsu_2ecc[v] =
                     find_2ecc(dsu_2ecc[v]);
  }
  int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v
               ? v
               : dsu_cc[v] = find_cc(dsu_cc[v]);
  }
  void make_root(int v) {
    v = find_2ecc(v);
    int root = v;
    int child = -1;
    while (v != -1) {
      int p = find_2ecc(parent[v]);
      parent[v] = child;
      dsu_cc[v] = root;
      child = v;
      v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
  }
  void merge_path(int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b;
    int lca = -1;
    while (lca == -1) {
      if (a != -1) {
        a = find_2ecc(a);
        path_a.push_back(a);
        if (last_visit[a] == lca_iteration) {
          lca = a;
          break;
        }
        last_visit[a] = lca_iteration;
        a = parent[a];
      }
      if (b != -1) {
        b = find_2ecc(b);
        path_b.push_back(b);
        if (last_visit[b] == lca_iteration) {
          lca = b;
          break;
        }
        last_visit[b] = lca_iteration;
        b = parent[b];
      }
    }
    for (auto v : path_a) {
      dsu_2ecc[v] = lca;
      if (v == lca) break;
      --bridges;
    }
    for (auto v : path_b) {
      dsu_2ecc[v] = lca;
      if (v == lca) break;
      --bridges;
    }
  }
  void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b) return;
    int ca = find_cc(a);
    int cb = find_cc(b);
    if (ca != cb) {
      ++bridges;
      if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
        swap(a, b);
        swap(ca, cb);
      }
      make_root(a);
      parent[a] = dsu_cc[a] = b;
      dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
      merge_path(a, b);
    }
  }
};
```

6.18 Floyd Warshall

Description: Simply finds the minimal distance for each node to every other node. $O(V^3)$

```
vector<vll> floyd_warshall(const vector<vll> &adj,
                           ll n) {
  auto dist = adj;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      for (int k = 0; k < n; ++k) {
        dist[j][k] = min(dist[j][k],
                         dist[j][i] + dist[i][k]);
      }
    }
  }
  return dist;
}
```

6.19 Functional/Successor Graph

Description: Given a functional graph find the vertice after k moves starting at u and also the distance between u and v, if it's impossible to reach v starting at u returns -1.
Time: build $O(N \cdot MAXLOG2)$, kth $O(MAXLOG2)$, dist $O(MAXLOG2)$

```
const int MAXN(2 '000' 000), MAXLOG2(24);
int N;
vi2d succ(MAXN, vi(MAXLOG2 + 1));
vi dst(MAXN, 0);
int vis[MAXN];
void dfsbuild(int u) {
  if (vis[u]) return;
  vis[u] = 1;
  int v = succ[u][0];
  dfsbuild(v);
  dst[u] = dst[v] + 1;
}
void build() {
  for (int i = 0; i < N; i++) {
    if (not vis[i]) dfsbuild(i);
  }
  for (int k = 1; k <= MAXLOG2; k++) {
    for (int i = 0; i < N; i++) {
      succ[i][k] = succ[succ[i][k - 1]][k - 1];
    }
  }
}
int kth(int u, ll k) {
  if (k <= 0) return u;
  for (int i = 0; i <= MAXLOG2; i++)
    if ((1ll << i) & k) u = succ[u][i];
  return u;
}
int dist(int u, int v) {
  int cu = kth(u, dst[u]);
  if (kth(u, dst[u] - dst[v]) == v)
    return dst[u] - dst[v];
  else if (kth(cu, dst[cu] - dst[v]) == v)
    return dst[u] + (dst[cu] - dst[v]);
  else
    return -1;
}
```

6.20 Kruskal

Description: Find the minimum spanning tree of a graph.
Time: $O(E \log E)$

```
struct UFDS {
  vector<int> ps, sz;
  int components;
  UFDS(int n)
      : ps(n + 1), sz(n + 1, 1), components(n) {
    iota(all(ps), 0);
  }
  int find_set(int x) {
    return (x == ps[x]
                ? x
                : (ps[x] = find_set(ps[x])));
  }
```

```
  bool same_set(int x, int y) {
    return find_set(x) == find_set(y);
  }
  void union_set(int x, int y) {
    x = find_set(x);
    y = find_set(y);
    if (x == y) return;
    if (sz[x] < sz[y]) swap(x, y);
    ps[y] = x;
    sz[x] += sz[y];
    components--;
  }
};
vector<tuple<ll, int, int>> kruskal(
    int n, vector<tuple<ll, int, int>> &edges) {
  UFDS ufds(n);
  vector<tuple<ll, int, int>> ans;
  sort(all(edges));
  for (auto [a, b, c] : edges) {
    if (ufds.same_set(b, c)) continue;
    ans.emplace_back(a, b, c);
    ufds.union_set(b, c);
  }
  return ans;
}
```

6.21 Lowest Common Ancestor

Description: Given two nodes of a tree find their lowest common ancestor, or their distance

```
template <typename T>
struct SparseTable {
  vector<T> v;
  int n;
  static const int b = 30;
  vi mask, t;
  int op(int x, int y) {
    return v[x] < v[y] ? x : y;
  }
  int msb(int x) {
    return __builtin_clz(1) - __builtin_clz(x);
  }
  SparseTable() {}
  SparseTable(const vector<T> &v_)
      : v(v_), n(v.size()), mask(n), t(n) {
    for (int i = 0, at = 0; i < n;
         mask[i++] = at |= 1) {
      at = (at << 1) & ((1 << b) - 1);
      while (at and op(i, i - msb(at & -at)) == i)
        at ^= at & -at;
    }
    for (int i = 0; i < n / b; i++)
      t[i] = b * i + b - 1 -
             msb(mask[b * i + b - 1]);
    for (int j = 1; (1 << j) <= n / b; j++)
      for (int i = 0; i + (1 << j) <= n / b; i++)
        t[n / b * j + i] =
            op(t[n / b * (j - 1) + i],
               t[n / b * (j - 1) + i +
                 (1 << (j - 1))]);
  }
  int small(int r, int sz = b) {
    return r - msb(mask[r] & ((1 << sz) - 1));
  }
  T query(int l, int r) {
    if (r - l + 1 <= b)
      return small(r, r - l + 1);
    int ans = op(small(l + b - 1), small(r));
    int x = l / b + 1, y = r / b - 1;
    if (x <= y) {
      int j = msb(y - x + 1);
      ans =
          op(ans,
             op(t[n / b * j + x],
                t[n / b * j + y - (1 << j) + 1]));
    }
    return ans;
  }
};
struct LCA {
  SparseTable<int> st;
  int n;
  vi v, pos, dep;
  LCA(const vi2d &g, int root)
      : n(len(g)), pos(n) {
    dfs(root, 0, -1, g);
    st = SparseTable<int>(vector<int>(all(dep)));
  }
  void dfs(int i, int d, int p, const vi2d &g) {
    v.eb(len(dep)) = i, pos[i] = len(dep),
    dep.eb(d);
    for (auto j : g[i])
      if (j != p) {
        dfs(j, d + 1, i, g);
        v.eb(len(dep)) = i, dep.eb(d);
      }
  }
  int lca(int a, int b) {
    int l = min(pos[a], pos[b]);
    int r = max(pos[a], pos[b]);
    return v[st.query(l, r)];
  }
  int dist(int a, int b) {
    return dep[pos[a]] + dep[pos[b]] -
           2 * dep[pos[lca(a, b)]];
  }
};
```

6.22 Lowest Common Ancestor (Binary Lifting)

Description: Given a directed tree, finds the LCA between two nodes using binary lifting, and answer a few queries with it.
Usage:

- lca: returns the LCA between the two given nodes
- on_path: fids if c is in the path from a to b

Time: build $O(N \cdot MAXLOG2)$, all queries $O(MAXLOG2)$

```
struct LCA {
  int n;
  const int maxlog;
  vector<vector<int>> up;
  vector<int> depth;
  LCA(const vector<vector<int>> &tree)
      : n(tree.size()),
        maxlog(ceil(log2(n))),
        up(n, vector<int>(maxlog + 1)),
        depth(n, -1) {
    for (int i = 0; i < n; i++) {
      if (depth[i] == -1) {
        depth[i] = 0;
        dfs(i, -1, tree);
      }
    }
  }
  void dfs(int u, int p,
           const vector<vector<int>> &tree) {
    if (p != -1) {
      depth[u] = depth[p] + 1;
      up[u][0] = p;
      for (int i = 1; i <= maxlog; i++) {
        up[u][i] = up[up[u][i - 1]][i - 1];
      }
    }
    for (int v : tree[u]) {
      if (v == p) continue;
      dfs(v, u, tree);
    }
  }
  int kth_jump(int u, int k) {
    for (int i = maxlog; i >= 0; i--) {
      if ((1 << i) & k) {
        u = up[u][i];
      }
    }
    return u;
  }
  int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    int diff = depth[u] - depth[v];
    u = kth_jump(u, diff);
    if (u == v) return u;
    for (int i = maxlog; i >= 0; i--) {
      if (up[u][i] != up[v][i]) {
        u = up[u][i];
        v = up[v][i];
      }
    }
    return up[u][0];
  }
  bool on_path(int u, int v, int s) {
    int uv = lca(u, v), us = lca(u, s),
        vs = lca(v, s);
    return (uv == s or (us == uv and vs == s) or
            (vs == uv and us == s));
  }
  int dist(int u, int v) {
    return depth[u] + depth[v] -
           2 * depth[lca(u, v)];
  }
};
```

6.23 Maximum flow (Dinic)

Description: Finds the **maximum flow** in a graph network, given the **source** s and the **sink** t. Add edge from a to b with capcity c.
Time: In general $O(E \cdot V^2)$, if every capacity is 1, and every vertice has in degree equal 1 or out degree equal 1 then $O(E \cdot \sqrt{V})$,
Warning: Suffle the edges list for every vertice may take you out of the worst case

```
struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() {
```

```
    return max(oc - c, 0LL);
  }  // if you need flows
};
vi lvl, ptr, q;
vector<vector<Edge>> adj;
Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
void addEdge(int a, int b, ll c, ll rcap = 0) {
  adj[a].pb({b, len(adj[b]), c, c});
  adj[b].pb({a, len(adj[a]) - 1, rcap, rcap});
}
ll dfs(int v, int t, ll f) {
  if (v == t || !f) return f;
  for (int &i = ptr[v]; i < len(adj[v]); i++) {
    Edge &e = adj[v][i];
    if (lvl[e.to] == lvl[v] + 1)
      if (ll p = dfs(e.to, t, min(f, e.c))) {
        e.c -= p, adj[e.to][e.rev].c += p;
        return p;
      }
  }
  return 0;
}
ll maxFlow(int s, int t) {
  ll flow = 0;
  q[0] = s;
  rep(L, 0, 31) {
    do {  // 'int L=30' maybe faster for random
          // data
      lvl = ptr = vi(len(q));
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (Edge e : adj[v])
          if (!lvl[e.to] && e.c >> (30 - L))
            q[qe++] = e.to,
            lvl[e.to] = lvl[v] + 1;
      }
      while (ll p = dfs(s, t, LLONG_MAX))
        flow += p;
    } while (lvl[t]);
  }
  return flow;
}
bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

6.24 Minimum Cost Flow

Description: Given a network find the minimum cost to achieve a flow of at most f. Works with **directed** and **undirected** graphs
Usage:

- **add(u, v, w, c):** adds an edge from u to v with capacity w and cost c.
- **flow(s, t, f):** return a pair $(flow, cost)$ with the maximum flow until f with source at s and sink at t, with the minimum cost possible.

Time: $O(N \cdot M + f \cdot m \log n)$

```
template <typename T>
struct mcmf {
  struct edge {
    int to, rev, flow, cap;
    bool res;  // if it's a reverse edge
    T cost;    // cost per unity of flow
    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) {
    edge a = edge(v, (int)g[v].size(), 0, w, cost,
                  false);
    edge b = edge(u, (int)g[u].size(), 0, 0,
                  -cost, true);
    g[u].emplace_back(a);
    g[v].emplace_back(b);
  }
  /* don't code this if there isn't negative cyles
   * ! */
  vector<T> spfa(int s) {
    deque<int> q;
    vector<char> 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 < (int)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 < (int)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) {
    vector<T> pot((int)g.size(), 0);
    /* comment or remove this line if there isn't
     * negative cyles*/
    // pot = spfa(s);
    int f = 0;
    T ret = 0;
    while (f < flow and dijkstra(s, t, pot)) {
      for (int i = 0; i < (int)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);
  }
};
```

6.25 Minimum Vertex Cover (already divided)

Description: Given a bipartite graph g with n vertices at left and m vertices at right, where $g[i]$ are the possible right side matches of vertex i from left side, find a minimum vertex cover. The size is the same as the size of the maximum matching, and the complement is a maximum independent set.

```
vector<int> min_vertex_cover(
    vector<vector<int>> &g, int n, int m) {
  vector<int> match(m, -1), vis;
  auto find = [&](auto &&self, int j) -> bool {
    if (match[j] == -1) return 1;
    vis[j] = 1;
    int di = match[j];
    for (int e : g[di])
      if (!vis[e] and self(self, e)) {
        match[e] = di;
        return 1;
      }
    return 0;
  };
  for (int i = 0; i < (int)g.size(); i++) {
    vis.assign(match.size(), 0);
    for (int j : g[i]) {
      if (find(find, j)) {
        match[j] = i;
        break;
      }
    }
  }
  int res =
      (int)match.size() -
      (int)count(match.begin(), match.end(), -1);
  vector<char> lfound(n, true), seen(m);
  for (int it : match)
    if (it != -1) lfound[it] = false;
  vector<int> q, cover;
  for (int i = 0; i < n; i++)
    if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back();
    q.pop_back();
    lfound[i] = 1;
    for (int e : g[i])
      if (!seen[e] and match[e] != -1) {
        seen[e] = true;
        q.push_back(match[e]);
      }
  }
  for (int i = 0; i < n; i++)
    if (!lfound[i]) cover.push_back(i);
  for (int i = 0; i < m; i++)
    if (seen[i]) cover.push_back(n + i);
  assert((int)size(cover) == res);
  return cover;
}
```

6.26 Prim (MST)

Description: Given a graph with N vertex finds the minimum spanning tree, if there is no such three returns inf, it starts using the edges that connect with each $s_i \in s$, if none is provided than it starts with the edges of node 0.
Time: $O(V \log E)$

```
const int MAXN(1 '00' 000);
int N;
vector<pair<ll, int>> G[MAXN];
ll prim(vi s = vi(1, 0)) {
  priority_queue<pair<ll, int>,
                 vector<pair<ll, int>>,
                 greater<pair<ll, int>>>
      pq;
  vector<char> ingraph(MAXN);
  int ingraphcnt(0);
  for (auto si : s) {
    ingraphcnt++;
    ingraph[si] = true;
    for (auto &[w, v] : G[si]) pq.emplace(w, v);
  }
  ll mstcost = 0;
  while (ingraphcnt < N and !pq.empty()) {
    ll w;
    int v;
    do {
      tie(w, v) = pq.top();
      pq.pop();
    } while (not pq.empty() and ingraph[v]);
    mstcost += w, ingraph[v] = true, ingraphcnt++;
    for (auto &[w2, v2] : G[v]) {
      pq.emplace(w2, v2);
    }
  }
  return ingraphcnt == N ? mstcost : oo;
}
```

6.27 Shortest Path With K-edges

Description: Given an adjacency matrix of a graph, and a number K computes the shortest path between all nodes that uses exactly K edges, so for $0 \leq i, j \leq N-1$ ans[i][j] = "the shortest path between i and j that uses exactly K edges, remember to initialize the adjacency matrix with ∞.
Time: $O(N^3 \cdot \log K)$

```
template <typename T>
vector<vector<T>> prod(vector<vector<T>> &a,
                       vector<vector<T>> &b) {
  const T _oo = numeric_limits<T>::max();
  int n = a.size();
  vector<vector<T>> c(n, vector<T>(n, _oo));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++)
        if (a[i][k] != _oo and b[k][j] != _oo)
          c[i][j] =
              min(c[i][j], a[i][k] + b[k][j]);
  return c;
}
template <typename T>
vector<vector<T>> shortest_with_k_moves(
    vector<vector<T>> adj, long long k) {
  if (k == 1) return adj;
  auto ans = adj;
  k--;
  while (k) {
    if (k & 1) ans = prod(ans, adj);
    k >>= 1;
    adj = prod(adj, adj);
  }
  return ans;
}
```

6.28 Strongly Connected Components (struct)

Description: Find the connected component for each edge (already in a topological order), some additional functions are also provided.
Time: Build: $O(V + E)$

```
struct SCC {
  int n, num_sccs;
  vi2d adj;
  vi scc_id;
  SCC(int _n)
      : n(_n),
        num_sccs(0),
        adj(n),
        scc_id(n, -1) {}
  void add_edge(int u, int v) { adj[u].eb(v); }
  void find_sccs() {
    int timer = 1;
    vi tin(n), st;
    st.reserve(n);
    function<int(int)> dfs = [&](int u) -> int {
      int low = tin[u] = timer++, siz = len(st);
      st.eb(u);
      for (int v : adj[u])
        if (scc_id[v] < 0)
          low =
              min(low, tin[v] ? tin[v] : dfs(v));
      if (tin[u] == low) {
        rep(i, siz, len(st)) scc_id[st[i]] =
            num_sccs;
        st.resize(siz);
        num_sccs++;
      }
      return low;
    };
    for (int i = 0; i < n; i++)
      if (!tin[i]) dfs(i);
  }
  vector<set<int>> build_gscc() {
    vector<set<int>> gscc;
    for (int i = 0; i < len(adj); ++i)
      for (auto j : adj[i])
        if (scc_id[i] != scc_id[j])
          gscc[scc_id[i]].emplace(scc_id[j]);
    return gscc;
  }
  vi2d per_comp() {
    vi2d ret(num_sccs);
    rep(i, 0, n) ret[scc_id[i]].eb(i);
    reverse(all(
        ret));  // already in topological order ;)
    return ret;
  }
};
```

6.29 Topological Sorting (Kahn)

Description: Finds the topological sorting in a **DAG**, if the given graph is not a **DAG** than an empty vector is returned, need to 'initialize' the **INCNT** as you build the graph.
Time: $O(V + E)$

```
const int MAXN(2 '00' 000);
int INCNT[MAXN];
vi2d GOUT(MAXN);
int N;
vi toposort() {
  vi order;
  queue<int> q;
```

```
  for (int i = 0; i < N; i++)
    if (!INCNT[i]) q.emplace(i);
  while (!q.empty()) {
    auto u = q.front();
    q.pop();
    order.emplace_back(u);
    for (auto v : GOUT[u]) {
      INCNT[v]--;
      if (INCNT[v] == 0) q.emplace(v);
    }
  }
  return len(order) == N ? order : vi();
}
```

6.30 Topological Sorting (Tarjan)

Description: Finds a the topological order for the graph, if there is no such order it means the graph is cyclic, then it returns an empty vector
Time: $O(V + E)$

```
const int maxn(1 '00' 000);
int n, m;
vi g[maxn];
int not_found = 0, found = 1, processed = 2;
int state[maxn];
bool dfs(int u, vi &order) {
  if (state[u] == processed) return true;
  if (state[u] == found) return false;
  state[u] = found;
  for (auto v : g[u]) {
    if (not dfs(v, order)) return false;
  }
  state[u] = processed;
  order.emplace_back(u);
  return true;
}
vi topo_sort() {
  vi order;
  memset(state, 0, sizeof state);
  for (int u = 0; u < n; u++) {
    if (state[u] == not_found and
        not dfs(u, order))
      return {};
  }
  reverse(all(order));
  return order;
}
```

6.31 Tree Isomorphism (not rooted)

Description: Two trees are considered **isomorphic** if the hash given by $thash()$ is the same.
Time: $O(V \cdot \log V)$

```
map<vi, int> mphash;
struct Tree {
  int n;
  vi2d g;
  vi sz, cs;
  Tree(int n_) : n(n_), g(n), sz(n) {}
  void add_edge(int u, int v) {
    g[u].emplace_back(v);
    g[v].emplace_back(u);
  }
  void dfs_centroid(int v, int p) {
    sz[v] = 1;
    bool cent = true;
    for (int u : g[v])
      if (u != p) {
        dfs_centroid(u, v);
        sz[v] += sz[u];
        cent &= not(sz[u] > n / 2);
      }
    if (cent and n - sz[v] <= n / 2)
      cs.push_back(v);
  }
  int fhash(int v, int p) {
    vi h;
    for (int u : g[v])
      if (u != p) h.push_back(fhash(u, v));
    sort(all(h));
    if (!mphash.count(h))
      mphash[h] = mphash.size();
    return mphash[h];
  }
  ll thash() {
    cs.clear();
    dfs_centroid(0, -1);
    if (cs.size() == 1) return fhash(cs[0], -1);
    ll h1 = fhash(cs[0], cs[1]),
       h2 = fhash(cs[1], cs[0]);
    return (min(h1, h2) << 30ll) + max(h1, h2);
  }
};
```

6.32 Tree Isomorphism (rooted)

Description: Given a rooted tree find the hash of each subtree, if two roots of two distinct trees have the same hash they are considered isomorphic
Time: hash first time in $O(\log N_v \cdot N_v)$ where (N_v) is the of the subtree of v

```
map<vi, int> hasher;
int hs = 0;
struct RootedTreeIso {
  int n;
  vi2d adj;
  vi hashes;
  RootedTreeIso(int _n)
      : n(_n), adj(_n), hashes(_n, -1){};
  void add_edge(int u, int v) {
    adj[u].emplace_back(v);
    adj[v].emplace_back(u);
  }
  int hash(int u, int p = -1) {
    if (hashes[u] != -1) return hashes[u];
    vi children;
    for (auto v : adj[u])
      if (v != p)
        children.emplace_back(hash(v, u));
    sort(all(children));
    if (!hasher.count(children))
      hasher[children] = hs++;
    return hashes[u] = hasher[children];
  }
};
```

6.33 Tree diameter (DP)

```
const int MAXN(1 '000' 000);
int N;
vi G[MAXN];
int diameter, toLeaf[MAXN];
void calcDiameter(int u = 0, int p = -1) {
  int d1, d2;
  d1 = d2 = -1;
  for (auto v : G[u]) {
    if (v != p) {
      calcDiameter(v, u);
      d1 = max(d1, toLeaf[v]);
      tie(d1, d2) = minmax({d1, d2});
    }
  }
  toLeaf[u] = d2 + 1;
  diameter = max(diameter, d1 + d2 + 2);
}
```

7 Math

7.1 Arithmetic Progression Sum

Usage:

- s : first term
- d : common difference
- n : number of terms

```
ll arithmeticProgressionSum(ll s, ll d, ll n) {
  return (s + (s + d * (n - 1))) * n / 2ll;
}
```

7.2 Binomial

Time: $O(N \cdot K)$
Memory: $O(K)$

```
ll binom(ll n, ll k) {
  if (k > n) return 0;
  vll dp(k + 1, 0);
  dp[0] = 1;
  for (ll i = 1; i <= n; i++)
    for (ll j = k; j > 0; j--)
      dp[j] = dp[j] + dp[j - 1];
  return dp[k];
}
```

7.3 Binomial MOD

Description: find $\binom{n}{k} \pmod{MOD}$
Time:

- precompute: on first call it takes $O(MAXNBIN)$ to precompute the factorials
- query: $O(1)$.

Memory: $O(MAXNBIN)$
Warning: Remember to set $MAXNBIN$ properly !

```cpp
const ll MOD = 998244353;
inline ll binom(ll n, ll k) {
  static const int BINMAX = 2'000'000;
  static vll FAC(BINMAX + 1), FINV(BINMAX + 1);
  static bool done = false;
  if (!done) {
    vll INV(BINMAX + 1);
    FAC[0] = FAC[1] = INV[1] = FINV[0] = FINV[1] =
        1;
    for (int i = 2; i <= BINMAX; i++) {
      FAC[i] = FAC[i - 1] * i % MOD;
      INV[i] = MOD - MOD / i * INV[MOD % i] % MOD;
      FINV[i] = FINV[i - 1] * INV[i] % MOD;
    }
    done = true;
  }
  if (n < k || n < 0 || k < 0) return 0;
  return FAC[n] * FINV[k] % MOD * FINV[n - k] %
         MOD;
}
```

7.4 Chinese Remainder Theorem

Description: Find the solution X to the N modular equations.

$$\begin{gathered} x \equiv a_1 (mod m_1) \\ \cdots \\ x \equiv a_n (mod m_n) \end{gathered} \tag{1}$$

The m_i don't need to be coprime, if there is no solution then it returns -1.

```cpp
tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
  if (!a) return {b, 0, 1};
  auto [g, x, y] = ext_gcd(b % a, a);
  return {g, y - b / a * x, x};
}
template <typename T = ll>
struct crt {
  T a, m;
  crt() : a(0), m(1) {}
  crt(T a_, T m_) : a(a_), m(m_) {}
  crt operator*(crt C) {
    auto [g, x, y] = ext_gcd(m, C.m);
    if ((a - C.a) % g != 0) a = -1;
    if (a == -1 or C.a == -1) return crt(-1, 0);
    T lcm = m / g * C.m;
    T ans =
        a + (x * (C.a - a) / g % (C.m / g)) * m;
    return crt((ans % lcm + lcm) % lcm, lcm);
  }
};
template <typename T = ll>
struct Congruence {
  T a, m;
};
template <typename T = ll>
T chinese_remainder_theorem(
    const vector<Congruence<T>> &equations) {
  crt<T> ans;
  for (auto &[a_, m_] : equations) {
    ans = ans * crt<T>(a_, m_);
  }
  return ans.a;
}
```

7.5 Derangement / Matching Problem

Description: Computes the derangement of N, which is given by the formula: $D_N = N!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + (-1)^N \frac{1}{N!}\right)$

Time: $O(N)$

```cpp
#warning Remember to call precompute !
const ll MOD = 1e9 + 7;
const int MAXN(1 '000' 000);
ll fats[MAXN + 1];
void precompute() {
  fats[0] = 1;
  for (ll i = 1; i <= MAXN; i++) {
    fats[i] = (fats[i - 1] * i) % MOD;
  }
}
ll fastpow(ll a, ll p, ll m) {
  ll ret = 1;
  while (p) {
    if (p & 1) ret = (ret * a) % MOD;
    p >>= 1;
    a = (a * a) % MOD;
  }
  return ret;
}
ll divmod(ll a, ll b) {
  return (a * fastpow(b, MOD - 2, MOD)) % MOD;
}
ll derangement(const ll n) {
  ll ans = fats[n];
  for (ll i = 1; i <= n; i++) {
    ll k = divmod(fats[n], fats[i]);
    if (i & 1) {
      ans = (ans - k + MOD) % MOD;
    } else {
      ans = (ans + k) % MOD;
    }
  }
  return ans;
}
```

7.6 Euler Phi $\varphi(N)$

Description: Computes the number of positive integers less than N that are coprimes with N, in $O(\sqrt{N})$.

```cpp
int phi(int n) {
  if (n == 1) return 1;
  auto fs = factorization(
      n);  // a vctor of pair or a map
  auto res = n;
  for (auto [p, k] : fs) {
    res /= p;
    res *= (p - 1);
  }
  return res;
}
```

7.7 Euler phi $\varphi(N)$ (in range)

Description: Computes the number of positive integers less than n that are coprimes with N, in the range $[1, N]$, in $O(N \log N)$.

```cpp
const int MAX = 1e6;
vi range_phi(int n) {
  bitset<MAX> sieve;
  vi phi(n + 1);
  iota(phi.begin(), phi.end(), 0);
  sieve.set();
  for (int p = 2; p <= n; p += 2) phi[p] /= 2;
  for (int p = 3; p <= n; p += 2) {
    if (sieve[p]) {
      for (int j = p; j <= n; j += p) {
        sieve[j] = false;
        phi[j] /= p;
        phi[j] *= (p - 1);
      }
    }
  }
  return phi;
}
```

7.8 FFT convolution and exponentiation

```cpp
const ld PI = acos(-1);
/* change the ld to doulbe may increase
 * performance =D */
struct num {
  ld a{0.0}, b{0.0};
  num() {}
  num(ld na) : a{na} {}
  num(ld na, ld nb) : a{na}, b{nb} {}
  const num operator+(const num &c) const {
    return num(a + c.a, b + c.b);
  }
  const num operator-(const num &c) const {
    return num(a - c.a, b - c.b);
  }
  const num operator*(const num &c) const {
    return num(a * c.a - b * c.b,
               a * c.b + b * c.a);
  }
  const num operator/(const ll &c) const {
    return num(a / c, b / c);
  }
};
void fft(vector<num> &a, bool invert) {
  int n = len(a);
  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 sz = 2; sz <= n; sz <<= 1) {
    ld ang = 2 * PI / sz * (invert ? -1 : 1);
    num wsz(cos(ang), sin(ang));
    for (int i = 0; i < n; i += sz) {
      num w(1);
      rep(j, 0, sz / 2) {
        num u = a[i + j],
            v = a[i + j + sz / 2] * w;
        a[i + j] = u + v;
        a[i + j + sz / 2] = u - v;
        w = w * wsz;
      }
    }
  }
  if (invert)
    for (num &x : a) x = x / n;
}
```

```cpp
vi conv(vi const a, vi const b) {
  vector<num> fa(all(a));
  vector<num> fb(all(b));
  int n = 1;
  while (n < len(a) + len(b)) n <<= 1;
  fa.resize(n);
  fb.resize(n);
  fft(fa, false);
  fft(fb, false);
  rep(i, 0, n) fa[i] = fa[i] * fb[i];
  fft(fa, true);
  vi result(n);
  rep(i, 0, n) result[i] = round(fa[i].a);
  while (len(result) and result.back() == 0)
    result.pop_back();
  /* Unconment this line if you want a boolean
   * convolution*/
  // for (auto &xi : result) xi = min(xi, 1ll);
  return result;
}
vll poly_exp(vll &ps, int k) {
  vll ret(len(ps));
  auto base = ps;
  ret[0] = 1;
  while (k) {
    if (k & 1) ret = conv(ret, base);
    k >>= 1;
    base = conv(base, base);
  }
  return ret;
}
```

7.9 Factorial Factorization

Description: Computes the factorization of $N!$ in $\varphi(N) * \log N$
Time: $O(\varphi(N) \cdot \log N)$

```cpp
ll E(ll n, ll p) {
  ll k = 0, b = p;
  while (b <= n) {
    k += n / b;
    b *= p;
  }
  return k;
}
map<ll, ll> factorial_factorization(
    ll n, const vll &primes) {
  map<ll, ll> fs;
  for (const auto &p : primes) {
    if (p > n) break;
    fs[p] = E(n, p);
  }
  return fs;
}
```

7.10 Factorization

Description: Computes the factorization of N.
Time: $O(\sqrt{n})$.

```cpp
map<ll, ll> factorization(ll n) {
  map<ll, ll> ans;
  for (ll i = 2; i * i <= n; i++) {
    ll count = 0;
    for (; n % i == 0; count++, n /= i)
      ;
    if (count) ans[i] = count;
  }
  if (n > 1) ans[n]++;
  return ans;
}
```

7.11 Factorization (Pollard's Rho)

Description: Factorizes a number into its prime factors.
Time: $O(N^{(\frac{1}{4})} * \log(N))$.

```cpp
ll mul(ll a, ll b, ll m) {
  ll ret =
      a * b - (ll)((ld)1 / m * a * b + 0.5) * m;
  return ret < 0 ? ret + m : ret;
}
ll pow(ll a, ll b, ll m) {
  ll ans = 1;
  for (; b > 0; b /= 2ll, a = mul(a, a, m)) {
    if (b % 2ll == 1) ans = mul(ans, a, m);
  }
  return 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, 795265022}) {
    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);
}
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(), r.begin(), r.end());
  return l;
}
```

7.12 Fast Pow

Description: Computes $a^b \pmod m$
Time: $O(\log B)$.

```cpp
ll fpow(ll a, ll b, ll m) {
  ll ret = 1;
  while (b) {
    if (b & 1) ret = (ret * a) % m;
    b >>= 1;
    a = (a * a) % m;
  }
  return ret;
}
```

7.13 Find diophantine equation solution

Description: Given a b, c finds the solution to the equation $ax + by = c$, the result will be stored in the reference variables $x0$ and $y0$.
Time: $O(\log min(a, b))$

```cpp
template <typename T>
tuple<T, T, T> ext_gcd(T a, T b) {
  if (b == 0) return {a, 1, 0};
  auto [d, x1, y1] = ext_gcd(b, a % b);
  return {d, y1, x1 - y1 * (a / b)};
}
template <typename T>
tuple<bool, T, T> find_any_solution(T a, T b,
                                    T c) {
  assert(a != 0 or b != 0);
#warning Be careful with overflow, use __int128 if
    needed !
  auto [d, x0, y0] =
      ext_gcd(a < 0 ? -a : a, b < 0 ? -b : b);
  if (c % d) return {false, 0, 0};
  x0 *= c / d;
  y0 *= c / d;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return {true, x0, y0};
}
```

7.14 Find multiplicatinve inverse

```cpp
ll inv(ll a, ll m) {
  return a > 1ll ? m - inv(m % a, a) * m / a
                 : 1ll;
}
```

7.15 GCD

```cpp
ll gcd(ll a, ll b) {
  return b ? gcd(b, a % b) : a;
}
```

7.16 Gauss XOR elimination / XOR-SAT

Description: Execute gaussian elimination with xor over the system $Ax = b$ in. The add method must receive a bitset indicating which variables are present in the equation, and the solution of the equation.
Time: $O(\frac{nm^2}{64})$

```
const int MAXXI = 2009;
using Equation = bitset<MAXXI>;
struct GaussXor {
  vector<char> B;
  vector<Equation> A;
  void add(const Equation &ai, bool bi) {
    A.push_back(ai);
    B.push_back(bi);
  }
  pair<bool, Equation> solution() {
    int cnt = 0, n = A.size();
    Equation vis;
    vis.set();
    Equation x;
    for (int j = MAXXI - 1, i; j >= 0; j--) {
      for (i = cnt; i < n; i++) {
        if (A[i][j]) break;
      }
      if (i == n) continue;
      swap(A[i], A[cnt]), swap(B[i], B[cnt]);
      i = cnt++;
      vis[j] = 0;
      for (int k = 0; k < n; k++) {
        if (i == k || !A[k][j]) continue;
        A[k] ^= A[i];
        B[k] ^= B[i];
      }
    }
    x = vis;
    for (int i = 0; i < n; i++) {
      int acum = 0;
      for (int j = 0; j < MAXXI; j++) {
        if (!A[i][j]) continue;
        if (!vis[j]) {
          vis[j] = 1;
          x[j] = acum ^ B[i];
        }
        acum ^= x[j];
      }
      if (acum != B[i])
        return {false, Equation()};
    }
    return {true, x};
  }
};
```

7.17 Integer partition

Description: Find the total of ways to partition a given number N in such way that none of the parts is greater than K.
Time: $O(N \cdot min(N, K))$
Memory: $O(N)$
Warning: Remember to memset everything to -1 before using it

```
const ll MOD = 1000000007;
const int MAXN(100);
ll memo[MAXN + 1];
ll dp(ll n, ll k = oo) {
  if (n == 0) return 1;
  ll &ans = memo[n];
  if (ans != -1) return ans;
  ans = 0;
  for (int i = 1; i <= min(n, k); i++) {
    ans = (ans + dp(n - i, k)) % MOD;
  }
  return ans;
}
```

7.18 LCM

```
ll gcd(ll a, ll b) {
  return b ? gcd(b, a % b) : a;
}
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
```

7.19 Linear Recurrence

Description: Find the n-th term of a linear recurrence, given the recurrence *rec* and the first K values of the recurrence, remember that first_k[i] is the value of $f(i)$, considering 0-indexing.
Time: $O(K^3 \log N)$

```
template <typename T>
vector<vector<T>> prod(vector<vector<T>> &a,
                       vector<vector<T>> &b,
                       const ll mod) {
  int n = a.size();
  vector<vector<T>> c(n, vector<T>(n));
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      for (int k = 0; k < n; k++) {
        c[i][j] = (c[i][j] +
                   ((a[i][k] * b[k][j]) % mod)) %
                  mod;
      }
    }
  }
  return c;
}
template <typename T>
vector<vector<T>> fpow(vector<vector<T>> &xs,
                       ll p, ll mod) {
  vector<vector<T>> ans(xs.size(),
                        vector<T>(xs.size()));
  for (int i = 0; i < (int)xs.size(); i++)
    ans[i][i] = 1;
  for (auto b = xs; p;
       p >>= 1, b = prod(b, b, mod))
    if (p & 1) ans = prod(ans, b, mod);
  return ans;
}
ll linear_req(vector<vector<ll>> rec,
              vector<ll> first_k, ll n, ll mod) {
  int k = first_k.size();
  if (n < k) {
    return first_k[n];
  }
  ll n2 = n - k + 1;
  rec = fpow(rec, n2, mod);
  ll ret = 0;
  for (int i = 0; i < k; i++) {
    ret = (ret +
           (rec.back()[i] * first_k[i]) % mod) %
          mod;
  }
  return ret;
}
```

7.20 List N elements choose K

Description: Process every possible combination of K elements from N elements, thoose index marked as 1 in the index vector says which elments are choosed at that moment.
Time: $O(\binom{N}{K} \cdot O(process))$

```
void process(vi &index) {
  for (int i = 0; i < len(index); i++) {
    if (index[i])
      cout << i << " \n"[i == len(index) - 1];
  }
}
void n_choose_k(int n, in k) {
  vi index(n);
  fill(index.end() - k, index.end(), 1);
  do {
    process(index);
  } while (next_permutation(all(index)));
}
```

7.21 List primes (Sieve of Eratosthenes)

```
const ll MAXN = 2e5;
vll list_primes(ll n = MAXN) {
  vll ps;
  bitset<MAXN + 1> sieve;
  sieve.set();
  sieve.reset(1);
  for (ll i = 2; i <= n; ++i) {
    if (sieve[i]) ps.push_back(i);
    for (ll j = i * 2; j <= n; j += i) {
      sieve.reset(j);
    }
  }
  return ps;
}
```

7.22 Matrix exponentiation

```
const ll MOD = 1 '000' 000'007;
template <typename T>
vector<vector<T>> prod(vector<vector<T>> &a,
                       vector<vector<T>> &b) {
  int n = len(a);
  vector<vector<T>> c(n, vector<T>(n));
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      for (int k = 0; k < n; k++) {
        c[i][j] = (c[i][j] +
                   ((a[i][k] * b[k][j]) % MOD)) %
                  MOD;
      }
    }
  }
  return c;
}
```

```
template <typename T>
vector<vector<T>> fpow(vector<vector<T>> &xs,
                       ll p) {
  vector<vector<T>> ans(len(xs),
                        vector<T>(len(xs)));
  for (int i = 0; i < len(xs); i++) ans[i][i] = 1;
  auto b = xs;
  while (p) {
    if (p & 1) ans = prod(ans, b);
    p >>= 1;
    b = prod(b, b);
  }
  return ans;
}
```

7.23 NTT integer convolution and exponentiation

Time:
- Convolution $O(N \cdot \log N)$,
- Exponentiation: $O(\log K \cdot N \cdot \log N)$

```
template <int _mod>
struct mint {
  ll expo(ll b, ll e) {
    ll ret = 1;
    while (e) {
      if (e % 2) ret = ret * b % _mod;
      e /= 2, b = b * b % _mod;
    }
    return ret;
  }
  ll inv(ll b) { return expo(b, _mod - 2); }
  using m = mint;
  ll v;
  mint() : v(0) {}
  mint(ll v_) {
    if (v_ >= _mod or v_ <= -_mod) v_ %= _mod;
    if (v_ < 0) v_ += _mod;
    v = v_;
  }
  m &operator+=(const m &a) {
    v += a.v;
    if (v >= _mod) v -= _mod;
    return *this;
  }
  m &operator-=(const m &a) {
    v -= a.v;
    if (v < 0) v += _mod;
    return *this;
  }
  m &operator*=(const m &a) {
    v = v * ll(a.v) % _mod;
    return *this;
  }
  m &operator/=(const m &a) {
    v = v * inv(a.v) % _mod;
    return *this;
  }
  m operator-() { return m(-v); }
  m &operator^=(ll e) {
    if (e < 0) {
      v = inv(v);
      e = -e;
    }
    v = expo(v, e);
    // possivel otimizacao:
    // cuidado com 0^0
    // 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; }
};
const ll MOD1 = 998244353;
const ll MOD2 = 754974721;
const ll MOD3 = 167772161;
template <int _mod>
void ntt(vector<mint<_mod>> &a, bool rev) {
  int n = len(a);
  auto b = a;
  assert(!(n & (n - 1)));
  mint<_mod> g = 1;
  while ((g ^ (_mod / 2)) == 1) g += 1;
  if (rev) g = 1 / g;
  for (int step = n / 2; step; step /= 2) {
    mint<_mod> w = g ^ (_mod / (n / step)),
               wn = 1;
    for (int i = 0; i < n / 2; i += step) {
      for (int j = 0; j < step; j++) {
        auto u = a[2 * i + j],
             v = wn * a[2 * i + j + step];
        b[i + j] = u + v;
        b[i + n / 2 + j] = u - v;
      }
      wn = wn * w;
    }
    swap(a, b);
  }
  if (rev) {
    auto n1 = mint<_mod>(1) / n;
    for (auto &x : a) x *= n1;
  }
}
template <ll _mod>
vector<mint<_mod>> convolution(
    const vector<mint<_mod>> &a,
    const vector<mint<_mod>> &b) {
  vector<mint<_mod>> l(all(a)), r(all(b));
  int N = len(l) + len(r) - 1, n = 1;
  while (n <= N) n *= 2;
  l.resize(n), r.resize(n);
  ntt(l, false), ntt(r, false);
  for (int i = 0; i < n; i++) l[i] *= r[i];
  ntt(l, true);
  l.resize(N);
  // Uncommnent for a boolean convolution :)
  /*
  for (auto& li : l) {
    li.v = min(li.v, 1ll);
  }
  */
  return l;
}
template <ll _mod>
vector<mint<_mod>> poly_exp(
    vector<mint<_mod>> &ps, int k) {
  vector<mint<_mod>> ret(len(ps));
  auto base = ps;
  ret[0] = 1;
  while (k) {
    if (k & 1) ret = convolution(ret, base);
    k >>= 1;
    base = convolution(base, base);
  }
  return ret;
}
```

7.24 NTT integer convolution and exponentiation (2 mods) modules

Description: Computes the convolution between the two polynomials and.
Time: $O(N \log N)$
Warning: This is pure magic !

```
template <int _mod>
struct mint {
  ll expo(ll b, ll e) {
    ll ret = 1;
    while (e) {
      if (e % 2) ret = ret * b % _mod;
      e /= 2, b = b * b % _mod;
    }
    return ret;
  }
  ll inv(ll b) { return expo(b, _mod - 2); }
  using m = mint;
  ll v;
  mint() : v(0) {}
  mint(ll v_) {
    if (v_ >= _mod or v_ <= -_mod) v_ %= _mod;
    if (v_ < 0) v_ += _mod;
    v = v_;
  }
  m &operator+=(const m &a) {
    v += a.v;
    if (v >= _mod) v -= _mod;
    return *this;
  }
  m &operator-=(const m &a) {
    v -= a.v;
    if (v < 0) v += _mod;
    return *this;
  }
  m &operator*=(const m &a) {
    v = v * ll(a.v) % _mod;
    return *this;
  }
  m &operator/=(const m &a) {
    v = v * inv(a.v) % _mod;
    return *this;
  }
  m operator-() { return m(-v); }
  m &operator^=(ll e) {
    if (e < 0) {
      v = inv(v);
      e = -e;
    }
    v = expo(v, e);
    // possivel otimizacao:
    // cuidado com 0^0
    // v = expo(v, e%(p-1));
```

```cpp
    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; }
};
const ll MOD1 = 998244353;
const ll MOD2 = 754974721;
const ll MOD3 = 167772161;
template <int _mod>
void ntt(vector<mint<_mod>> &a, bool rev) {
  int n = len(a);
  auto b = a;
  assert(!(n & (n - 1)));
  mint<_mod> g = 1;
  while ((g ^ (_mod / 2)) == 1) g += 1;
  if (rev) g = 1 / g;
  for (int step = n / 2; step; step /= 2) {
    mint<_mod> w = g ^ (_mod / (n / step)),
             wn = 1;
    for (int i = 0; i < n / 2; i += step) {
      for (int j = 0; j < step; j++) {
        auto u = a[2 * i + j],
             v = wn * a[2 * i + j + step];
        b[i + j] = u + v;
        b[i + n / 2 + j] = u - v;
      }
      wn = wn * w;
    }
    swap(a, b);
  }
  if (rev) {
    auto n1 = mint<_mod>(1) / n;
    for (auto &x : a) x *= n1;
  }
}
tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
  if (!a) return {b, 0, 1};
  auto [g, x, y] = ext_gcd(b % a, a);
  return {g, y - b / a * x, x};
}
template <typename T = ll>
struct crt {
  T a, m;
  crt() : a(0), m(1) {}
  crt(T a_, T m_) : a(a_), m(m_) {}
  crt operator*(crt C) {
    auto [g, x, y] = ext_gcd(m, C.m);
    if ((a - C.a) % g != 0) a = -1;
    if (a == -1 or C.a == -1) return crt(-1, 0);
    T lcm = m / g * C.m;
    T ans =
        a + (x * (C.a - a) / g % (C.m / g)) * m;
    return crt((ans % lcm + lcm) % lcm, lcm);
  }
};
template <typename T = ll>
struct Congruence {
  T a, m;
};
template <typename T = ll>
T chinese_remainder_theorem(
    const vector<Congruence<T>> &equations) {
  crt<T> ans;
  for (auto &[a_, m_] : equations) {
    ans = ans * crt<T>(a_, m_);
  }
  return ans.a;
}
#define int long long
template <ll m1, ll m2>
vll merge_two_mods(const vector<mint<m1>> &a,
                   const vector<mint<m2>> &b) {
  int n = len(a);
  vll ans(n);
  for (int i = 0; i < n; i++) {
    auto cur = crt<ll>();
    auto ai = a[i].v;
    auto bi = b[i].v;
    cur = cur * crt<ll>(ai, m1);
    cur = cur * crt<ll>(bi, m2);
    ans[i] = cur.a;
  }
  return ans;
}
vll convolution_2mods(const vll &a,
                      const vll &b) {
  vector<mint<MOD1>> l(all(a)), r(all(b));
  int N = len(l) + len(r) - 1, n = 1;
  while (n <= N) n *= 2;
  l.resize(n), r.resize(n);
  ntt(l, false), ntt(r, false);
  for (int i = 0; i < n; i++) l[i] *= r[i];
  ntt(l, true);
  l.resize(N);
  vector<mint<MOD2>> l2(all(a)), r2(all(b));
  l2.resize(n), r2.resize(n);
  ntt(l2, false), ntt(r2, false);
  rep(i, 0, n) l2[i] *= r2[i];
  ntt(l2, true);
  l2.resize(N);
  return merge_two_mods(l, l2);
}
vll poly_exp(const vll &xs, ll k) {
  vll ret(len(xs));
  ret[0] = 1;
  auto base = xs;
  while (k) {
    if (k & 1) ret = convolution_2mods(ret, base);
    k >>= 1;
    base = convolution_2mods(base, base);
  }
  return ret;
}
```

7.25 Polyominoes

Usage: *buildPolyominoes*(x) creates every polyomino until size x, and put it in *polyominoes*$[x]$, access *polyomino.v* to find the vector of pairs representing the coordinates of each piece, considering that the polyomino was 'rooted' in coordinate $(0, 0)$.

Warning: note that when acessing *polyominoes*$[x]$ only the first x coordinates are valid.

```cpp
const int MAXP = 10;
using pii = pair<int, int>;
// This implementation considers the rotations as
// distinct
//                0, 10, 10+9, 10+9+8...
int pos[11] = {0,  10, 19, 27, 34, 40,
               45, 49, 52, 54, 55};
struct Polyominoes {
  pii v[MAXP];
  ll id;
  int n;
  Polyominoes() {
    n = 1;
    v[0] = {0, 0};
    normalize();
  }
  pii &operator[](int i) { return v[i]; }
  bool add(int a, int b) {
    for (int i = 0; i < n; i++)
      if (v[i].first == a and v[i].second == b)
        return false;
    v[n++] = pii(a, b);
    normalize();
    return true;
  }
  void normalize() {
    int mnx = 100, mny = 100;
    for (int i = 0; i < n; i++)
      mnx = min(mnx, v[i].first),
      mny = min(mny, v[i].second);
    id = 0;
    for (int i = 0; i < n; i++) {
      v[i].first -= mnx, v[i].second -= mny;
      id |= (1LL << (pos[v[i].first] +
                     v[i].second));
    }
  }
};
vector<Polyominoes> polyominoes[MAXP + 1];
void buildPolyominoes(int mxN = 10) {
  vector<pair<int, int>> dt(
      {{1, 0}, {-1, 0}, {0, -1}, {0, 1}});
  for (int i = 0; i <= mxN; i++)
    polyominoes[i].clear();
  Polyominoes init;
  queue<Polyominoes> q;
  unordered_set<int64_t> used;
  q.push(init);
  used.insert(init.id);
  while (!q.empty()) {
    Polyominoes u = q.front();
    q.pop();
    polyominoes[u.n].push_back(u);
    if (u.n == mxN) continue;
    for (int i = 0; i < u.n; i++) {
      for (auto [dx, dy] : dt) {
        Polyominoes to = u;
        bool ok = to.add(to[i].first + dx,
                         to[i].second + dy);
        if (ok and !used.count(to.id)) {
          q.push(to);
          used.insert(to.id);
        }
      }
```

```
    }
  }
}
```

8 Primitives

8.1 Bigint

```
const int maxn = 1e2 + 14, lg = 15;
const int base = 1000000000;
const int base_digits = 9;
struct bigint {
  vi a;
  int sign;
  int size() {
    if (a.empty()) return 0;
    int ans = (a.size() - 1) * base_digits;
    int ca = a.back();
    while (ca) ans++, ca /= 10;
    return ans;
  }
  bigint operator^(const bigint &v) {
    bigint ans = 1, a = *this, b = v;
    while (!b.isZero()) {
      if (b % 2) ans *= a;
      a *= a, b /= 2;
    }
    return ans;
  }
  string to_string() {
    stringstream ss;
    ss << *this;
    string s;
    ss >> s;
    return s;
  }
  int sumof() {
    string s = to_string();
    int ans = 0;
    for (auto c : s) ans += c - '0';
    return ans;
  }
  /*</arpa>*/
  bigint() : sign(1) {}
  bigint(long long v) { *this = v; }
  bigint(const string &s) { read(s); }
  void operator=(const bigint &v) {
    sign = v.sign;
    a = v.a;
  }
  void operator=(long long v) {
    sign = 1;
    a.clear();
    if (v < 0) sign = -1, v = -v;
    for (; v > 0; v = v / base)
      a.push_back(v % base);
  }
  bigint operator+(const bigint &v) const {
    if (sign == v.sign) {
      bigint res = v;
      for (int i = 0, carry = 0;
           i < (int)max(a.size(), v.a.size()) ||
           carry;
           ++i) {
        if (i == (int)res.a.size())
          res.a.push_back(0);
        res.a[i] +=
            carry +
            (i < (int)a.size() ? a[i] : 0);
        carry = res.a[i] >= base;
        if (carry) res.a[i] -= base;
      }
      return res;
    }
    return *this - (-v);
  }
  bigint operator-(const bigint &v) const {
    if (sign == v.sign) {
      if (abs() >= v.abs()) {
        bigint res = *this;
        for (int i = 0, carry = 0;
             i < (int)v.a.size() || carry; ++i) {
          res.a[i] -=
              carry +
              (i < (int)v.a.size() ? v.a[i] : 0);
          carry = res.a[i] < 0;
          if (carry) res.a[i] += base;
        }
        res.trim();
        return res;
      }
      return -(v - *this);
    }
    return *this + (-v);
  }
  void operator*=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = 0, carry = 0;
         i < (int)a.size() || carry; ++i) {
      if (i == (int)a.size()) a.push_back(0);
      long long cur = a[i] * (long long)v + carry;
      carry = (int)(cur / base);
      a[i] = (int)(cur % base);
      // asm("divl %%ecx" : "=a"(carry),
      // "=d"(a[i]) : "A"(cur), "c"(base));
    }
    trim();
  }
  bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
    return res;
  }
  void operator*=(long long v) {
    if (v < 0) sign = -sign, v = -v;
    if (v > base) {
      *this = *this * (v / base) * base +
              *this * (v % base);
      return;
    }
    for (int i = 0, carry = 0;
         i < (int)a.size() || carry; ++i) {
      if (i == (int)a.size()) a.push_back(0);
      long long cur = a[i] * (long long)v + carry;
      carry = (int)(cur / base);
      a[i] = (int)(cur % base);
      // asm("divl %%ecx" : "=a"(carry),
      // "=d"(a[i]) : "A"(cur), "c"(base));
    }
    trim();
  }
  bigint operator*(long long v) const {
    bigint res = *this;
    res *= v;
    return res;
  }
  friend pair<bigint, bigint> divmod(
      const bigint &a1, const bigint &b1) {
    int norm = base / (b1.a.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.a.resize(a.a.size());
    for (int i = a.a.size() - 1; i >= 0; i--) {
      r *= base;
      r += a.a[i];
      int s1 = r.a.size() <= b.a.size()
                   ? 0
                   : r.a[b.a.size()];
      int s2 = r.a.size() <= b.a.size() - 1
                   ? 0
                   : r.a[b.a.size() - 1];
      int d = ((long long)base * s1 + s2) /
              b.a.back();
      r -= b * d;
      while (r < 0) r += b, --d;
      q.a[i] = d;
    }
    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
  }
  bigint operator/(const bigint &v) const {
    return divmod(*this, v).first;
  }
  bigint operator%(const bigint &v) const {
    return divmod(*this, v).second;
  }
  void operator/=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = (int)a.size() - 1, rem = 0;
         i >= 0; --i) {
      long long cur =
          a[i] + rem * (long long)base;
      a[i] = (int)(cur / v);
      rem = (int)(cur % v);
    }
    trim();
  }
  bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
  }
  int operator%(int v) const {
    if (v < 0) v = -v;
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i)
      m = (a[i] + m * (long long)base) % v;
    return m * sign;
  }
  void operator+=(const bigint &v) {
    *this = *this + v;
  }
  void operator-=(const bigint &v) {
    *this = *this - v;
  }
  void operator*=(const bigint &v) {
```

```
    *this = *this * v;
  }
  void operator/=(const bigint &v) {
    *this = *this / v;
  }
  bool operator<(const bigint &v) const {
    if (sign != v.sign) return sign < v.sign;
    if (a.size() != v.a.size())
      return a.size() * sign <
             v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
      if (a[i] != v.a[i])
        return a[i] * sign < v.a[i] * sign;
    return false;
  }
  bool operator>(const bigint &v) const {
    return v < *this;
  }
  bool operator<=(const bigint &v) const {
    return !(v < *this);
  }
  bool operator>=(const bigint &v) const {
    return !(*this < v);
  }
  bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
  }
  bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
  }
  void trim() {
    while (!a.empty() && !a.back()) a.pop_back();
    if (a.empty()) sign = 1;
  }
  bool isZero() const {
    return a.empty() || (a.size() == 1 && !a[0]);
  }
  bigint operator-() const {
    bigint res = *this;
    res.sign = -sign;
    return res;
  }
  bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
  }
  long long longValue() const {
    long long res = 0;
    for (int i = a.size() - 1; i >= 0; i--)
      res = res * base + a[i];
    return res * sign;
  }
  friend bigint gcd(const bigint &a,
                    const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
  }
  friend bigint lcm(const bigint &a,
                    const bigint &b) {
    return a / gcd(a, b) * b;
  }
  void read(const string &s) {
    sign = 1;
    a.clear();
    int pos = 0;
    while (pos < (int)s.size() &&
        (s[pos] == '-' || s[pos] == '+')) {
      if (s[pos] == '-') sign = -sign;
      ++pos;
    }
    for (int i = s.size() - 1; i >= pos;
         i -= base_digits) {
      int x = 0;
      for (int j = max(pos, i - base_digits + 1);
           j <= i; j++)
        x = x * 10 + s[j] - '0';
      a.push_back(x);
    }
    trim();
  }
  friend istream &operator>>(istream &stream,
                             bigint &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
  }
  friend ostream &operator<<(ostream &stream,
                             const bigint &v) {
    if (v.sign == -1) stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back());
    for (int i = (int)v.a.size() - 2; i >= 0; --i)
      stream << setw(base_digits) << setfill('0')
             << v.a[i];
    return stream;
  }
  static vector<int> convert_base(
      const vector<int> &a, int old_digits,
      int new_digits) {
    vector<long long> p(
        max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int)p.size(); i++)
      p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int)a.size(); i++) {
      cur += a[i] * p[cur_digits];
      cur_digits += old_digits;
      while (cur_digits >= new_digits) {
        res.push_back(int(cur % p[new_digits]));
        cur /= p[new_digits];
        cur_digits -= new_digits;
      }
    }
    res.push_back((int)cur);
    while (!res.empty() && !res.back())
      res.pop_back();
    return res;
  }
  typedef vector<long long> vll;
  static vll karatsubaMultiply(const vll &a,
                               const vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
      for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
          res[i + j] += a[i] * b[j];
      return res;
    }
    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());
    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);
    for (int i = 0; i < k; i++) a2[i] += a1[i];
    for (int i = 0; i < k; i++) b2[i] += b1[i];
    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int)a1b1.size(); i++)
      r[i] -= a1b1[i];
    for (int i = 0; i < (int)a2b2.size(); i++)
      r[i] -= a2b2[i];
    for (int i = 0; i < (int)r.size(); i++)
      res[i + k] += r[i];
    for (int i = 0; i < (int)a1b1.size(); i++)
      res[i] += a1b1[i];
    for (int i = 0; i < (int)a2b2.size(); i++)
      res[i + n] += a2b2[i];
    return res;
  }
  bigint operator*(const bigint &v) const {
    vector<int> a6 =
        convert_base(this->a, base_digits, 6);
    vector<int> b6 =
        convert_base(v.a, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size()) a.push_back(0);
    while (b.size() < a.size()) b.push_back(0);
    while (a.size() & (a.size() - 1))
      a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int)c.size();
         i++) {
      long long cur = c[i] + carry;
      res.a.push_back((int)(cur % 1000000));
      carry = (int)(cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
  }
};
```

8.2 Integer Mod

```
const ll MOD = 1'000'000'000 + 7;
template <ll _mod = MOD>
struct mint {
  ll value;
  static const ll MOD_value = _mod;
  mint(ll v = 0) {
    value = v % _mod;
    if (value < 0) value += _mod;
  }
  mint(ll a, ll b) : value(0) {
    *this += a;
    *this /= b;
  }
  mint &operator+=(mint const &b) {
    value += b.value;
    if (value >= _mod) value -= _mod;
```

```
    return *this;
  }
  mint &operator-=(mint const &b) {
    value -= b.value;
    if (value < 0) value += _mod;
    return *this;
  }
  mint &operator*=(mint const &b) {
    value = (ll)value * b.value % _mod;
    return *this;
  }
  friend mint mexp(mint a, ll e) {
    mint res = 1;
    while (e) {
      if (e & 1) res *= a;
      a *= a;
      e >>= 1;
    }
    return res;
  }
  friend mint inverse(mint a) {
    return mexp(a, _mod - 2);
  }
  mint &operator/=(mint const &b) {
    return *this *= inverse(b);
  }
  friend mint operator+(mint a, mint const b) {
    return a += b;
  }
  mint operator++(int) {
    return this->value = (this->value + 1) % _mod;
  }
  mint operator++() {
    return this->value = (this->value + 1) % _mod;
  }
  friend mint operator-(mint a, mint const b) {
    return a -= b;
  }
  friend mint operator-(mint const a) {
    return 0 - a;
  }
  mint operator--(int) {
    return this->value =
           (this->value - 1 + _mod) % _mod;
  }
  mint operator--() {
    return this->value =
           (this->value - 1 + _mod) % _mod;
  }
  friend mint operator*(mint a, mint const b) {
    return a *= b;
  }
  friend mint operator/(mint a, mint const b) {
    return a /= b;
  }
  friend std::ostream &operator<<(
      std::ostream &os, mint const &a) {
    return os << a.value;
  }
  friend bool operator==(mint const &a,
                         mint const &b) {
    return a.value == b.value;
  }
  friend bool operator!=(mint const &a,
                         mint const &b) {
    return a.value != b.value;
  }
};
```

8.3 Matrix

```
template <typename T>
struct Matrix {
  vector<vector<T>> d;
  Matrix() : Matrix(0) {}
  Matrix(int n) : Matrix(n, n) {}
  Matrix(int n, int m)
      : Matrix(
            vector<vector<T>>(n, vector<T>(m))) {}
  Matrix(const vector<vector<T>> &v) : d(v) {}
  constexpr int n() const {
    return (int)d.size();
  }
  constexpr int m() const {
    return n() ? (int)d[0].size() : 0;
  }
  void rotate() { *this = rotated(); }
  Matrix<T> rotated() const {
    Matrix<T> res(m(), n());
    for (int i = 0; i < m(); i++) {
      for (int j = 0; j < n(); j++) {
        res[i][j] = d[n() - j - 1][i];
      }
    }
    return res;
  }
  Matrix<T> pow(int power) const {
    assert(n() == m());
    auto res = Matrix<T>::identity(n());
    auto b = *this;
    while (power) {
      if (power & 1) res *= b;
      b *= b;
      power >>= 1;
    }
    return res;
  }
  Matrix<T> submatrix(int start_i, int start_j,
                      int rows = INT_MAX,
                      int cols = INT_MAX) const {
    rows = min(rows, n() - start_i);
    cols = min(cols, m() - start_j);
    if (rows <= 0 or cols <= 0) return {};
    Matrix<T> res(rows, cols);
    for (int i = 0; i < rows; i++)
      for (int j = 0; j < cols; j++)
        res[i][j] = d[i + start_i][j + start_j];
    return res;
  }
  Matrix<T> translated(int x, int y) const {
    Matrix<T> res(n(), m());
    for (int i = 0; i < n(); i++) {
      for (int j = 0; j < m(); j++) {
        if (i + x < 0 or i + x >= n() or
            j + y < 0 or j + y >= m())
          continue;
        res[i + x][j + y] = d[i][j];
      }
    }
    return res;
  }
  static Matrix<T> identity(int n) {
    Matrix<T> res(n);
    for (int i = 0; i < n; i++) res[i][i] = 1;
    return res;
  }
  vector<T> &operator[](int i) { return d[i]; }
  const vector<T> &operator[](int i) const {
    return d[i];
  }
  Matrix<T> &operator+=(T value) {
    for (auto &row : d) {
      for (auto &x : row) x += value;
    }
    return *this;
  }
  Matrix<T> operator+(T value) const {
    auto res = *this;
    for (auto &row : res) {
      for (auto &x : row) x = x + value;
    }
    return res;
  }
  Matrix<T> &operator-=(T value) {
    for (auto &row : d) {
      for (auto &x : row) x -= value;
    }
    return *this;
  }
  Matrix<T> operator-(T value) const {
    auto res = *this;
    for (auto &row : res) {
      for (auto &x : row) x = x - value;
    }
    return res;
  }
  Matrix<T> &operator*=(T value) {
    for (auto &row : d) {
      for (auto &x : row) x *= value;
    }
    return *this;
  }
  Matrix<T> operator*(T value) const {
    auto res = *this;
    for (auto &row : res) {
      for (auto &x : row) x = x * value;
    }
    return res;
  }
  Matrix<T> &operator/=(T value) {
    for (auto &row : d) {
      for (auto &x : row) x /= value;
    }
    return *this;
  }
  Matrix<T> operator/(T value) const {
    auto res = *this;
    for (auto &row : res) {
      for (auto &x : row) x = x / value;
    }
    return res;
  }
  Matrix<T> &operator+=(const Matrix<T> &o) {
    assert(n() == o.n() and m() == o.m());
    for (int i = 0; i < n(); i++) {
      for (int j = 0; j < m(); j++) {
        d[i][j] += o[i][j];
      }
    }
    return *this;
  }
  Matrix<T> operator+(const Matrix<T> &o) const {
    assert(n() == o.n() and m() == o.m());
    auto res = *this;
    for (int i = 0; i < n(); i++) {
      for (int j = 0; j < m(); j++) {
        res[i][j] = res[i][j] + o[i][j];
```

```
      }
    }
    return res;
  }
  Matrix<T> &operator-=(const Matrix<T> &o) {
    assert(n() == o.n() and m() == o.m());
    for (int i = 0; i < n(); i++) {
      for (int j = 0; j < m(); j++) {
        d[i][j] -= o[i][j];
      }
    }
    return *this;
  }
  Matrix<T> operator-(const Matrix<T> &o) const {
    assert(n() == o.n() and m() == o.m());
    auto res = *this;
    for (int i = 0; i < n(); i++) {
      for (int j = 0; j < m(); j++) {
        res[i][j] = res[i][j] - o[i][j];
      }
    }
    return res;
  }
  Matrix<T> &operator*=(const Matrix<T> &o) {
    *this = *this * o;
    return *this;
  }
  Matrix<T> operator*(const Matrix<T> &o) const {
    assert(m() == o.n());
    Matrix<T> res(n(), o.m());
    for (int i = 0; i < res.n(); i++) {
      for (int j = 0; j < res.m(); j++) {
        auto &x = res[i][j];
        for (int k = 0; k < m(); k++) {
          x += (d[i][k] * o[k][j]);
        }
      }
    }
    return res;
  }
  friend istream &operator>>(istream &is,
                             Matrix<T> &mat) {
    for (auto &row : mat)
      for (auto &x : row) is >> x;
    return is;
  }
  friend ostream &operator<<(
      ostream &os, const Matrix<T> &mat) {
    bool frow = 1;
    for (auto &row : mat) {
      if (not frow) os << '\n';
      bool first = 1;
      for (auto &x : row) {
        if (not first) os << ' ';
        os << x;
        first = 0;
      }
      frow = 0;
    }
    return os;
  }
  auto begin() { return d.begin(); }
  auto end() { return d.end(); }
  auto rbegin() { return d.rbegin(); }
  auto rend() { return d.rend(); }
  auto begin() const { return d.begin(); }
  auto end() const { return d.end(); }
  auto rbegin() const { return d.rbegin(); }
  auto rend() const { return d.rend(); }
};
```

9 Strings

9.1 Count distinct anagrams

```
const ll MOD = 1e9 + 7;
const int maxn = 1e6;
vll fs(maxn + 1);
void precompute() {
  fs[0] = 1;
  for (ll i = 1; i <= maxn; i++) {
    fs[i] = (fs[i - 1] * i) % MOD;
  }
}
ll fpow(ll a, int n, ll mod = LLONG_MAX) {
  if (n == 0) return 1;
  if (n == 1) return a;
  ll x = fpow(a, n / 2, mod) % mod;
  return ((x * x) % mod * (n & 1 ? a : 1ll)) %
         mod;
}
ll distinctAnagrams(const string &s) {
  precompute();
  vi hist('z' - 'a' + 1, 0);
  for (auto &c : s) hist[c - 'a']++;
  ll ans = fs[len(s)];
  for (auto &q : hist) {
    ans = (ans * fpow(fs[q], MOD - 2, MOD)) % MOD;
  }
  return ans;
}
```

9.2 Double hash range query

```
using ll = long long;
using vll = vector<ll>;
using pll = pair<ll, ll>;
const int MAXN(1 '000' 000);
const ll MOD = 1000027957;
const ll MOD2 = 1000015187;
const ll P = 31;
ll p[MAXN + 1], p2[MAXN + 1];
void precompute() {
  p[0] = p2[0] = 1;
  for (int i = 1; i <= MAXN; i++)
    p[i] = (P * p[i - 1]) % MOD,
    p2[i] = (P * p2[i - 1]) % MOD2;
}
struct Hash {
  int n;
  vll h, h2, hi, hi2;
  Hash() {}
  Hash(const string &s)
      : n(s.size()), h(n), h2(n), hi(n), hi2(n) {
    h[0] = h2[0] = s[0];
    for (int i = 1; i < n; i++)
      h[i] = (s[i] + h[i - 1] * P) % MOD,
      h2[i] = (s[i] + h2[i - 1] * P) % MOD2;
    hi[n - 1] = hi2[n - 1] = s[n - 1];
    for (int i = n - 2; i >= 0; i--)
      hi[i] = (s[i] + hi[i + 1] * P) % MOD,
      hi2[i] = (s[i] + hi2[i + 1] * P) % MOD2;
  }
  pll query(int l, int r) {
    ll hash =
        (h[r] -
         (l ? h[l - 1] * p[r - l + 1] % MOD : 0));
    ll hash2 =
        (h2[r] -
         (l ? h2[l - 1] * p2[r - l + 1] % MOD2
            : 0));
    return {(hash < 0 ? hash + MOD : hash),
            (hash2 < 0 ? hash2 + MOD2 : hash2)};
  }
  pll query_inv(int l, int r) {
    ll hash = (hi[l] -
               (r + 1 < n ? hi[r + 1] *
                                p[r - l + 1] % MOD
                          : 0));
    ll hash2 =
        (hi2[l] -
         (r + 1 < n
              ? hi2[r + 1] * p2[r - l + 1] % MOD2
              : 0));
    return {(hash < 0 ? hash + MOD : hash),
            (hash2 < 0 ? hash2 + MOD2 : hash2)};
  }
};
```

9.3 Hash range query

```
const ll P = 31;
const ll MOD = 1e9 + 9;
const int MAXN(1e6);
ll ppow[MAXN + 1];
void pre_calc() {
  ppow[0] = 1;
  for (int i = 1; i <= MAXN; i++)
    ppow[i] = (ppow[i - 1] * P) % MOD;
}
struct Hash {
  int n;
  vll h, hi;
  Hash(const string &s)
      : n(s.size()), h(n), hi(n) {
    h[0] = s[0];
    hi[n - 1] = s[n - 1];
    for (int i = 1; i < n; i++) {
      h[i] = (s[i] + h[i - 1] * P) % MOD;
      hi[n - i - 1] =
          (s[n - i - 1] + hi[n - i - 1] * P) %
          MOD;
    }
  }
  ll qry(int l, int r) {
    ll hash =
        (h[r] -
         (l ? h[l - 1] * ppow[r - l + 1] % MOD
            : 0));
    return hash < 0 ? hash + MOD : hash;
  }
  ll qry_inv(int l, int r) {
    ll hash =
        (hi[l] -
         (r + 1 < n
              ? hi[r + 1] * ppow[r - l + 1] % MOD
              : 0));
    return hash < 0 ? hash + MOD : hash;
  }
};
```

9.4 Hash unsigned long long $2^{64} - 1$

Description: Arithmetic mod $2^{64} - 1$. 2x slower than mod 2^{64} and more code, but works on evil test data (e.g. Thue-Morse, where ABBA... and BAAB... of length 2^{10} hash the same mod 2^{64}).
"typedef ull H;" instead if you think test data is random.

```
typedef uint64_t ull;
struct H {
  ull x;
  H(ull x = 0) : x(x) {}
  H operator+(H o) {
    return x + o.x + (x + o.x < x);
  }
  H operator-(H o) { return *this + ~o.x; }
  H operator*(H o) {
    auto m = (__uint128_t)x * o.x;
    return H((ull)m) + (ull)(m >> 64);
  }
  ull get() const { return x + !~x; }
  bool operator==(H o) const {
    return get() == o.get();
  }
  bool operator<(H o) const {
    return get() < o.get();
  }
};
static const H C =
    (long long)1e11 +
    3;  // (order ~ 3e9; random also ok)
struct Hash {
  int n;
  vector<H> ha, pw;
  Hash(string &str)
      : n(str.size()),
        ha((int)str.size() + 1),
        pw(ha) {
    pw[0] = 1;
    for (int i = 0; i < (int)str.size(); i++)
      ha[i + 1] = ha[i] * C + str[i],
             pw[i + 1] = pw[i] * C;
  }
  H query(int a, int b) {  // hash [a, b]
    b++;
    return ha[b] - ha[a] * pw[b - a];
  }
};
vector<H> getHashes(string &str, int length) {
  if ((int)str.size() < length) return {};
  H h = 0, pw = 1;
  for (int i = 0; i < length; i++)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
  for (int i = length; i < (int)str.size(); i++)
    ret.push_back(h = h * C + str[i] -
                      pw * str[i - length]);
  return ret;
}
H hashString(string &s) {
  H h{};
  for (char c : s) h = h * C + c;
  return h;
}
```

9.5 K-th digit in digit string

Description: Find the k-th digit in a *digit string*, only works for $1 <= k <= 10^{18}$!
Time: precompute $O(1)$, query $O(1)$

```
using vull = vector<ull>;
vull pow10;
vector<array<ull, 4>> memo;
void precompute(int maxpow = 18) {
  ull qtd = 1;
  ull start = 1;
  ull end = 9;
  ull curlenght = 9;
  ull startstr = 1;
  ull endstr = 9;
  for (ull i = 0, j = 1ll; (int)i < maxpow;
       i++, j *= 10ll)
    pow10.eb(j);
  for (ull i = 0; i < maxpow - 1ull; i++) {
    memo.push_back(
        {start, end, startstr, endstr});
    start = end + 1ll;
    end = end + (9ll * pow10[qtd]);
    curlenght = end - start + 1ull;
    qtd++;
    startstr = endstr + 1ull;
    endstr =
        (endstr + 1ull) + (curlenght)*qtd - 1ull;
  }
}
char kthDigit(ull k) {
  int qtd = 1;
  for (auto [s, e, ss, es] : memo) {
    if (k >= ss and k <= es) {
      ull pos = k - ss;
      ull index = pos / qtd;
      ull nmr = s + index;
      int i = k - ss - qtd * index;
      return ((nmr / pow10[qtd - i - 1]) % 10) +
             '0';
    }
    qtd++;
  }
  return 'X';
}
```

9.6 Longest Palindrome Substring (Manacher)

Description: Finds the longest palindrome substring, manacher returns a vector where the i-th position is how much is possible to grow the string to the left and the right of i and keep it a palindrome.
Time: $O(N)$

```
vi manacher(const string &s) {
  int n = len(s) - 2;
  vi p(n + 2);
  int l = 1, r = 1;
  for (int i = 1; i <= n; i++) {
    p[i] = max(0, min(r - i, p[l + (r - i)]));
    while (s[i - p[i]] == s[i + p[i]]) p[i]++;
    if (i + p[i] > r) l = i - p[i], r = i + p[i];
    p[i]--;
  }
  return p;
}
string longest_palindrome(const string &s) {
  string t("$#");
  for (auto c : s)
    t.push_back(c), t.push_back('#');
  t.push_back('^');
  vi xs = manacher(t);
  int mpos = max_element(all(xs)) - xs.begin();
  string p;
  for (int k = xs[mpos], i = mpos - k;
       i <= mpos + k; i++)
    if (t[i] != '#') p.push_back(t[i]);
  return p;
}
```

9.7 Longest palindrome

```
string longest_palindrome(const string &s) {
  int n = (int)s.size();
  vector<array<int, 2>> dp(n);
  pii odd(0, -1), even(0, -1);
  pii ans;
  for (int i = 0; i < n; i++) {
    int k = 0;
    if (i > odd.second)
      k = 1;
    else
      k = min(dp[odd.first + odd.second - i][0],
              odd.second - i + 1);
    while (i - k >= 0 and i + k < n and
           s[i - k] == s[i + k])
      k++;
    dp[i][0] = k--;
    if (i + k > odd.second) odd = {i - k, i + k};
    if (2 * dp[i][0] - 1 > ans.second)
      ans = {i - k, 2 * dp[i][0] - 1};
    k = 0;
    if (i <= even.second)
      k = min(
        dp[even.first + even.second - i + 1][1],
        even.second - i + 1);
    while (i - k - 1 >= 0 and i + k < n and
           s[i - k - 1] == s[i + k])
      k++;
    dp[i][1] = k--;
    if (i + k > even.second)
      even = {i - k - 1, i + k};
    if (2 * dp[i][1] > ans.second)
      ans = {i - k - 1, 2 * dp[i][1]};
  }
  return s.substr(ans.first, ans.second);
}
```

9.8 Rabin-Karp

```
size_t rabin_karp(const string &s,
                  const string &p) {
  if (s.size() < p.size()) return 0;
  auto n = s.size(), m = p.size();
  const ll p1 = 31, p2 = 29, q1 = 1e9 + 7,
           q2 = 1e9 + 9;
  const ll p1_1 = fpow(p1, q1 - 2, q1),
           p1_2 = fpow(p1, m - 1, q1);
  const ll p2_1 = fpow(p2, q2 - 2, q2),
           p2_2 = fpow(p2, m - 1, q2);
  pair<ll, ll> hs, hp;
  for (int i = (int)m - 1; ~i; --i) {
```

```
    hs.first = (hs.first * p1) % q1;
    hs.first = (hs.first + (s[i] - 'a' + 1)) % q1;
    hs.second = (hs.second * p2) % q2;
    hs.second =
        (hs.second + (s[i] - 'a' + 1)) % q2;
    hp.first = (hp.first * p1) % q1;
    hp.first = (hp.first + (p[i] - 'a' + 1)) % q1;
    hp.second = (hp.second * p2) % q2;
    hp.second =
        (hp.second + (p[i] - 'a' + 1)) % q2;
  }
  size_t occ = 0;
  for (size_t i = 0; i < n - m; i++) {
    occ += (hs == hp);
    int fi = s[i] - 'a' + 1;
    int fm = s[i + m] - 'a' + 1;
    hs.first = (hs.first - fi + q1) % q1;
    hs.first = (hs.first * p1_1) % q1;
    hs.first = (hs.first + fm * p1_2) % q1;
    hs.second = (hs.second - fi + q2) % q2;
    hs.second = (hs.second * p2_1) % q2;
    hs.second = (hs.second + fm * p2_2) % q2;
  }
  occ += hs == hp;
  return occ;
}
```

9.9 Suffix array

```
vector<int> sort_cyclic_shifts(string const &s) {
  int n = s.size();
  const int alphabet = 128;
  vector<int> p(n), c(n),
      cnt(max(alphabet, n), 0);
  for (int i = 0; i < n; i++) cnt[s[i]]++;
  for (int i = 1; i < alphabet; i++)
    cnt[i] += cnt[i - 1];
  for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
  c[p[0]] = 0;
  int classes = 1;
  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i - 1]]) classes++;
    c[p[i]] = classes - 1;
  }
  vector<int> pn(n), cn(n);
  for (int h = 0; (1 << h) < n; ++h) {
    for (int i = 0; i < n; i++) {
      pn[i] = p[i] - (1 << h);
      if (pn[i] < 0) pn[i] += n;
    }
    fill(cnt.begin(), cnt.begin() + classes, 0);
    for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
    for (int i = 1; i < classes; i++)
      cnt[i] += cnt[i - 1];
    for (int i = n - 1; i >= 0; i--)
      p[--cnt[c[pn[i]]]] = pn[i];
    cn[p[0]] = 0;
    classes = 1;
    for (int i = 1; i < n; i++) {
      pair<int, int> cur = {
          c[p[i]], c[(p[i] + (1 << h)) % n]};
      pair<int, int> prev = {
          c[p[i - 1]],
          c[(p[i - 1] + (1 << h)) % n]};
      if (cur != prev) ++classes;
      cn[p[i]] = classes - 1;
    }
    c.swap(cn);
  }
  return p;
}
vector<int> suffix_array(string s) {
  s += "$";
  vector<int> p = sort_cyclic_shifts(s);
  p.erase(p.begin());
  return p;
}
```

9.10 Suffix automaton

```
struct state {
  int len, link, cnt, firstpos;
  // this can be optimized using a vector with the
  // alphabet size
  map<char, int> next;
  vi inv_link;
};
struct SuffixAutomaton {
  vector<state> st;
  int sz = 0;
  int last;
  vc cloned;
  SuffixAutomaton(const string &s, int maxlen)
      : st(maxlen * 2), cloned(maxlen * 2) {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
    for (auto &c : s) add_char(c);
    // precompute for count occurences
    for (int i = 1; i < sz; i++) {
      st[i].cnt = !cloned[i];
    }
    vector<pair<state, int>> aux;
    for (int i = 0; i < sz; i++) {
      aux.push_back({st[i], i});
    }
    sort(all(aux), [](const pair<state, int> &a,
                      const pair<state, int> &b) {
      return a.fst.len > b.fst.len;
    });
    for (auto &[stt, id] : aux) {
      if (stt.link != -1) {
        st[stt.link].cnt += st[id].cnt;
      }
    }
    // for find every occurende position
    for (int v = 1; v < sz; v++) {
      st[st[v].link].inv_link.push_back(v);
    }
  }
  void add_char(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    st[cur].firstpos = st[cur].len - 1;
    int p = last;
    // follow the suffix link until find a
    // transition to c
    while (p != -1 and !st[p].next.count(c)) {
      st[p].next[c] = cur;
      p = st[p].link;
    }
    // there was no transition to c so create and
    // leave
    if (p == -1) {
      st[cur].link = 0;
      last = cur;
      return;
    }
    int q = st[p].next[c];
    if (st[p].len + 1 == st[q].len) {
      st[cur].link = q;
    } else {
      int clone = sz++;
      cloned[clone] = true;
      st[clone].len = st[p].len + 1;
      st[clone].next = st[q].next;
      st[clone].link = st[q].link;
      st[clone].firstpos = st[q].firstpos;
      while (p != -1 and st[p].next[c] == q) {
        st[p].next[c] = clone;
        p = st[p].link;
      }
      st[q].link = st[cur].link = clone;
    }
    last = cur;
  }
  bool checkOccurrence(
      const string &t) {  // O(len(t))
    int cur = 0;
    for (auto &c : t) {
      if (!st[cur].next.count(c)) return false;
      cur = st[cur].next[c];
    }
    return true;
  }
  ll totalSubstrings() {  // distinct, O(len(s))
    ll tot = 0;
    for (int i = 1; i < sz; i++) {
      tot += st[i].len - st[st[i].link].len;
    }
    return tot;
  }
  // count occurences of a given string t
  int countOccurences(const string &t) {
    int cur = 0;
    for (auto &c : t) {
      if (!st[cur].next.count(c)) return 0;
      cur = st[cur].next[c];
    }
    return st[cur].cnt;
  }
  // find the first index where t appears a
  // substring O(len(t))
  int firstOccurence(const string &t) {
    int cur = 0;
    for (auto c : t) {
      if (!st[cur].next.count(c)) return -1;
      cur = st[cur].next[c];
    }
    return st[cur].firstpos - len(t) + 1;
  }
  vi everyOccurence(const string &t) {
    int cur = 0;
    for (auto c : t) {
      if (!st[cur].next.count(c)) return {};
```

```
      cur = st[cur].next[c];
    }
    vi ans;
    getEveryOccurence(cur, len(t), ans);
    return ans;
  }
  void getEveryOccurence(int v, int P_length,
                         vi &ans) {
    if (!cloned[v])
      ans.pb(st[v].firstpos - P_length + 1);
    for (int u : st[v].inv_link)
      getEveryOccurence(u, P_length, ans);
  }
};
```

9.11 Trie

Description:

- build with the size of the alphabet (*sigma*) and the first char (*norm*)
- $insert(s)$ insert the string in the trie $O(|s| * sigma)$
- $erase(s)$ remove the string from the trie $O(|s|)$
- $find(s)$ return the last node from the string s, 0 if not found $O(|s|)$

```
struct trie {
  vi2d to;
  vi end, pref;
  int sigma;
  char norm;
  trie(int sigma_ = 26, char norm_ = 'a')
      : sigma(sigma_), norm(norm_) {
    to = {vector<int>(sigma)};
    end = {0}, pref = {0};
  }
  int next(int node, char key) {
    return to[node][key - norm];
  }
  void insert(const string &s) {
    int x = 0;
    for (auto c : s) {
      int &nxt = to[x][c - norm];
      if (!nxt) {
        nxt = len(to);
        to.push_back(vi(sigma));
        end.emplace_back(0), pref.emplace_back(0);
      }
      x = nxt, pref[x]++;
    }
    end[x]++, pref[0]++;
  }
  void erase(const string &s) {
    int x = 0;
    for (char c : s) {
      int &nxt = to[x][c - norm];
      x = nxt, pref[x]--;
      if (!pref[x]) nxt = 0;
    }
    end[x]--, pref[0]--;
  }
  int find(const string &s) {
    int x = 0;
    for (auto c : s) {
      x = to[x][c - norm];
      if (!x) return 0;
    }
    return x;
  }
};
```

9.12 Z-function get occurence positions

Time: $O(len(s) + len(p))$

```
vi getOccPos(string &s, string &p) {
  // Z-function
  char delim = '#';
  string t{p + delim + s};
  vi zs(len(t));
  int l = 0, r = 0;
  for (int i = 1; i < len(t); i++) {
    if (i <= r) zs[i] = min(zs[i - l], r - i + 1);
    while (zs[i] + i < len(t) and
           t[zs[i]] == t[i + zs[i]])
      zs[i]++;
    if (r < i + zs[i] - 1)
      l = i, r = i + zs[i] - 1;
  }
  // Iterate over the results of Z-function to get
  // ranges
  vi ans;
  int start = len(p) + 1 + 1 - 1;
  for (int i = start; i < len(zs); i++) {
    if (zs[i] == len(p)) {
      int l = i - start;
      ans.emplace_back(l);
    }
  }
  return ans;
}
```