Contents

Base 2
  Snippets 2
    cfdiv.cpp 2
    chminmax.cpp 2
    debug.cpp 2
    dir.cpp 2
    disable_leaksanitizer.cpp 2
    fileio.cpp 2
    increase_stack_size.cpp 2
    min_priority_queue.cpp 2
    multitest.cpp 2
    ordered_set.cpp 2
    pragma.cpp 2
    rng.cpp 2
    rng_advanced.cpp 2
  solution_notes.cpp 2
  template.cpp 2
  template_beta.cpp 3
  template_contest.cpp 3
  template_minimal.cpp 4
Data Structures 4
  circular_fenwick.cpp 4
  color_update.cpp 4
  compress.cpp 4
  dsu.cpp 4
  fenwick_tree.cpp 4
  inversions.cpp 4
  line_container.cpp 5
  median.cpp 5
  min_window.cpp 5
  mo.cpp 5
  range_fenwick.cpp 5
  Segment Tree 5
    monoid.cpp 5
    monoid_lazy_context.cpp 6
    monoid_segment_tree.cpp 6
    monoid_segment_tree_lazy.cpp 6
    segment_tree.cpp 7
    segment_tree_lazy_ap.cpp 7
    segment_tree_simple.cpp 7
    segment_tree_sum.cpp 8
  sparse_table.cpp 8
  treap.cpp 8
  treap_mini.cpp 9
Dp 9
  cht_deque.cpp 9
  lis.cpp 9
Geometry 10
  base.cpp 10
Graphs 12
  bfs.cpp 12
  block_cut_tree.cpp 12
  dijkstra.cpp 13
  dinitz.cpp 13
  dinitz_mincost.cpp 14
  eulerian_cycle.cpp 14
  hopcroft_karp.cpp 14
  kosaraju.cpp 15
  lca.cpp 15
  tarjan.cpp 15
  topological_sort.cpp 16
  tour.cpp 16
  two_sat.cpp 16
Math 16
  baby_steps.cpp 16
  chinese_remainder_theorem.cpp 17
  combinatorics.cpp 17
  ext_gcd.cpp 17
  factor.cpp 17
  fexp.cpp 17
  fft.cpp 17
  floor_sum.cpp 18
  lagrange.cpp 18
  matrix_exp.cpp 18
  modular.cpp 18
  modular_mini.cpp 19
  next_array.cpp 19
  non_prime_multinv.cpp 19
  ntt.cpp 19
  primitive_root.cpp 20
  psum.cpp 20
  psum_2d.cpp 20
  sieve.cpp 20
  xor_basis.cpp 20
  xor_hash.cpp 20
String 20
  aho_corasick.cpp 20
  hashed_string.cpp 21
  prefix_automaton.cpp 21
  prefix_function.cpp 21
  prefix_periods.cpp 21
  suffix_array.cpp 22
  trie.cpp 22
  z_function.cpp 22

Base

Snippets

cfdiv.cpp

```cpp
int cdiv(int a, int b) { return a/b+((a^b)>0&&a%b); }
int fdiv(int a, int b) { return a/b-((a^b)<0&&a%b); }
```

chminmax.cpp

```cpp
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); }
```

debug.cpp

```cpp
// Debug {{{
#define var(x) "[", #x, " ", x, "] "
template<typename ...A> void db(A const&... a) { ((cout << (a)), ...); cout << endl; }
//}}}
```

dir.cpp

```cpp
vector<pair<int, int>> D4{{-1, 0},{ 0,-1},{ 0,+1},{+1, 0}};
vector<pair<int, int>> D8{{-1,-1},{-1, 0},{-1,+1},{ 0,-1},{ 0,+1},{+1,-1},{+1, 0},{+1,+1}};
```

disable_leaksanitizer.cpp

```cpp
// Disable LeakSanitizer when working with pointers
extern "C" const char* __asan_default_options() { return "detect_leaks=0"; }
```

fileio.cpp

```cpp
  freopen("file.in", "r", stdin);
  freopen("file.out", "w", stdout);
```

increase_stack_size.cpp

```cpp
void _main() {
  // Real main here
}
static void run_with_stack_size(void (*func)(void), size_t stsize) {
  char *stack, *send;
  stack = (char *)malloc(stsize);
  send = stack + stsize - 16;
  send = (char *)((uintptr_t)send / 16 * 16);
  asm volatile(
      "mov %%rsp, (%0)\n"
      "mov %0, %%rsp\n"
      :
      : "r"(send));
  func();
  asm volatile("mov (%0), %%rsp\n" : : "r"(send));
  free(stack);
}
signed main() {
  run_with_stack_size(_main, 1024 * 1024 * 1024); // 1 GiB stack
  return 0;
}
```

min_priority_queue.cpp

```cpp
template<typename T>
using min_priority_queue = priority_queue<T, vector<T>, greater<T>>;
```

multitest.cpp

```cpp
  int T;
  cin >> T;
  while (T--) solve();
```

ordered_set.cpp

```cpp
#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>;
```

pragma.cpp

```cpp
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

rng.cpp

```cpp
// Random Number Generation {{{
mt19937 rng((int)chrono::steady_clock::now().time_since_epoch().count());
int randint(int l, int r) { return rng() % (r-l+1) + l; }
//}}}
```

rng_advanced.cpp

```cpp
struct Random {

};
```

solution_notes.cpp

```cpp
/* Solution Notes {{{


}}} */
```

template.cpp

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

// Template (v1.5.2 - 2023-10-09) (codeforces:cebolinha, atcoder:edu) {{{
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<class T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,tree_order_statistics_node_update>;

#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

#define int long long
#define fastio ios::sync_with_stdio(false); cin.tie(nullptr)

template<class T> using min_priority_queue = priority_queue<T, vector<T>, greater<T>>;
using ii = pair<int, int>;
using iii = array<int, 3>;

#define V vector
#define all(c) begin(c), end(c)
#define rall(c) rbegin(c), rend(c)
#define sz(c) ((int)size(c))
#define pb push_back
#define eb emplace_back
#define ff first
#define ss second
#define nemo ><>
#define loop(ii, n) for (int ii = 0; ii < (n); ii++)
#define iloop(ii, l, r) for (int ii = (l); ii <= (r); ii++)
#define cond(c, t, f) ((c) ? (t) : (f))
#define mem(a, b) memset((a), (b), sizeof(a))
#define inbounds(x, l, r) ((l) <= (x) && (x) <= (r))
#define L1(res...) [&](auto const& x){ return res; }
#define L2(res...) [&](auto const& x, auto const& y){ return res; }

template<class T, class U> inline void chmin(T& a, U b){ if (a > b) a = b; }
template<class T, class U> inline void chmax(T& a, U b){ if (a < b) a = b; }

template<class T, class U> auto &operator>>(istream &is, pair<T, U> &p) { return is >> p.ff >> p.ss; }
template<class T, class U> auto &operator<<(ostream &os, pair<T, U> const& p) { return os << '{' << p.first << ' ' << p.second << '}'; }

const auto ES = "", SEP = " ";
template<class T> auto &operator>>(istream& is, vector<T> &c) { for (auto &x : c) is >> x; return is; }
template<class T> auto &operator<<(ostream& os, vector<T> const &c) { auto sep = ES; for (auto x : c) os << sep << x, sep = SEP; return os; }
template<class T> auto &operator<<(ostream& os, set<T> const &c) { auto sep = ES; for (auto x : c) os << sep << x, sep =
```

```
SEP; return os; }
template<class T> auto &operator<<(ostream& os, multiset<T>
const &c) { auto sep = ES; for (auto x : c) os << sep << x,
sep = SEP; return os; }
template<class T> auto &operator<<(ostream& os,
unordered_set<T> const &c) { auto sep = ES; for (auto x : c)
os << sep << x, sep = SEP; return os; }
template<class T> auto &operator<<(ostream& os, ordered_set<T>
const &c) { auto sep = ES; for (auto x : c) os << sep << x,
sep = SEP; return os; }
template<class T> auto &operator<<(ostream& os, deque<T> const
&c) { auto sep = ES; for (auto x : c) os << sep << x, sep =
SEP; return os; }
template<class K, class V> auto &operator<<(ostream& os,
map<K,V> const &c) { auto sep = ES; for (auto x : c) os << sep
<< x, sep = SEP; return os; }
template<class K, class V> auto &operator<<(ostream& os,
unordered_map<K,V> const &c) { auto sep = ES; for (auto x : c)
os << sep << x, sep = SEP; return os; }

template<class... A> void in(A &...a) { ((cin >> a), ...); }
template<class... A> void out(A const&... a) { auto sep = ES;
((cout << sep << a, sep = SEP), ...); cout << '\n'; }
template<class... A> void print(A const&... a) { ((cout <<
a), ...); }
#define var(x) "[", #x, " ", x, "] "
template<class... A> void db(A const&... a) { ((cout <<
(a)), ...); cout << endl; }
//}}}

auto main() -> signed {
  fastio;

}
```

template_beta.cpp

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

// Template (v2.0.0 beta - 2024-01-01) (codeforces:cebolinha,
atcoder:edu) {{{
#define tcT template<class T
#define tcTU tcT, class U
#define tcA template<class...A

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

#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

#define int long long

#define V vector
tcT> using min_priority_queue = priority_queue<T, vector<T>,
greater<T>>;
using str = string;
using ii = pair<int, int>;
using iii = array<int, 3>;

#define all(C) begin(C), end(C)
#define rall(C) rbegin(C), rend(C)
#define allb(C) begin(C), end(C), begin(C)
#define sz(C) ((int)size(C))

#define tp top()
#define ft front()
#define bk back()
#define pb push_back
#define ins insert
#define ff first
#define ss second
#define nemo ><>

#define iloop(I,L,R) for (int I = (L); I <= (R); I++)
#define loop(I,N) iloop(I,0,(N)-1)
#define rep(N) loop(_, N)
#define ipool(I,L,R) for (int I = (R); I >= (L); I--)
#define pool(I,N) ipool(I,0,(N)-1)
#define tloop int __T; cin >> __T; while (__T--)
#define each(X,C) for(auto &X : (C))
#define eachc(X,C) for(auto const& X : (C))
#define apply(C,L) each(x,C) L;

#define cond(C, T, F) ((C) ? (T) : (F))
#define mem(C, X) memset((C), (X), sizeof(C))
#define ibi(X, L, R) ((L) <= (X) && (X) <= (R))
#define ib(X, N) ((0) <= (X) && (X) <= (N-1))
#define L1(X...) [&](auto const& x){ return X; }
#define L2(X...) [&](auto const& x, auto const& y){ return
X; }

tcTU> inline bool chmax(T &a, U const& b) { return (a < b ? a
= b, 1 : 0); }
tcTU> inline bool chmin(T &a, U const& b) { return (a > b ? a
= b, 1 : 0); }

int cdiv(int a, int b) { return a/b+((a^b)>0&&a%b); }
int fdiv(int a, int b) { return a/b-((a^b)<0&&a%b); }

#define fastio ios::sync_with_stdio(false); cin.tie(nullptr)

tcTU> auto &operator>>(istream &is, pair<T, U> &p) { return is
>> p.ff >> p.ss; }
tcTU> auto &operator<<(ostream &os, pair<T, U> const& p)
{ return os << '{' << p.first << ' ' << p.second << '}'; }
tcT> auto &operator>>(istream& is, vector<T> &C) { for (auto
&x : C) is >> x; return is; }
```



```
tcT> struct is_container : std::false_type {};
#define __ADDC(C) tcA> struct is_container<C<A...>> :
std::true_type {};
__ADDC(vector); __ADDC(set); __ADDC(multiset);
__ADDC(unordered_set); __ADDC(map); __ADDC(unordered_map);

const auto __ES = "", __SEP = " ";
tcT> enable_if<is_container<T>::value, ostream>::type
&operator<<(ostream& os, T const &C) { auto sep = __ES;
each(x, C) os << sep << x, sep = __SEP; return os; }
tcT> auto &operator<<(ostream& os, ordered_set<T> const &C)
{ auto sep = __ES; each(x, C) os << sep << x, sep = __SEP;
return os; }

tcA> void in(A &...a) { ((cin >> a), ...); }
tcA> void out(A const&... a) { auto sep = __ES; ((cout << sep
<< a, sep = __SEP), ...); cout << '\n'; }
tcA> void print(A const&... a) { ((cout << a), ...); }
#define var(x) "[", #x, " ", x, "] "
#ifdef LOCAL
tcA> void db(A const&... a) { ((cout << (a)), ...); cout <<
endl; }
#else
tcA> void db(A const&... a) {}
#endif
//}}}

auto main() -> signed {
  fastio;

}
```

template_contest.cpp

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

#define int long long
#define endl '\n'
#define V vector

#define all(x) begin(x),end(x)
#define sz(x) (int)size(x)
#define loop(i,n) for (int i = 0; i < (n); i++)

signed main() {
  ios::sync_with_stdio(false); cin.tie(nullptr);

}
```

template_minimal.cpp

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

#define int long long

signed main() {
  ios::sync_with_stdio(false); cin.tie(nullptr);

}
```

Data Structures

---

circular_fenwick.cpp

```cpp
// {{{ Circular Fenwick
struct CircularFenwick {
  int N;
  RangeFenwick RF;

  CircularFenwick(int N) : N(N), RF(N) {}

  void update(int l, int r, int x) {
    if (l <= r) RF.update(l, r, x);
    else {
      RF.update(l, N-1, x);
      RF.update(0, r, x);
    }
  }

  int get(int p) {
    return RF.query(p);
  }
};
//}}}
```

color_update.cpp

```cpp
// Color Update {{{
template<class T = long long>
struct ColorUpdate {
  struct Range {
    int l, r;
    T x;

    Range(int l) : l(l) {}
    Range(int l, int r, T x) : l(l), r(r), x(x) {}
    bool operator<(const Range& o) const { return l < o.l; }
  };

  set<Range> ranges;

  vector<Range> update(int l, int r, T x) {
    if (l > r) return {};

    auto it = ranges.lower_bound(l);
    if (it != begin(ranges)) {
      it--;
      if (it->r >= l) {
        auto cur = *it;
        ranges.erase(it);
        ranges.insert(Range(cur.l, l-1, cur.x));
        ranges.insert(Range(l, cur.r, cur.x));
      }
    }
    it = ranges.lower_bound(r+1);
    if (it != begin(ranges)) {
      it--;
      if (it->r > r) {
        auto cur = *it;
        ranges.erase(it);
        ranges.insert(Range(cur.l, r, cur.x));
        ranges.insert(Range(r+1, cur.r, cur.x));
      }
    }

    vector<Range> ans;
    for (auto it = ranges.lower_bound(l); it != end(ranges) &&
it->l <= r; it++) {
      ans.push_back(*it);
    }
    ranges.erase(ranges.lower_bound(l),
ranges.lower_bound(r+1));
    ranges.insert(Range(l, r, x));
    return ans;
  }
};
//}}}
```

compress.cpp

```cpp
// Compress Values {{{
map<int, int> compress(vector<int> A) {
  sort(begin(A), end(A));
  map<int, int> M;
  for (auto x : A) if (!M.count(x)) M[x] = size(M);
  return M;
}
//}}}
```

dsu.cpp

```cpp
//{{{ DSU
struct DSU {
  vector<int> P, S;
  explicit DSU (int N) : P(N, -1), S(N, 1) {};
  int leader(int a) {
    if (P[a] == -1) return a;
    return P[a] = leader(P[a]);
  }
  int merge(int a, int b) {
    a = leader(a);
    b = leader(b);
    if (a == b) return a;
    if (S[a] < S[b]) swap(a, b);
    P[b] = a;
    S[a] += S[b];
    return a;
  }
  int same(int a, int b) {
    return leader(a) == leader(b);
  }
};
//}}}
```

fenwick_tree.cpp

```cpp
//{{{ FenwickTree
struct FenwickTree {
  int N;
  vector<int> data;

  FenwickTree(int N) : N(N), data(N) {}

  void add(int idx, int delta) {
    for (; idx < N; idx |= idx+1)
      data[idx] += delta;
  }

  int sum(int r) {
    int ret = 0;
    for (; r >= 0; r &= r+1, r--)
      ret += data[r];
    return ret;
  }

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

inversions.cpp

```cpp
// Inversion Counting {{{
int inversions(vector<int> const& A) {
  ordered_set<pair<int, int>> OS;
  int ans = 0;
  for (int i = 0; i < size(A); i++) {
    ans += OS.size() - OS.order_of_key({A[i], i});
    OS.insert({A[i], i});
  }
  return ans;
```

```
}
//}}}
```

line_container.cpp

```
// LineContainer {{{
// Line = k*x + m, has maximum value for x up to P
struct Line {
  mutable int k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(int x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  static const int INF = LLONG_MAX;
  int fdiv(int a, int b) { return a/b-((a^b)<0&&a%b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = INF, 0;
    if (x->k == y->k) x->p = x->m > y->m ? INF : -INF;
    else x-> p = fdiv(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(int k, int m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  int query(int x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};

struct MinLineContainer {
  LineContainer LC;
  void add(int k, int m) { LC.add(-k, -m); };
  int query(int x) { return -LC.query(x); }
};
//}}}
```

median.cpp

```
struct Median {
  int ss = 0, sb = 0;
  multiset<int> S, B;

  void rebalance() {
    while (sz(S) < sz(B)) S.insert(*begin(B)), ss +=
*begin(B), sb -= *begin(B), B.erase(begin(B));
    while (sz(S) > sz(B)+1) B.insert(*rbegin(S)), sb +=
*rbegin(S), ss -= *rbegin(S), S.erase(prev(S.end()));
    while (!empty(S) && !empty(B) && *rbegin(S) > *begin(B)) {
      int a = *rbegin(S), b = *begin(B);
      ss -= a, ss += b;
      sb -= b, sb += a;
      S.erase(prev(end(S)));
      B.erase(begin(B));
      S.insert(b);
      B.insert(a);
    }
  }

  void insert(int x) {
    S.insert(x);
    ss += x;
    rebalance();
  }

  void remove(int x) {
    if (x <= *rbegin(S)) S.erase(S.find(x)), ss -= x;
    else B.erase(B.find(x)), sb -= x;
    rebalance();
  }

  int get() {
    return *rbegin(S);
  }

  int cost() {
    int med = get();
    int cost = med * sz(S) - ss;
    cost += sb - med * sz(B);
    return cost;
  }
};
```

min_window.cpp

```
// Minimum Window {{{
// Be careful with case W = 0
struct MinWindow {
  int W;
  deque<pair<int, int>> Q;

public:
  explicit MinWindow(int W) : W(W) {}

  void push(int idx, int x) {
    while (!Q.empty() && Q.front().ff < idx-W+1)
Q.pop_front();
    while (!Q.empty() && Q.back().ss >= x) Q.pop_back();
    Q.pb({idx, x});
  }

  int get() const { return Q.front().ss; }
};
//}}}
```

mo.cpp

```
// Mo's Algorithm {{{
const int BLK = 400;
struct MoQuery {
  int l, r, idx;
  bool operator<(MoQuery const& o) const {
    if (l/BLK != o.l/BLK) return l < o.l;
    if (l/BLK % 2 == 0) return r < o.r;
    else return r > o.r;
  }
};
//}}}
```

range_fenwick.cpp

```
// Range Fenwick {{{
struct RangeFenwick {
  FenwickTree FT;
  RangeFenwick(int N) : FT(N+1) {}

  void update(int l, int r, int x) {
    FT.add(l, x);
    FT.add(r+1, -x);
  }

  int query(int idx) {
    return FT.sum(idx);
  }
};
//}}}
```

Segment Tree

monoid.cpp

```
// Monoid {{{
template<typename T, typename F>
struct Monoid {
  const T identity;
  const F op;
  constexpr Monoid(T identity, F op) : identity(identity),
op(op) {}
  constexpr T operator()(T const& a, T const& b) const
{ return op(a, b); }
};

namespace Monoids {
  template<typename T> constexpr Monoid Sum(T(),
std::plus<T>{});
  template<typename T> constexpr Monoid SumPair(std::pair<T,
T>{T(), T()}, [](auto const& a, auto const& b) {
    return std::pair<T, T>{a.first+b.first,
a.second+b.second};
  });
```

```
  template<typename T> constexpr Monoid Product(T(1),
std::multiplies<T>{});
  template<typename T, T INF> constexpr Monoid Max(T(-INF), []
(T const& a, T const& b) { return max(a, b); });
  template<typename T, T INF> constexpr Monoid Min(T(+INF), []
(T const& a, T const& b) { return min(a, b); });
  template<typename T, T INF> constexpr Monoid
MaxIdx(std::pair<T, T>{-INF, T(-1)}, [](auto const& a, auto
const& b) {
    if (a.first >= b.first) return a;
    return b;
  });
  template<typename T, T INF> constexpr Monoid
MinIdx(std::pair<T, T>{+INF, T(-1)}, [](auto const& a, auto
const& b) {
    if (a.first <= b.first) return a;
    return b;
  });
  template<typename T> constexpr Monoid
LinearComposition(std::pair<T, T>{T(1), T(0)}, [](auto const&
a, auto const& b) {
    // This applies b on a, swap arguments if needed
    return std::pair<T, T>{a.first*b.first, a.second*b.first +
b.second};
  });
};
// }}}
```

monoid_lazy_context.cpp

```
// Lazy Context {{{
template<typename AF, typename T, typename L>
concept LazyApplyFunction = requires(AF apply, T t, L l, int
il, int ir) {
  { std::invoke(apply, t, l, il, ir) } -> std::same_as<T>;
};

template<typename T, typename TF, typename L, typename LF,
typename AF>
requires LazyApplyFunction<AF, T, L>
struct LazyContext {
  const Monoid<T, TF> data_M;
  const Monoid<L, LF> lazy_M;
  const AF apply;
  constexpr LazyContext(Monoid<T, TF> const& data_M, Monoid<L,
LF> const& lazy_M, AF const& apply)
    : data_M(data_M), lazy_M(lazy_M), apply(apply) {}
};

namespace LazyContexts {
  template<typename T> constexpr LazyContext
RangeAddQuerySum(Monoids::Sum<T>, Monoids::Sum<T>, [](T d, T
l, int il, int ir) { return d+l*(ir-il+1); });
  template<typename T, T INF> constexpr LazyContext
RangeAddQueryMax(Monoids::Max<T, INF>, Monoids::Sum<T>, [](T
d, T l, int il, int ir) { return d+l; });
  template<typename T, T INF> constexpr LazyContext
RangeAddQueryMaxIdx(Monoids::MaxIdx<T, INF>, Monoids::Sum<T>,
[](pair<T, T> d, T l, int il, int ir) { return pair{d.first+l,
d.second}; });
  template<typename T, T INF> constexpr LazyContext
RangeAddQueryMin(Monoids::Min<T, INF>, Monoids::Sum<T>, [](T
d, T l, int il, int ir) { return d+l; });
  template<typename T, T INF> constexpr LazyContext
RangeAddQueryMinIdx(Monoids::MinIdx<T, INF>, Monoids::Sum<T>,
[](pair<T, T> d, T l, int il, int ir) { return pair{d.first+l,
d.second}; });
};
//}}}
```

monoid_segment_tree.cpp

```
// Segment Tree {{{
template<typename T, typename TF>
class SegmentTree {
  const int N;
  const Monoid<T, TF> M;
  std::vector<T> data;

  constexpr int left(int id) const { return 2*id; }
  constexpr int right(int id) const { return 2*id+1; }

public:
  explicit SegmentTree(int N, Monoid<T, TF> const& M) : N(N),
data(2*N, M.identity), M(M) {}
  explicit SegmentTree(std::vector<T> const& A, Monoid<T, TF>
const& M) : SegmentTree(size(A), M) {
    for (int i = 0; i < N; i++) set(i, A[i]);
  }

  void set(int p, T const& val) {
    for (data[p+=N]=val; p /= 2;)
      data[p] = M(data[left(p)], data[right(p)]);
  }

  T get(int p) const {
    return data[p+N];
  }

  void add(int p, T const& val) {
    set(p, get(p)+val);
  }

  T query(int l, int r) const {
    T rl = M.identity, rr = M.identity;
    for (l += N, r += N+1; l < r; l/=2, r/=2) {
      if (l & 1) rl = M(rl, data[l++]);
      if (r & 1) rr = M(data[--r],rr);
    }
    return M(rl, rr);
  }
};
//}}}
```

monoid_segment_tree_lazy.cpp

```
// Lazy Segment Tree {{{
template<typename T, typename TF, typename L, typename LF,
typename AF>
struct SegmentTreeLazy {
  const int N;
  const LazyContext<T, TF, L, LF, AF> context;

  std::vector<T> data;
  std::vector<L> lazy;

  constexpr int left(int id) const { return 2*id; }
  constexpr int right(int id) const { return 2*id+1; }

  void push(int id, int il, int ir) {
    int len = ir-il+1;
    if (len >= 2) {
      lazy[left(id)] = context.lazy_M(lazy[left(id)],
lazy[id]);
      lazy[right(id)] = context.lazy_M(lazy[right(id)],
lazy[id]);
    }
    data[id] = context.apply(data[id], lazy[id], il, ir);
    lazy[id] = context.lazy_M.identity;
  }

  void set(int p, T const& val, int id, int il, int ir) {
    push(id, il, ir);
    if (il == ir) {
      data[id] = val;
      return;
    }
    int im = std::midpoint(il, ir);
    if (p <= im) set(p, val, left(id), il, im);
    else set(p, val, right(id), im+1, ir);
    data[id] = context.data_M(data[left(id)],
data[right(id)]);
  }

  void update(int l, int r, L x, int id, int il, int ir) {
    push(id, il, ir);
    if (r < il || ir < l) return;
    if (l <= il && ir <= r) {
      lazy[id] = context.lazy_M(lazy[id], x);
      push(id, il, ir);
      return;
    }
    int im = std::midpoint(il, ir);
    update(l, r, x, left(id), il, im);
    update(l, r, x, right(id), im+1, ir);
    data[id] = context.data_M(data[left(id)],
data[right(id)]);
  }

  T query(int l, int r, int id, int il, int ir) {
    push(id, il, ir);
```

```
    if (r < il || ir < l) return context.data_M.identity;
    if (l <= il && ir <= r) return data[id];
    int im = std::midpoint(il, ir);
    return context.data_M(query(l, r, left(id), il, im),
query(l, r, right(id), im+1, ir));
  }

  void build(std::vector<T> const& A, int id, int il, int ir)
{
    if (il == ir) {
      data[id] = A[il];
      return;
    }
    int im = std::midpoint(il, ir);
    build(A, left(id), il, im);
    build(A, right(id), im+1, ir);
    data[id] = context.data_M(data[left(id)],
data[right(id)]);
  }

public:
  explicit SegmentTreeLazy(int N, LazyContext<T, TF, L, LF,
AF> const& LC)
    : N(N), data(4*N, LC.data_M.identity), lazy(4*N,
LC.lazy_M.identity), context(LC) {}

  explicit SegmentTreeLazy(std::vector<T> const& A,
LazyContext<T, TF, L, LF, AF> const& LC)
    : LazySegmentTree(size(A), LC) {
    build(A, 1, 0, N-1);
  }

  void update(int l, int r, L x) { update(l, r, x, 1, 0,
N-1); }

  void set(int p, T const& val) { set(p, val, 1, 0, N-1); }

  [[nodiscard]]
  T query(int l, int r) { return query(l, r, 1, 0, N-1); }

  void debug() {
    std::cout << "Debug: ";
    for (int i = 0; i < N; i++) {
      std::cout << query(i, i) << ' ';
    }
    std::cout << std::endl;
  }
};
//}}}
```

segment_tree.cpp

```
//{{{ Segment Tree
template<typename T>
class SegmentTree {
  const int N;
  vector<T> data;
public:
  explicit SegmentTree(int N) : N(N), data(2*N) {}
  explicit SegmentTree(vector<T> const& A) : N(size(A)),
data(2*size(A)) {
    for (int i = 0; i < N; i++) set(i, A[i]);
  }

  void set(int p, T const& val) {
    for (data[p+=N]=val; p /= 2;)
      data[p] = data[2*p]+data[2*p+1];
  }

  T get(int p) const {
    return data[p + N];
  }

  void add(int p, T const& val) {
    set(p, get(p)+val);
  }

  T sum(int l, int r)  const {
    T rl = T(), rr = T();
    for (l+=N, r+=N+1; l<r; l/=2, r/=2) {
      if (l&1) rl = rl+data[l++];
      if (r&1) rr = data[--r]+rr;
    }
    return rl+rr;
  }
};
//}}}
```

segment_tree_lazy_ap.cpp

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

#define int long long

signed main() {
  ios::sync_with_stdio(false); cin.tie(nullptr);

}

// Lazy Segment Tree (Arithmetic Progression) {{{
using ii = pair<int, int>;
ii operator+(ii a, ii b) { return {a.first + b.first, a.second
+ b.second}; }

struct SegmentTreeLazyAP {
  int N;
  vector<ii> L;
  vector<int> T;

  explicit SegmentTreeLazyAP(int N) : N(N) {
    L.resize(4*N);
    T.resize(4*N);
  }

  int ap_sum(int base, int step, int len) {
    return (base + base+step*(len-1)) * len / 2;
  }

  void push(int id, int il, int ir) {
    auto [base, step] = L[id];
    int len = ir-il+1;

    T[id] += ap_sum(base, step, len);

    if (len > 1) {
      int im = midpoint(il, ir);
      L[2*id] = L[2*id] + ii{base, step};
      L[2*id+1] = L[2*id+1] + ii{base + (im+1-il)*step, step};
    }

    L[id] = {0, 0};
  }

  int update(int l, int r, ii x, int id, int il, int ir) {
    push(id, il, ir);
    if (r < il || ir < l) return T[id];
    if (l <= il && ir <= r) {
      L[id] = L[id] + ii{x.first + (il-l)*x.second, x.second};
      push(id, il, ir);
      return T[id];
    }
    int im = midpoint(il, ir);
    return T[id] = update(l, r, x, 2*id, il, im)
                 + update(l, r, x, 2*id+1, im+1, ir);
  }
  void update(int l, int r, ii x) { update(l, r, x, 1, 0,
N-1); }

  int query(int l, int r, int id, int il, int ir) {
    push(id, il, ir);
    if (r < il || ir < l) return 0;
    if (l <= il && ir <= r) return T[id];
    int im = midpoint(il, ir);
    return query(l, r, 2*id, il, im)
         + query(l, r, 2*id+1, im+1, ir);
  }
  int query(int l, int r) { return query(l, r, 1, 0, N-1); }
};
//}}}
```

segment_tree_simple.cpp

```
//{{{ Segment Tree Simple
template<typename T>
class SegmentTreeSimple {
  int N;
  T neutral;
  vector<T> data;
  function<T(T,T)> merge;
```

```
public:
  SegmentTreeSimple(int N, T neutral, function<T(T,T)> merge)
{
    this->N = N;
    this->neutral = neutral;
    this->merge = merge;
    data.assign(2*N, neutral);
  }

  SegmentTreeSimple(vector<T> const& A, T neutral,
function<T(T,T)> merge) {
    this->N = A.size();
    this->neutral = neutral;
    this->merge = merge;
    data.resize(2*N);
    for (int i = 0; i < N; i++) data[i+N] = A[i];
    for (int i=N-1; i>0; i--)
      data[i]=merge(data[2*i],data[2*i+1]);
  }

  void set(int p, T const& val) {
    for (data[p+=N]=val; p /= 2;)
      data[p] = merge(data[2*p], data[2*p+1]);
  }

  T get(int p) const {
    return data[p + N];
  }

  void add(int p, T const& val) {
    set(p, merge(get(p),val));
  }

  T sum(int l, int r) const {
    T rl = neutral, rr = neutral;
    for (l+=N, r+=N+1; l<r; l/=2, r/=2) {
      if (l&1) rl = merge(rl, data[l++]);
      if (r&1) rr = merge(data[--r], rr);
    }
    return merge(rl, rr);
  }
};
//}}}
```

segment_tree_sum.cpp

```
// Lazy Segment Tree (Sum) {{{
struct SegmentTreeLazy {
  int N;
  vector<int> L, T;

  explicit SegmentTreeLazy(int N) : N(N) {
    L.resize(4*N);
    T.resize(4*N);
  }

  void push(int id, int il, int ir) {
    T[id] += L[id] * (ir-il+1);
    if (il < ir) {
      L[2*id] += L[id];
      L[2*id+1] += L[id];
    }
    L[id] = 0;
  }

  int update(int l, int r, int x, int id, int il, int ir) {
    push(id, il, ir);
    if (r < il || ir < l) return T[id];
    if (l <= il && ir <= r) {
      L[id] += x;
      push(id, il, ir);
      return T[id];
    }
    int im = midpoint(il, ir);
    return T[id] = update(l, r, x, 2*id, il, im)
                 + update(l, r, x, 2*id+1, im+1, ir);
  }
  void update(int l, int r, int x) { update(l, r, x, 1, 0,
N-1); }

  int query(int l, int r, int id, int il, int ir) {
    push(id, il, ir);
    if (r < il || ir < l) return 0;
    if (l <= il && ir <= r) return T[id];

    int im = midpoint(il, ir);
    return query(l, r, 2*id, il, im)
         + query(l, r, 2*id+1, im+1, ir);
  }
  int query(int l, int r) { return query(l, r, 1, 0, N-1); }
};
//}}}
```

sparse_table.cpp

```
// Sparse Table (Min) {{{
struct SparseTable {
  int N;
  vector<vector<int>> ST;
  vector<int> LOG;
  SparseTable(vector<int> const& A) : N(size(A)),
LOG(size(A)+1) {
    LOG[1] = 0;
    for (int i = 2; i < size(LOG); i++) {
      LOG[i] = LOG[i/2] + 1;
    }

    ST.assign(LOG[N]+1, vector<int>(N));
    ST[0] = A;

    for (int l = 1; l < size(ST); l++) {
      int len = (1 << l);
      for (int i = 0; i+len-1 < N; i++) {
        ST[l][i] = min(ST[l-1][i], ST[l-1][i+len/2]);
      }
    }
  }

  int query(int l, int r) {
    int lg = LOG[r-l+1];
    return min(ST[lg][l], ST[lg][r-(1<<lg)+1]);
  }
};
//}}}
```

treap.cpp

```
// Treap {{{
struct Node {
  int X, P, S;
  Node *L, *R;
  Node (int x) : X(x), P(rng()), S(1), L(nullptr), R(nullptr)
{}
};

int size(Node* t) { return t ? t->S : 0; }

pair<Node*, Node*> split(Node *t, int cnt) {
  if (!t) return {nullptr, nullptr};

  if (size(t->L) < cnt) {
    auto [l, r] = split(t->R, cnt-size(t->L)-1);
    t->R = l;
    t->S = 1 + size(t->L) + size(t->R);
    return {t, r};
  } else {
    auto [l, r] = split(t->L, cnt);
    t->L = r;
    t->S = 1 + size(t->L) + size(t->R);
    return {l, t};
  }
}

tuple<Node*, Node*, Node*> split(Node* t, int l, int r) {
  auto [L, x] = split(t, l);
  auto [M, R] = split(x, r-l+1);
  return {L, M, R};
}

Node* meld(Node *l, Node *r) {
  if (!l) return r;
  if (!r) return l;

  if (l->P > r->P) {
    l->R = meld(l->R, r);
    l->S = 1 + size(l->L) + size(l->R);
    return l;
  } else {
    r->L = meld(l, r->L);
```

```
    r->S = 1 + size(r->L) + size(r->R);
    return r;
  }
}

Node* meld(Node *l, Node *m, Node *r) {
  return meld(meld(l, m), r);
}

ostream &operator<<(ostream &os, Node* x) {
  if (!x) return os;
  os << x->L;
  os << x->X;
  os << x->R;
  return os;
}
//}}}
```

treap_mini.cpp

```
// Minimum Treap {{{
template<typename T>
struct Node {
  T X;
  int P, S;
  int mini;
  Node *L, *R;
  Node (T x) : X(x), mini(x), P(rng()), S(1), L(nullptr),
R(nullptr) {}

  void update() {
    S = 1 + gsize(L) + gsize(R);
    mini = min({X, gmin(L), gmin(R)});
  }
};

template<typename T>
int gsize(Node<T>* t) { return t ? t->S : 0; }

template<typename T>
int gmin(Node<T>* t) { return t ? t->mini : 1e9; }

template<typename T>
pair<Node<T>*, Node<T>*> split(Node<T> *t, int cnt) {
  if (!t) return {nullptr, nullptr};

  if (gsize(t->L) < cnt) {
    auto [l, r] = split(t->R, cnt-gsize(t->L)-1);
    t->R = l;
    t->update();
    return {t, r};
  } else {
    auto [l, r] = split(t->L, cnt);
    t->L = r;
    t->update();
    return {l, t};
  }
}

template<typename T>
tuple<Node<T>*, Node<T>*, Node<T>*> split(Node<T>* t, int l,
int r) {
  auto [L, x] = split(t, l);
  auto [M, R] = split(x, r-l+1);
  return {L, M, R};
}

template<typename T>
Node<T>* merge(Node<T> *l, Node<T> *r) {
  if (!l) return r;
  if (!r) return l;

  if (l->P > r->P) {
    l->R = merge(l->R, r);
    l->update();
    return l;
  } else {
    r->L = merge(l, r->L);
    r->update();
    return r;
  }
}

template<typename T>
Node<T>* merge(Node<T> *l, Node<T> *m, Node<T> *r) {
  return merge(merge(l, m), r);
}

template<typename T>
ostream &operator<<(ostream &os, Node<T>* x) {
  if (!x) return os;
  os << x->L;
  os << x->X;
  os << x->R;
  return os;
}
//}}}
```

Dp

---

cht_deque.cpp

```
// TODO: Fix this, for now it's just for reference
// CHT Deque {{{
#include <bits/stdc++.h>
using namespace std;

#define int long long

int cdiv(int a, int b) { return a/b+((a^b)>0&&a%b); }

using Line = pair<int, int>;
int inter(Line a, Line b) { return cdiv(b.second-a.second,
a.first-b.first); }
int eval(Line a, int x) { return x * a.first + a.second; }

signed main() {
  ios::sync_with_stdio(false); cin.tie(nullptr);

  int N, X;
  cin >> N >> X;

  vector<int> S(N), F(N);
  for (auto &x : S) cin >> x;
  for (auto &x : F) cin >> x;

  deque<Line> D;
  D.push_back({X, 0});

  for (int i = 0; i < N; i++) {
    while (size(D) >= 2 && inter(D[0], D[1]) <= S[i])
D.pop_front();
    int val = eval(D[0], S[i]);

    Line l{F[i], val};
    if (end(D)[-1].first == l.first) continue;

    while (size(D) >= 2 && inter(end(D)[-2], end(D)[-1]) >=
inter(end(D)[-1], l)) D.pop_back();
    D.push_back(l);
  }

  cout << eval(D.front(), S.back()) << endl;
}
//}}}
```

lis.cpp

```
// Longest Increasing Subsequence {{{
int lis(vector<int> const& A) {
  vector<int> most(size(A)+1, numeric_limits<int>::max());

  int ans = 0;
  for (auto x : A) {
    auto lb = lower_bound(begin(most), end(most), x);
    *lb = x;
    ans = max(ans, (int)(lb - begin(most) + 1));
  }
  return ans;
}
//}}}
```

Geometry

base.cpp

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

#define int long long

signed main() {
  ios::sync_with_stdio(false); cin.tie(nullptr);

}

//{{{ Geometry Base
const long double EPS = 1e-9;

template<typename T>
T sq(T x) { return x*x; }

template<typename T>
bool eq(T const& a, T const& b) {
  return abs(a-b) <= EPS;
}

template<>
bool eq<int>(int const& a, int const& b) {
  return a == b;
}

template<typename T>
struct Point {
  T x, y;
  Point () : x(0), y(0) {}
  Point (T x, T y) : x(x), y(y) {}

  Point operator+(Point const& o) const { return { x+o.x,
y+o.y }; }
  Point operator-(Point const& o) const { return { x-o.x, y-
o.y }; }
  Point operator*(T const& t) const { return { x*t, y*t }; }
  Point operator/(T const& t) const { return { x/t, y/t }; }
  T operator*(Point const& o) const { return x*o.x + y*o.y; }
  T operator^(Point const& o) const { return x*o.y - y*o.x; }

  bool operator<(Point const& o) const {
    if (!eq(x, o.x)) return x < o.x;
    if (!eq(y, o.y)) return y < o.y;
    return 0;
  }

  bool operator==(Point const& o) const {
    return eq(x, o.x) && eq(y, o.y);
  }

  bool operator!=(Point const& o) const {
    return !(*this == o);
  }

  friend ostream& operator<<(ostream& os, Point const& p) {
    return os << p.x << ' ' << p.y;
  }

  friend istream& operator>>(istream& is, Point &p) {
    return is >> p.x >> p.y;
  }
};

template<typename T>
long double norm(Point<T> const& a) {
  return sqrtl(a * a);
}

template<typename T>
long double norm(Point<T> const& a, Point<T> const& b) {
  return norm(a-b);
}

template<typename T>
T norm2(Point<T> const& a) {
  return a * a;
}

template<typename T>
T norm2(Point<T> const& a, Point<T> const& b) {
  return norm2(a-b);
}

template<typename T>
Point<T> unit(Point<T> const& a) {
  return a / norm(a);
}

template<typename T>
T proj_len(Point<T> const& p, Point<T> const& a, Point<T>
const& b) {
  T len = (p-a) * (b-a) / norm(b-a);
  return len;
}

template<typename T>
Point<T> proj(Point<T> const& p, Point<T> const& a, Point<T>
const& b) {
  return a + unit(b-a) * proj_len(p, a, b);
}

template<typename T>
Point<T> reflection(Point<T> const& p, Point<T> const& a,
Point<T> const& b) {
  return p + 2*(proj(p, a, b)-p);
}

template<typename T>
T sarea(Point<T> const& a, Point<T> const& b, Point<T> const&
c) {
  return ((b-a)^(c-b))/2;
}

// left = +1
// collinear = 0
// right = -1
template<typename T>
int side(Point<T> const& p, Point<T> const& a, Point<T> const&
b) {
  T x = (b-a) ^ (p-a);
  return (x > EPS) - (x < -EPS);
}

template<typename T>
Point<T> rot(Point<T> const& p, long double a) {
  return Point{p.x * cos(a) - p.y * sin(a),
               p.y * cos(a) + p.x * sin(a)};
}

template<typename T>
Point<T> rot90cw(Point<T> const& a) {
  return Point{a.y, -a.x};
}

template<typename T>
Point<T> rot90ccw(Point<T> const& a) {
  return Point{-a.y, a.x};
}

template<typename T>
Point<T> transp(Point<T> const& a) {
  return Point{a.y, a.x};
}

// Everything is untested
template<typename T>
struct Line  {
  Point<T> p1, p2;
  T a, b, c;

  Line () {}

  Line (Point<T> const& p1, Point<T> const& p2) : p1(p1),
p2(p2),
    a(p1.y - p2.y),
    b(p2.x - p1.x),
    c(p1^p2) {}

  Line (T a, T b, T c) : a(a), b(b), c(c) {
    if (b == 0) {
      p1 = Point<T>(1, -c/a);
      p2 = Point<T>(0, -c/a);
```

```
    } else {
      p1 = Point<T>(1, (-c-a*1)/b);
      p2 = Point<T>(0, -c/b);
    }
  }

  bool operator<(Line const& o) const {
    if (p1 != o.p1) return p1 < o.p1;
    if (p2 != o.p2) return p2 < o.p2;
    return 0;
  }

  T eval(Point<T> const& p) const {
    return a * p.x + b * p.y + c;
  }

  T eval(T const& x) const {
    return (-c-a*x)/b;
  }

  bool inside(Point<T> const& p) const {
    return eq(eval(p), T());
  }

  bool inside_seg(Point<T> const& p) const {
    return eq(((p1-p) ^ (p2-p)), T())
           && ((p1-p) * (p2-p)) <= EPS;
  }
};

// Unlikely to work with integral T
// WARN: Doesn't work with a line that is a single point
template<typename T>
vector<Point<T>> inter_line(Line<T> const& l1, Line<T> const&
l2) {
  auto det = l1.a*l2.b - l1.b*l2.a;
  if (eq(det, T())) return {};
  auto x = (l1.b*l2.c - l1.c*l2.b)/det;
  auto y = (l1.c*l2.a - l1.a*l2.c)/det;
  return {{x,y}};
}

// Segments with point overlap
template<typename T>
vector<Point<T>> inter_seg_proper(Line<T> const& l1, Line<T>
const& l2) {
  auto ans = inter_line(l1, l2);
  if (ans.empty()) return {};
  if (!l1.inside_seg(ans.front())) return {};
  if (!l2.inside_seg(ans.front())) return {};
  return ans;
}

template<typename T>
bool seg_has_inter(Line<T> const& l1, Line<T> const& l2) {
  if (side(l2.p1, l1.p1, l1.p2) * side(l2.p2, l1.p1, l1.p2) <
0
   && side(l1.p1, l2.p1, l2.p2) * side(l1.p2, l2.p1, l2.p2) <
0) return true;
  if (l1.inside_seg(l2.p1)) return true;
  if (l1.inside_seg(l2.p2)) return true;
  if (l2.inside_seg(l1.p1)) return true;
  if (l2.inside_seg(l1.p2)) return true;
  return false;
}

template<typename T>
vector<Point<T>> inter_seg(Line<T> const& l1, Line<T> const&
l2) {
  if (!seg_has_inter(l1, l2)) return {};

  vector<Point<T>> ps;
  if (l1.inside_seg(l2.p1)) ps.push_back(l2.p1);
  if (l1.inside_seg(l2.p2)) ps.push_back(l2.p2);
  if (l2.inside_seg(l1.p1)) ps.push_back(l1.p1);
  if (l2.inside_seg(l1.p2)) ps.push_back(l1.p2);
  sort(begin(ps), end(ps));
  ps.erase(unique(begin(ps), end(ps)), end(ps));

  if (size(ps) == 1) return {ps.front()};
  else if (size(ps) > 1) return {ps.front(), ps.back()};

  return {inter_seg_proper(l1, l2).front()};
}

template<typename T>
T point_line_dist(Point<T> const& p, Line<T> const& l) {
  if (l.p1 == l.p2) return norm(l.p1-p);
  return 2 * abs(sarea(p, l.p1, l.p2)) / norm(l.p1-l.p2);
}

template<typename T>
T point_seg_dist(Point<T> const& p, Line<T> const& l) {
  if (l.p1 == l.p2) return norm(l.p1-p);
  if ((l.p2-l.p1)*(p-l.p1) < 0) return norm(l.p1-p);
  if ((l.p1-l.p2)*(p-l.p2) < 0) return norm(l.p2-p);
  return point_line_dist(p, l);
}

template<typename T>
T seg_dist(Line<T> const& l1, Line<T> const& l2) {
  if (seg_has_inter(l1, l2)) return T();
  return min({point_seg_dist(l1.p1, l2),
              point_seg_dist(l1.p2, l2),
              point_seg_dist(l2.p1, l1),
              point_seg_dist(l2.p2, l1)});
}

template<typename T>
struct Circle {
  Point<T> c;
  T r;
  Circle (Point<T> const& c,  T r) : c(c), r(r) {}
  bool inside(Point<T> const& a) const {
    return norm(a-c) <= r + EPS;
  }
};

template<typename T>
vector<Point<T>> inter_circle(Circle<T> const& c1, Circle<T>
const& c2) {
  if (c1.c == c2.c) return {};
  Point vec = c2.c - c1.c;
  T d2 = vec * vec;
  T sum = c1.r + c2.r;
  T dif = c1.r - c2.r;
  T p = (d2 + c1.r * c1.r - c2.r * c2.r) / (2 * d2);
  T h2 = c1.r * c1.r - p * p * d2;
  if (sum * sum < d2 || dif * dif > d2) return {};
  Point mid = c1.c + vec*p;
  Point per = Point(-vec.y, vec.x) * sqrt(max(T(), h2) / d2);
  if (per == Point<T>()) return {mid};
  return {mid + per, mid - per};
}

// TODO: Convert inside(pt, pt, pt, pt) and inside_convex(pt,
poly) to use "Where" enum (?)

template<typename T>
bool inside(Point<T> const& p, Point<T> const& a, Point<T>
const& b, Point<T> const& c) {
  int x = side(p, a, b);
  int y = side(p, b, c);
  int z = side(p, c, a);
  return !((x == 1 or y == 1 or z == 1) and (x==-1 or y == -1
or z == -1));
}

template<typename T>
using Poly = vector<Point<T>>;

template<typename T>
bool inside_convex(Point<T> const& p, Poly<T> const& poly) {
  int bl = 2, br = (int)size(poly) - 1;
  while (bl < br) {
    /* int bm = midpoint(bl, br); */
    int bm = (bl+br)/2;
    if (side(p, poly[0], poly[bm]) == 1) bl = bm+1;
    else br = bm;
  }

  return inside(p, poly[0], poly[br-1], poly[br]);
}

enum Where {
  Inside = 1,
  Outside = -1,
  Boundary = 0,
};

template<typename T>
```

```
Where inside_simple(Point<T> const& p, Poly<T> const& A) {
  int N = size(A);
  int w = 0;
  for (int i = 0; i < N; i++) {
    if (p == A[i]) return Boundary;
    int j = (i+1)%N;
    if (A[i].y == p.y && A[j].y == p.y) {
      if (min(A[i].x, A[j].x) <= p.x && p.x <= max(A[i].x,
A[j].x))
        return Boundary;
    } else {
      bool ibelow = A[i].y < p.y;
      bool jbelow = A[j].y < p.y;
      if (ibelow != jbelow) {
        auto o = side(p, A[i], A[j]);
        if (o == 0) return Boundary;
        if (ibelow == (o > 0)) w += (ibelow ? +1 : -1);
      }
    }
  }

  return (w ? Inside : Outside);
}

template<typename T>
long double sarea(Poly<T> const& P) {
  int N = size(P);
  long double total = 0;
  for (int i = 0; i < N; i++) {
    total += P[i].x * P[(i+1)%N].y;
    total -= P[i].y * P[(i+1)%N].x;
  }
  return total/2;
}

template<typename T>
long double area(Poly<T> const& P) {
  return abs(sarea(P));
}

template<typename T>
bool clockwise(Poly<T> const& P) {
  return sarea(P) < 0;
}

template<typename T>
Poly<T> convex_hull(Poly<T> P) {
  sort(begin(P), end(P));

  vector<Point<T>> L, U;
  for (auto p : P) {
    while (size(L) >= 2 && side(p, end(L)[-2], end(L)[-1]) !=
1) L.pop_back();
    L.push_back(p);
  }
  reverse(begin(P), end(P));
  for (auto p : P) {
    while (size(U) >= 2 && side(p, end(U)[-2], end(U)[-1]) !=
1) U.pop_back();
    U.push_back(p);
  }
  L.pop_back();
  L.insert(end(L), begin(U), end(U)-1);
  return L;
}

template<typename T>
T polygon_cut_length(Poly<T> const& A, Line<T> const& l) {
  int N = size(A);
  auto a = l.p1, b = l.p2;

  T ans{};
  for (int i = 0; i < N; i++) {
    int j = (i+1)%N;

    int si = side(A[i], a, b);
    int sj = side(A[j], a, b);

    if (si == 0 && sj == 0) {
      if ((b-a)*(A[j]-A[i]) > 0) {
        ans += proj_len(A[j], a, b);
        ans -= proj_len(A[i], a, b);
      }
    } else if (si <= 0 && sj > 0) {
      auto it = inter_line(l, {A[i], A[j]}).front();
      ans -= proj_len(it, a, b);
    } else if (si > 0 && sj <= 0) {
      auto it = inter_line(l, {A[i], A[j]}).front();
      ans += proj_len(it, a, b);
    }
  }

  return abs(ans);
}


template<typename T>
pair<Point<T>, Point<T>> closest_pair(vector<Point<T>> P) {
  const long double CP_INF = 1e18;

  pair<long double, pair<Point<T>, Point<T>>> best;
  best.first = CP_INF;

  set<Point<T>> S;
  sort(begin(P), end(P));

  int il = 0;
  for (int ir = 0; ir < size(P); ir++) {
    if (ir && P[ir] == P[ir-1]) return {P[ir], P[ir-1]};
    while (P[ir].x-P[il].x >= best.first) {
      S.erase(transp(P[il]));
      il++;
    }
    for (auto it = S.upper_bound({P[ir].y-best.first,
CP_INF});
         it != end(S) && it->x < P[ir].y+best.first;
         it++) {
      auto p = transp(*it);
      best = min(best, {norm(P[ir], p), pair{P[ir], p}});
    }
    S.insert(transp(P[ir]));
  }
  return best.second;
}
//}}}
```

Graphs

bfs.cpp

```
// Breadth First Search {{{
vector<int> bfs(vector<vector<int>> const& G, int source) {
  vector<int> dist(size(G), -1);
  queue<int> Q;
  dist[source] = 0;
  Q.push(source);

  while (!Q.empty()) {
    int v = Q.front(); Q.pop();
    for (auto u : G[v]) {
      if (dist[u] == -1) {
        dist[u] = dist[v] + 1;
        Q.push(u);
      }
    }
  }

  return dist;
}
//}}}
```

block_cut_tree.cpp

```
// Block-Cut Tree {{{
struct BlockCutTree {
  int N;
  vector<vector<int>> const& G;

  stack<pair<int, int>> S;
  int TIMER = -1;
  vector<int> pre, low;

  vector<int> art;
  vector<bool> is_art;
  vector<vector<pair<int, int>>> bcc;
```

```
vector<vector<int>> BCT;
vector<int> comp;

void make_bcc(pair<int, int> until) {
  bcc.push_back({});
  pair<int, int> e{-1, -1};
  while (e != until) {
    e = S.top(); S.pop();
    bcc.back().push_back(e);
  }
}

void dfs(int v, int p) {
  pre[v] = low[v] = ++TIMER;

  int children = 0;
  bool low_child = false;

  for (auto u : G[v]) {
    if (u == p) continue;
    if (pre[u] == -1) {
      S.push({v, u});
      dfs(u, v);
      children++;

      low[v] = min(low[v], low[u]);
      if (low[u] >= pre[v]) {
        low_child = true;
        make_bcc({v, u});
      }
    } else {
      low[v] = min(low[v], pre[u]);
    }
  }

  if ((p == -1 && children >= 2) || (p != -1 && low_child))
    art.push_back(v);
}


  BlockCutTree(vector<vector<int>> const& G) : G(G),
N(size(G)) {
    pre.assign(N, -1);
    low.assign(N, -1);
    for (int i = 0; i < N; i++) {
      if (pre[i] == -1) {
        dfs(i, -1);
      }
    }

    is_art.resize(N, false);
    for (auto v : art) is_art[v] = true;

    BCT.resize(N + size(bcc));
    comp.resize(N);
    for (int i = 0; i < size(bcc); i++) {
      for (auto [v, u] : bcc[i]) {
        if (is_art[v] && (empty(BCT[v]) || BCT[v].back() !=
N+i)) BCT[v].push_back(N+i), BCT[N+i].push_back(v);
        if (is_art[u] && (empty(BCT[u]) || BCT[u].back() !=
N+i)) BCT[u].push_back(N+i), BCT[N+i].push_back(u);
        comp[v] = comp[u] = N+i;
      }
    }

    for (auto v : art) comp[v] = v;
  }
};
//}}}
```

dijkstra.cpp

```
// Dijkstra {{{
vector<int> dijkstra(vector<vector<pair<int, int>>> const& G,
int source) {
  vector<int> dist(size(G), -1);

  min_priority_queue<pair<int, int>> Q;
  dist[source] = 0;
  Q.push({dist[source], source});

  while (!Q.empty()) {
    auto [d, v] = Q.top(); Q.pop();
    if (d > dist[v]) continue;

    for (auto [u, d] : G[v]) {
      if (dist[v] + d  < dist[u] || dist[u] == -1) {
        dist[u] = dist[v] + d;
        Q.push({dist[u], u});
      }
    }
  }

  return dist;
}
//}}}
```

dinitz.cpp

```
// Dinitz {{{
struct Dinitz {
  struct Edge {
    int v, u, cap, flow=0;
    Edge (int v, int u, int cap) : v(v), u(u), cap(cap) {}
  };

  vector<Edge> edges;
  vector<vector<int>> adj;

  int n, s, t;
  Dinitz(int n, int s, int t) : n(n), s(s), t(t) {
    adj.resize(n);
  }

  void add_edge(int v, int u, int cap) {
    edges.emplace_back(v, u, cap);
    adj[v].push_back(edges.size()-1);
    edges.emplace_back(u, v, 0);
    adj[u].push_back(edges.size()-1);
  }

  vector<int> level;
  bool bfs() {
    queue<int> Q;
    level.assign(n, -1);
    level[s] = 0;
    Q.push(s);
    while (!Q.empty()) {
      int v = Q.front(); Q.pop();
      for (auto eid : adj[v]) {
        auto e = edges[eid];
        if (e.cap - e.flow <= 0) continue;
        if (level[e.u] != -1) continue;
        level[e.u] = level[v] + 1;
        Q.push(e.u);
      }
    }
    return level[t] != -1;
  }

  vector<int> ptr;
  int dfs(int v, int f) {
    if (f == 0 || v == t) return f;
    for (int &cid = ptr[v]; cid < adj[v].size(); cid++) {
      int eid = adj[v][cid];
      auto &e = edges[eid];
      if (e.cap - e.flow <= 0) continue;
      if (level[e.u] != level[e.v] + 1) continue;
      int newf = dfs(e.u, min(f, e.cap-e.flow));
      if (newf == 0) continue;
      e.flow += newf;
      edges[eid^1].flow -= newf;
      return newf;
    }
    return 0;
  }

  int flow() {
    int f = 0;
    while (bfs()) {
      ptr.assign(n, 0);
      while (int newf = dfs(s, INF))
        f += newf;
    }
    return f;
  }
};
//}}}
```

dinitz_mincost.cpp

```cpp
// Dinitz Min Cost {{{
const int INF = 0x3f3f3f3f3f3f3f3f;

struct Dinitz {
  struct Edge {
    int v, u, cap, flow=0, cost;
    Edge(int v, int u, int cap, int cost) : v(v), u(u),
cap(cap), cost(cost) {}
  };

  int n, s, t;
  Dinitz(int n, int s, int t) : n(n), s(s), t(t) {
    adj.resize(n);
  }

  vector<Edge> edges;
  vector<vector<int>> adj;
  void add_edge(int v, int u, int cap, int cost) {
    edges.emplace_back(v, u, cap, cost);
    adj[v].push_back(size(edges)-1);
    edges.emplace_back(u, v, 0, -cost);
    adj[u].push_back(size(edges)-1);
  }

  vector<int> dist;
  bool spfa() {
    dist.assign(n, INF);

    queue<int> Q;
    vector<bool> inqueue(n, false);

    dist[s] = 0;
    Q.push(s);
    inqueue[s] = true;

    vector<int> cnt(n);

    while (!Q.empty()) {
      int v = Q.front(); Q.pop();
      inqueue[v] = false;

      for (auto eid : adj[v]) {
        auto const& e = edges[eid];
        if (e.cap - e.flow <= 0) continue;
        if (dist[e.u] > dist[e.v] + e.cost) {
          dist[e.u] = dist[e.v] + e.cost;
          if (!inqueue[e.u]) {
            Q.push(e.u);
            inqueue[e.u] = true;
          }
        }
      }
    }
    return dist[t] != INF;
  }

  int cost = 0;
  vector<int> ptr;
  int dfs(int v, int f) {
    if (v == t || f == 0) return f;
    for (auto &cid = ptr[v]; cid < size(adj[v]);) {
      auto eid = adj[v][cid];
      auto &e = edges[eid];
      cid++;
      if (e.cap - e.flow <= 0) continue;
      if (dist[e.v] + e.cost != dist[e.u]) continue;
      int newf = dfs(e.u, min(f, e.cap-e.flow));
      if (newf == 0) continue;
      e.flow += newf;
      edges[eid^1].flow -= newf;
      cost += e.cost * newf;
      return newf;
    }
    return 0;
  }

  int total_flow = 0;
  int flow() {
    while (spfa()) {
      ptr.assign(n, 0);
      while (int newf = dfs(s, INF))
        total_flow += newf;
    }
    return total_flow;
  }
};
//}}}
```

eulerian_cycle.cpp

```cpp
// Eulerian Cycle {{{
pair<bool, vector<int>> eulerian_cycle(int N, vector<pair<int,
int>> const& E) {
  int M = size(E);

  vector<vector<pair<int, int>>> G(N);
  for (int i = 0; i < M; i++) {
    auto [v, u] = E[i];
    G[v].push_back({u, i});
    G[u].push_back({v, i});
  }

  for (int i = 0; i < N; i++)
    if (size(G[i]) % 2)
      return {false, {}};

  vector<int> path;
  vector<bool> seen(M);

  auto dfs = [&](auto &&F, int v) -> void {
    while (!G[v].empty()) {
      auto [u, idx] = G[v].back();
      G[v].pop_back();
      if (seen[idx]) continue;
      seen[idx] = true;
      F(F, u);
    }
    path.push_back(v);
  };
  dfs(dfs, 0);

  if (size(path) != M+1) return {false, {}};

  reverse(begin(path), end(path));
  return {true, path};
}
//}}}
```

hopcroft_karp.cpp

```cpp
// Hopcroft-Karp {{{
struct HopcroftKarp {
  const int NONE_R;
  const int INF = 1e9 + 8;

  int L, R;
  int ans = 0;
  vector<int> ml, mr;
  vector<int> lvl;
  vector<vector<int>> const& G;

  bool bfs() {
    queue<int> Q;
    for (int l = 0; l < size(ml); l++) {
      if (ml[l] == -1) {
        lvl[l] = 0;
        Q.push(l);
      } else {
        lvl[l] = INF;
      }
    }
    lvl[NONE_R] = INF;
    while (!empty(Q)) {
      int l = Q.front();
      Q.pop();
      if (lvl[l] < lvl[NONE_R]) {
        for (auto r : G[l]) {
          if (lvl[mr[r]] == INF) {
            lvl[mr[r]] = lvl[l] + 1;
            Q.push(mr[r]);
          }
        }
      }
    }
    return lvl[NONE_R] != INF;
```

```
  }

  bool dfs(int l) {
    if (l == NONE_R) return true;
    for (auto r : G[l]) {
      if (lvl[mr[r]] == lvl[l] + 1) {
        if (dfs(mr[r])) {
          ml[l] = r;
          mr[r] = l;
          return true;
        }
      }
    }
    lvl[l] = INF;
    return false;
  }

  HopcroftKarp(int _L, int _R, vector<vector<int>> const& G) :
L(_L), R(_R), NONE_R(_L), G(G) {
    ml.assign(L, -1);
    mr.assign(R, NONE_R);
    lvl.assign(L+1, -1);

    while (bfs()) {
      for (int l = 0; l < L; l++) {
        if (ml[l] == -1) {
          if (dfs(l)) ans++;
        }
      }
    }
  }
};
//}}}
```

kosaraju.cpp

```
// Kosaraju {{{
struct Kosaraju {
  int N;
  vector<vector<int>> const& G;
  vector<vector<int>> Ginv;
  vector<bool> vis;
  stack<int> S;

  vector<int> comp;
  vector<vector<int>> comps;

  Kosaraju(vector<vector<int>> const& G)
    : N(size(G)), G(G), Ginv(N), vis(N), comp(N, -1) {
    for (int i = 0; i < N; i++) {
      for (auto u : G[i]) {
        Ginv[u].push_back(i);
      }
    }

    for (int i = 0; i < N; i++) if (!vis[i]) dfs(i);
    fill(begin(vis), end(vis), false);
    int cc = 0;
    while (!S.empty()) {
      int v = S.top();
      S.pop();
      if (!vis[v]) {
        comps.push_back({});
        scc(v, cc++);
      }
    }
  }

  void dfs(int v) {
    vis[v] = true;
    for (auto u : G[v]) if (!vis[u]) dfs(u);
    S.push(v);
  }

  void scc(int v, int c) {
    vis[v] = true;
    comp[v] = c;
    comps.back().push_back(v);
    for (auto u : Ginv[v]) if (!vis[u]) scc(u, c);
  }
};
// }}}
```

lca.cpp

```
// Lowest Common Ancestor {{{
struct LCA {
  const int LOG = 22;

  int N;
  vector<vector<int>> const& G;

  int TIMER = -1;
  vector<int> pre, pos, dep;
  vector<vector<int>> parent;

  void dfs(int v, int p) {
    parent[v][0] = p;
    for (int b = 1; b < LOG; b++) {
      parent[v][b] = parent[parent[v][b-1]][b-1];
    }

    pre[v] = ++TIMER;
    for (auto u : G[v]) {
      if (u == p) continue;
      dep[u] = dep[v] + 1;
      dfs(u, v);
    }
    pos[v] = TIMER;
  }

  bool is_ancestor(int anc, int child) {
    return pre[anc] <= pre[child] && pos[child] <= pos[anc];
  }

  int lca(int v, int u) {
    if (is_ancestor(v, u)) return v;
    if (is_ancestor(u, v)) return u;

    for (int b = LOG-1; b >= 0; b--) {
      int nv = parent[v][b];
      if (!is_ancestor(nv, u)) v = nv;
    }
    v = parent[v][0];
    return v;
  }

  int dist(int v, int u) {
    int l = lca(v, u);
    int dist = dep[v] + dep[u] - 2*dep[l];
    return dist;
  }

  LCA (vector<vector<int>> const& G) : G(G), N(size(G)) {
    pre.assign(N, -1);
    pos.assign(N, -1);
    dep.assign(N, 0);
    parent.resize(N, vector<int>(LOG));
    for (int i = 0; i < N; i++) {
      if (pre[i] == -1) {
        dfs(i, i);
      }
    }
  }
};
//}}}
```

tarjan.cpp

```
// Tarjan {{{
struct Tarjan {
  int N;
  vector<vector<int>> const& G;
  vector<int> comp;
  vector<vector<int>> comps;

  Tarjan(vector<vector<int>> const& G) : G(G) {
    N = size(G);
    pre.assign(N, -1);
    low.assign(N, -1);
    comp.assign(N, -1);

    for (int i = 0; i < N; i++) {
      if (pre[i] == -1) {
        dfs(i);
      }
```

```
    }
  }

  void make_comp(int v) {
    comps.push_back({});
    while (true) {
      int x = S.top();
      S.pop();
      comps.back().push_back(x);
      comp[x] = size(comps)-1;
      if (x == v) break;
    }
  }

  int TIMER = -1;
  vector<int> pre, low;
  vector<bool> used;
  stack<int> S;
  void dfs(int v) {
    S.push(v);
    pre[v] = low[v] = ++TIMER;

    for (auto u : G[v]) {
      if (pre[u] == -1) {
        dfs(u);
        low[v] = min(low[v], low[u]);
      } else if (comp[u] == -1) {
        low[v] = min(low[v], pre[u]);
      }
    }

    if (pre[v] == low[v]) make_comp(v);
  }
};
//}}}
```

topological_sort.cpp

```
// Topological Sort {{{
// If v points to u, v comes before u
vector<int> topo(vector<vector<int>> G) {
  int N = size(G);

  vector<int> din(N);
  for (int i = 0; i < N; i++) {
    for (auto u : G[i]) {
      din[u]++;
    }
  }

  queue<int> Q;
  for (int i = 0; i < N; i++) {
    if (din[i] == 0) {
      Q.push(i);
    }
  }

  vector<int> topo;
  while (!empty(Q)) {
    auto v = Q.front();
    Q.pop();

    topo.push_back(v);

    for (auto u : G[v]) {
      if (!--din[u]) {
        Q.push(u);
      }
    }
  }

  return topo;
}
//}}}
```

tour.cpp

```
// Euler Tour {{{
struct Tour {
  int source;
  vector<vector<int>> const& G;

  int TIMER = -1;
  vector<int> pre, pos, dep, who;

  Tour (vector<vector<int>> const& G, int source) : G(G),
source(source) {
    int N = size(G);
    pre.assign(N, -1);
    pos.assign(N, -1);
    dep.assign(N, -1);
    who.assign(N, -1);
    dep[source] = 0;
    walk(source);
  }

  void walk(int v) {
    pre[v] = ++TIMER;
    who[pre[v]] = v;
    for (auto u : G[v]) {
      if (pre[u] == -1) {
        dep[u] = dep[v] + 1;
        walk(u);
      }
    }
    pos[v] = TIMER;
  }
};
//}}}
```

two_sat.cpp

```
// TwoSat {{{
struct TwoSat {
  int N;
  vector<vector<int>> G;
  TwoSat(int N) : N(N), G(2*N) {}

  int neg(int u) { return u + ((u < N) ? N : -N); }
  void add_or(int v, int u) {
    G[neg(v)].push_back(u);
    G[neg(u)].push_back(v);
  }
  void add_diff(int v, int u) {
    add_or(v, u);
    add_or(neg(v), neg(u));
  }
  void add_impl(int v, int u) {
    add_or(neg(v), u);
  }
  void add_same(int v, int u) {
    add_impl(v, u);
    add_impl(neg(v), neg(u));
  }
  void add_true(int v) {
    add_or(v, v);
  }
  void add_false(int v) {
    add_or(neg(v), neg(v));
  };

  // Assumes Kosaraju returns components in topological
ordering v -> u implies scc[v] <= scc[u]
  pair<bool, vector<bool>> solve() {
    vector<bool> res(N);
    auto scc = Kosaraju(G).comp;
    for (int i = 0; i < N; i++) {
      if (scc[i] == scc[neg(i)]) return {false, {}};
      res[i] = scc[i] < scc[neg(i)];
    }
    return {true, res};
  }
};
//}}}
```

Math

baby_steps.cpp

```
// Baby Steps, Giant Steps {{{
// https://cp-algorithms.com/algebra/discrete-log.html
```

```cpp
// Returns minimum x for which a ^ x % m = b % m, a and m are
coprime.
int babysteps(int a, int b, int m) {
  a %= m, b %= m;
  int n = sqrt(m) + 1;

  int an = 1;
  for (int i = 0; i < n; ++i)
    an = (an * 1ll * a) % m;

  unordered_map<int, int> vals;
  for (int q = 0, cur = b; q <= n; ++q) {
    vals[cur] = q;
    cur = (cur * 1ll * a) % m;
  }

  for (int p = 1, cur = 1; p <= n; ++p) {
    cur = (cur * 1ll * an) % m;
    if (vals.count(cur)) {
      int ans = n * p - vals[cur];
      return ans;
    }
  }
  return -1;
}
//}}}
```

chinese_remainder_theorem.cpp

```cpp
// Chinese Remainder Theorem {{{
struct CRT {
  int A, M;
  CRT() : A(0), M(1) {}
  CRT(int A, int M) : A(A), M(M) {}
  CRT operator*(CRT const& C) {
    auto [g, x, y] = ext_gcd(M, C.M);
    if ((A - C.A) % g) A = -1;
    if (A == -1 || C.A == -1) return CRT(-1, 0);
    int L = M/g*C.M;
    int ans = A + (x * (C.A-A))/g % (C.M/g) * M;
    return CRT((ans % L + L) % L, L);
  }

  int count(int r) const {
    if (r < 0) return 0;
    int total = r/M;
    r %= M;
    if (r >= A) total++;
    return total;
  }

  int count(int l, int r) const {
    return count(r) - count(l-1);
  }
};
//}}}
```

combinatorics.cpp

```cpp
// Combinatorics {{{
template <unsigned P>
struct Combinatorics {
  vector<Z<P>> fact, ifact;

  explicit Combinatorics(int N) : fact(N), ifact(N) {
    fact[0] = 1;
    for (int i = 1; i < N; i++) fact[i] = fact[i-1] * i;
    ifact[N-1] = 1 / fact[N-1];
    for (int i = N-1; i-1 >= 0; i--) ifact[i-1] = ifact[i] *
i;
  }

  Z<P> C(int n, int k) const {
    return k < 0 || n < k ?  0 : fact[n] * ifact[k] * ifact[n-
k];
  }

  Z<P> S(int n, int k) const {
    return k == 0 ? n == 0 : C(n + k - 1, k - 1);
  }
};
//}}}
```

ext_gcd.cpp

```cpp
// Extended GCD {{{
tuple<int, int, int> ext_gcd(int a, int b) {
  if (b == 0) return {a, 1, 0};
  auto [g, x, y] = ext_gcd(b, a%b);
  return {g, y, x - (a/b) * y};
}

tuple<bool, int, int> dio(int a, int b, int c) {
  auto [g, x, y] = ext_gcd(a, b);
  if (c % g) return {false, -1, -1};
  return {true, x * (c/g), y * (c/g)};
}
//}}}
```

factor.cpp

```cpp
// Factor {{{
vector<pair<int, int>> factor(int N) {
  vector<pair<int, int>> F;
  for (auto p : primes) {
    if (p*p > N) break;
    if (N % p == 0) {
      F.push_back({p, 0});
      while (N % p == 0) {
        N /= p;
        F.back().second++;
      }
    }
  }
  if (N != 1) F.push_back({N, 1});
  return F;
}
//}}}
```

fexp.cpp

```cpp
// Fast Exponentiation {{{
int fexp(int b, int e) {
  b %= MOD;
  int ans = 1;
  while (e) {
    if (e & 1) (ans *= b) %= MOD;
    e >>= 1;
    (b *= b) %= MOD;
  }
  return ans;
}
//}}}
```

fft.cpp

```cpp
// FFT {{{
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> &A, bool invert) {
  int N = size(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 len = 2; len <= N; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < N; i += len) {
      cd w(1);
      for (int j = 0; j < len/2; j++) {
        cd u = A[i+j], v = A[i+j+len/2] * w;
        A[i+j] = u + v;
        A[i+j+len/2] = u-v;
        w *= wlen;
      }
    }
  }
```

```
  if (invert) {
    for (auto &x : A)
      x /= N;
  }
}

vector<int> multiply(vector<int> const& A, vector<int> const&
B) {
  vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
  int N = 1;
  while (N < size(A) + size(B))
    N <<= 1;
  fa.resize(N);
  fb.resize(N);

  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < N; i++)
    fa[i] *= fb[i];
  fft(fa, true);

  vector<int> result(N);
  for (int i = 0; i < N; i++)
    result[i] = round(fa[i].real());
  return result;
}
// }}}
```

floor_sum.cpp

```
// Floor Sum {{{
// sum of floor[i=0...n-1]((a*i+b)/m)
unsigned long long __floor_sum_unsigned(unsigned long long n,
                                        unsigned long long m,
                                        unsigned long long a,
                                        unsigned long long b) {
  unsigned long long ans = 0;
  while (true) {
    if (a >= m) {
      ans += n * (n - 1) / 2 * (a / m);
      a %= m;
    }
    if (b >= m) {
      ans += n * (b / m);
      b %= m;
    }

    unsigned long long y_max = a * n + b;
    if (y_max < m) break;
    // y_max < m * (n + 1)
    // floor(y_max / m) <= n
    n = (unsigned long long)(y_max / m);
    b = (unsigned long long)(y_max % m);
    swap(m, a);
  }
  return ans;
}

long long floor_sum(long long n, long long m, long long a,
long long b) {
  unsigned long long ans = 0;
  if (a < 0) {
    unsigned long long a2 = (a%m+m)%m;
    ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
    a = a2;
  }
  if (b < 0) {
    unsigned long long b2 = (a%m+m)%m;
    ans -= 1ULL * n * ((b2 - b) / m);
    b = b2;
  }
  return ans + __floor_sum_unsigned(n, m, a, b);
}
//}}}
```

lagrange.cpp

```
// Lagrange Interpolation {{{
mint eval_interpolation(int X, vector<mint> const& Y) {
  int N = size(Y);

  vector<mint> pref(N), suff(N);
  for (int i = 0; i < N; i++) pref[i] = suff[i] = X-i;

  auto mult = [](auto x, auto y) { return x*y; };
  partial_sum(begin(pref), end(pref), begin(pref), mult);
  partial_sum(rbegin(suff), rend(suff), rbegin(suff), mult);

  mint ans = 0;
  for (int i = 0; i < N; i++) {
    mint num = Y[i];
    if (i-1 >= 0) num *= pref[i-1];
    if (i+1 < N) num *= suff[i+1];

    mint den = 1;
    den *= C.ifact[i];
    if (N-1-i >= 0) {
      den *= C.ifact[N-1-i];
      if ((N-1-i) % 2 == 1) den *= -1;
    }

    ans += num * den;
  }

  return ans;
}
//}}}
```

matrix_exp.cpp

```
// Matrix Exponentiation {{{
struct Matrix {
  int N;
  vector<vector<mint>> M;
  Matrix (int N) : N(N), M(N, vector<mint>(N)) {}
  Matrix operator*(Matrix const& rhs) {
    Matrix result(N);
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
        for (int k = 0; k < N; k++) {
          result.M[i][j] += M[i][k] * rhs.M[k][j];
        }
      }
    }
    return result;
  }
};

Matrix ident(int N) {
  Matrix M(N);
  for (int i = 0; i < N; i++) M.M[i][i] = 1;
  return M;
}

Matrix pow(Matrix const& M, int e) {
  if (e == 0) return ident(M.N);
  Matrix h = pow(M, e/2);
  h = h * h;
  if (e % 2) h = h * M;
  return h;
}
//}}}
```

modular.cpp

```
// Z_P (Modular Arithmetic) {{{
template <unsigned P>
struct Z {
  unsigned value;

  constexpr Z() : value(0) {}

  template<typename T, typename =
enable_if_t<std::is_integral<T>::value>>
  constexpr Z(T a) : value((((long long)a % P) + P) % P) {}

  Z& operator+=(Z rhs) {
    value += rhs.value;
    if (value >= P) value -= P;
    return *this;
  }

  Z& operator-=(Z rhs) {
    value += P - rhs.value;
```

```
    if (value >= P) value -= P;
    return *this;
  }

  Z& operator*=(Z rhs) {
    value = (unsigned long long)value * rhs.value % P;
    return *this;
  }

  Z& operator/=(Z rhs) { return *this *= pow(rhs, -1); }

  Z operator+() const { return *this; }

  Z operator-() const { return Z() - *this; }

  bool operator==(Z rhs) const { return value == rhs.value; }

  bool operator!=(Z rhs) const { return value != rhs.value; }

  friend Z operator+(Z lhs, Z rhs) { return lhs += rhs; }

  friend Z operator-(Z lhs, Z rhs) { return lhs -= rhs; }

  friend Z operator*(Z lhs, Z rhs) { return lhs *= rhs; }

  friend Z operator/(Z lhs, Z rhs) { return lhs /= rhs; }

  friend ostream& operator<<(ostream& out, Z a) { return out
<< a.value; }

  friend istream& operator>>(istream& in, Z& a) {
    long long x;
    in >> x;
    a = Z(x);
    return in;
  }
};

template<unsigned P>
Z<P> pow(Z<P> x, long long p) {
  if (x == 0) {
    return p == 0 ? 1 : 0;
  }
  p %= P -1;
  if (p < 0) p += P-1;
  Z<P> res = 1;
  while (p) {
    if (p & 1) {
      res *= x;
    }
    x *= x;
    p >>= 1;
  }
  return res;
}
//}}}
```

modular_mini.cpp

```
int mul(int a, int b) { return (a * b) % MOD; }
int add(int a, int b) { return (a + b) % MOD; }
int fexp(int b, int e=MOD-2) {
  b %= MOD;
  int ans = 1;
  while (e) {
    if (e & 1) ans = mul(ans, b);
    e >>= 1;
    b = mul(b, b);
  }
  return ans;
}
```

next_array.cpp

```
bool next_array(vector<int> &A, int limit) {
  A.back()++;
  for (int i = size(A)-1; i >= 0; i--) {
    if (A[i] > limit) {
      A[i] = 0;
      if (i-1 >= 0) A[i-1]++;
      else {
        return false;
      }
    }
  }
  return true;
}
```

non_prime_multinv.cpp

```
// Non-prime Multiplicative Inverse {{{
int multinv(int A, int P) {
  auto [g, x, y] = ext_gcd(A, P);
  if (g != 1) return -1;
  x = (x % P + P) % P;
  return (x % P + P) % P;
}
//}}}
```

ntt.cpp

```
// NTT {{{
const int MOD = 998'244'353; // 7*17*2^23 + 1
const int G = 3;
const int ROOT_23 = 15311432; // G ^ (7 * 17)
const int ROOT_23_INV = 469870224; // ROOT_23 ^ -1
const int ROOT_POW = (1 << 23);

int fexp(int b, int e) {
  int ans = 1;
  while (e) {
    if (e & 1) ans = ans * b % MOD;
    e >>= 1;
    b = b * b % MOD;
  }
  return ans;
}

int multinv(int x) {
  return fexp(x, MOD-2);
}

void fft(vector<int> &A, bool invert) {
  int N = A.size();

  for (int i = 1, j = 0; i < N; i++) {
    int bit = N/2;
    while (j & bit) j ^= bit, bit >>= 1;
    j ^= bit;
    if (i < j) swap(A[i], A[j]);
  }

  for (int len = 2; len <= N; len <<= 1) {
    int wlen = invert ? ROOT_23_INV : ROOT_23;
    for (int i = len; i < ROOT_POW; i <<= 1)
      wlen = (wlen * wlen) % MOD;

    for (int i = 0; i < N; i += len) {
      int w = 1;
      for (int j = 0; j < len/2; j++) {
        int u = A[i+j];
        int v = A[i+j+len/2] * w % MOD;
        A[i+j] = u + v;
        if (A[i+j] >= MOD) A[i+j] -= MOD;
        A[i+j+len/2] = u - v;
        if (A[i+j+len/2] < 0) A[i+j+len/2] += MOD;
        w = w * wlen % MOD;
      }
    }
  }

  if (invert) {
    int N_INV = multinv(N);
    for (auto &x : A) x = x * N_INV % MOD;
  }
}

vector<int> multiply(vector<int> const& A, vector<int> const&
B) {
  vector<int> fa(begin(A), end(A)), fb(begin(B), end(B));
  int N = 1;
  while (N < A.size() + B.size()) N <<= 1;
  fa.resize(N);
  fb.resize(N);

  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < N; i++)
    fa[i] = fa[i] * fb[i] % MOD;
```

```
  fft(fa, true);

  while (fa.back() == 0) fa.pop_back();

  return fa;
}
//}}}
```

primitive_root.cpp

```
// Primitive Root {{{
// Assume P is prime
int fexp_mod(int b, int e, const int P) {
  int ans = 1;
  while (e) {
    if (e & 1) ans = ans * b % P;
    e >>= 1;
    b = b * b % P;
  }
  return ans;
}

int generator(const int P) {
  vector<int> pfact;
  int phi = P-1, N = phi;
  for (int i = 2; i*i <= N; i++) {
    if (N % i == 0) {
      pfact.push_back(i);
      while (N % i == 0) N /= i;
    }
  }
  if (N > 1) pfact.push_back(N);

  for (int ans = 2; ans <= P; ans++) {
    bool ok = true;
    for (int i = 0; i < pfact.size() && ok; i++)
      ok &= fexp_mod(ans, phi/pfact[i], P) != 1;
    if (ok) return ans;
  }
  return -1;
}
// }}}
```

psum.cpp

```
// Prefix Sum {{{
int psum(vector<int> const& A, int l, int r) {
  if (r < l) return 0;
  return A[r] - (l ? A[l-1] : 0);
}
// }}}
```

psum_2d.cpp

```
// Prefix Sum 2d {{{
void partial_sum(vector<vector<int>> &A) {
  for (int i = 0; i < size(A); i++) {
    for (int j = 0; j < size(A.front()); j++) {
      if (i) A[i][j] += A[i-1][j];
      if (j) A[i][j] += A[i][j-1];
      if (i && j) A[i][j] -= A[i-1][j-1];
    }
  }
}

int psum(vector<vector<int>> const& A, int i1, int j1, int i2,
int j2) {
  if (i1 > i2 || j1 > j2) return 0;
  int sum = A[i2][j2];
  if (i1) sum -= A[i1-1][j2];
  if (j1) sum -= A[i2][j1-1];
  if (i1 && j1) sum += A[i1-1][j1-1];
  return sum;
}
// }}}
```

sieve.cpp

```
// Sieve of Eratosthenes {{{
vector<int> sieve(const int MAX) {
  vector<bool> prime(MAX+1, true);
  prime[0] = prime[1] = false;

  vector<int> primes;
  for (int i = 2; i <= MAX; i++) {
    if (prime[i]) {
      primes.push_back(i);
      for (int j = 2*i; j <= MAX; j+=i) {
        prime[j] = false;
      }
    }
  }
  return primes;
}
//}}}
```

xor_basis.cpp

```
// XOR Basis {{{
struct Basis {
  vector<int> B;
  int reduce(int x) {
    for (auto b : B) x = min(x, x^b);
    return x;
  }
  void insert(int x) {
    int r = reduce(x);
    if (r) B.push_back(r);
  }
};
//}}}
```

xor_hash.cpp

```
// XOR Hash {{{
unsigned long long mix(unsigned long long o){
  o+=0x9e3779b97f4a7c15;
  o=(o^(o>>30))*0xbf58476d1ce4e5b9;
  o=(o^(o>>27))*0x94d049bb133111eb;
  return o^(o>>31);
}
//}}}
```

String

---

aho_corasick.cpp

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

#define int long long

signed main() {
  ios::sync_with_stdio(false); cin.tie(nullptr);

}

// Aho-Corasick {{{
struct AhoCorasick {
  const int MALPHA = 26;

  vector<vector<int>> trie{vector<int>(MALPHA, -1)};
  vector<int> depth{0};
  vector<bool> is_word{0};
  vector<int> suffix_link;
  vector<int> output_link;

  explicit AhoCorasick(vector<string> const& v) {
    for (auto const& s : v) {
      insert(s);
    }
    create_links();
  }

private:
  int add_node(int parent) {
    trie.push_back(vector<int>(MALPHA, -1));
    is_word.push_back(false);
    depth.push_back(depth[parent] + 1);
    return size(trie) - 1;
  }

  void insert(string const& s) {
    int cur = 0;
```

```
    for (auto c : s) {
      if (trie[cur][c-'a'] == -1)
        trie[cur][c-'a'] = add_node(cur);
      cur = trie[cur][c-'a'];
    }
    is_word[cur] = true;
  }

  void create_links() {
    suffix_link.assign(size(trie), 0);
    output_link.assign(size(trie), -1);

    queue<int> Q;
    Q.push(0);

    while (!empty(Q)) {
      int v = Q.front();
      Q.pop();

      for (int c = 0; c < MALPHA; c++) {
        int u = trie[v][c];
        if (u == -1) continue;

        if (v == 0) {
          suffix_link[u] = 0;
        } else {
          int cur = suffix_link[v];
          while (trie[cur][c] == -1 && cur != 0) {
            cur = suffix_link[cur];
          }
          if (trie[cur][c] != -1) {
            suffix_link[u] = trie[cur][c];
          }
        }

        Q.push(u);
      }

      int cur = suffix_link[v];
      if (is_word[cur]) {
        output_link[v] = cur;
      } else {
        output_link[v] = output_link[cur];
      }
    }
  }
};
// }}}
```

hashed_string.cpp

```
// Hashed String {{{
class HashedString {
  static const int M = (1LL << 61) - 1;
  static const int B;
  static vector<int> pow;
  int N;
  vector<int> p_hash;

  __int128 mul(int a, int b) { return (__int128)a * b; }
  int mod_mul(int a, int b) { return mul(a, b) % M; }

  public:
  explicit HashedString(string const& s) {
    N = size(s);
    while (size(pow) < size(s) + 1)
pow.push_back(mod_mul(pow.back(), B));

    p_hash.resize(size(s) + 1);
    p_hash[0] = 0;
    for (int i = 0; i < size(s); i++)
      p_hash[i + 1] = (mul(p_hash[i], B) + s[i]) % M;
  }

  int get_hash(int l, int r) {
    int raw_val = p_hash[r + 1] - mod_mul(p_hash[l], pow[r - l
+ 1]);
    return (raw_val + M) % M;
  }

  int prefix(int len) { return get_hash(0, len-1); }
  int suffix(int len) { return get_hash(N-len, N-1); }
  int whole() { return get_hash(0, N-1); }
  int from(int l, int len) {
    int r = l+len-1;
    r = min(r, N-1);
    return get_hash(l, r);
  }
  int to(int r, int len) {
    int l = r-len+1;
    l = max(l, 0LL);
    return get_hash(l, r);
  }
};
vector<int> HashedString::pow{1};
mt19937
rng((uint32_t)chrono::steady_clock::now().time_since_epoch().count());
const int HashedString::B = uniform_int_distribution<int>(0, M
- 1)(rng);
//}}}
```

prefix_automaton.cpp

```
// Prefix Automaton {{{
const int MALPHA = 26;

vector<vector<int>> prefix_automaton(string const& S) {
  auto pi = prefix_function(S);

  int N = size(S);
  vector<vector<int>> A(N+1, vector<int>(MALPHA));
  for (int i = 0; i <= N; i++) {
    for (int c = 0; c < MALPHA; c++) {
      if (i < size(S) && S[i]-'a' == c) {
        A[i][c] = i+1;
      } else {
        if (i == 0) A[i][c] = 0;
        else A[i][c] = A[pi[i-1]][c];
      }
    }
  }

  return A;
}
//}}}
```

prefix_function.cpp

```
// Prefix Function {{{
vector<int> prefix_function(string const& S) {
  int N = size(S);
  vector<int> pi(N);
  for (int i = 1; i < N; i++) {
    int j = pi[i-1];
    while (j > 0 && S[i] != S[j]) j = pi[j-1];
    if (S[i] == S[j]) j++;
    pi[i] = j;
  }
  return pi;
}
//}}}
```

prefix_periods.cpp

```
// Prefix Periods {{{
vector<vector<int>> prefix_periods(string const& S) {
  int N = size(S);
  auto Z = z_function(S);
  Z[0] = N;

  vector<vector<int>> P(N);
  for (int len = 1; 2*len <= N; len++)
    if (Z[0] >= len && Z[len] >= len)
      P[2*len-1].push_back(len);

  for (int i = 0; i < N-1; i++)
    for (auto p : P[i])
      if (Z[i+1] >= p)
        P[i+p].push_back(p);

  return P;
}
//}}}
```

suffix_array.cpp

```
// Suffix Array {{{
vector<int> sort_cyclic_shifts(string const& s) {
  int N = s.size();

  vector<int> p(N);
  iota(begin(p), end(p), 0);
  sort(begin(p), end(p), [&](int x, int y) {
    return s[x] < s[y];
  });

  vector<int> eq(N);
  eq[p[0]] = 0;
  for (int i = 1; i < N; i++)
    eq[p[i]] = eq[p[i - 1]] + (s[p[i]] != s[p[i - 1]]);

  for (int shift = 1; shift < N; shift *= 2) {
    vector<int> cnt(N);
    for (int i = 0; i < N; i++)
      cnt[eq[i]]++;
    partial_sum(begin(cnt), end(cnt), begin(cnt));

    vector<int> tmp(N);
    for (int i = 0; i < N; i++)
      tmp[N - 1 - i] = (p[i] - shift + N) % N;
    for (auto i : tmp)
      p[--cnt[eq[i]]] = i;

    auto key = [&](int x) {
      return pair(eq[x], eq[(x + shift) % N]);
    };

    tmp[p[0]] = 0;
    for (int i = 1; i < N; i++)
      tmp[p[i]] = tmp[p[i - 1]] + (key(p[i]) != key(p[i -
1]));
    swap(tmp, eq);
  }

  return p;
}

vector<int> kasai(string const& s, vector<int> const& p) {
  int N = size(s);
  vector<int> rank(N);
  for (int i = 0; i < N; i++)
    rank[p[i]] = i;

  int k = 0;
  vector<int> lcp(N-1);
  for (int i = 0; i < N; i++) {
    if (rank[i] == N-1) {
      k = 0;
      continue;
    }
    int j = p[rank[i] + 1];
    while (i + k < N && j + k < N && s[i+k] == s[j+k]) k++;
    lcp[rank[i]] = k;
    if (k) k--;
  }

  return lcp;
}
//}}}
```

trie.cpp

```
// Trie {{{
struct Trie {
  const int MALPHA = 26;

  vector<vector<int>> trie{vector<int>(MALPHA, -1)};
  vector<int> word_cnt{0};

  int add_node() {
    trie.push_back(vector<int>(MALPHA, -1));
    word_cnt.push_back(0);
    return size(trie)-1;
  }

  void insert(string const& s) {
    int cur = 0;
    for (auto c : s) {
      if (trie[cur][c-'a'] == -1)
        trie[cur][c-'a'] = add_node();
      cur = trie[cur][c-'a'];
    }
    word_cnt[cur]++;
  }

  int count(string const& s) {
    int cur = 0;
    for (auto c : s) {
      if (trie[cur][c-'a'] == -1)
        return false;
      cur = trie[cur][c-'a'];
    }
    return word_cnt[cur];
  }
};
//}}}
```

z_function.cpp

```
// Z-Function {{{
vector<int> z_function(string const& S) {
  int N = size(S);
  vector<int> z(N);
  int l = 0, r = 0 ;
  for (int i = 1; i < N; i++) {
    if (i < r) z[i] = min(r-i, z[i-l]);
    while (i + z[i] < N && S[z[i]] == S[i+z[i]]) z[i]++;
    if (i + z[i] > r) {
      l = i;
      r = i+z[i];
    }
  }
  return z;
}
//}}}
```