Contents

Sublime Build

```
{
  "cmd" : ["g++ -std=c++20 -DLOCAL -Wall $file_name -o
      prog && timeout 5s ./prog<~/Codes/in>~/Codes/out"],
  "selector" : "source.cpp",
  "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
  "shell": true,
  "working_dir" : "$file_path"
}
```

vimrc

```
set mouse=a
set termguicolors
filetype plugin indent on
syntax on

set smartindent expandtab ignorecase smartcase incsearch
    relativenumber nowrap autoread splitright splitbelow
set tabstop=4       "the width of a tab
set shiftwidth=4    "the width for indent
colorscheme torte

inoremap {<ENTER> {}<LEFT><CR><ESC><S-o>

inoremap jj <ESC>

autocmd filetype cpp map <F5> :wa<CR>:!clear && g++ % -
    DLOCAL -std=c++20 -Wall -Wextra -Wconversion -
    Wshadow -Wfloat-equal -o ~/Codes/prog && (timeout 5
    ~/Codes/prog < ~/Codes/in) > ~/Codes/out<CR>

map <F4> :!xclip -o -sel c > ~/Codes/in <CR><CR>
map <F3> :!xclip -sel c % <CR><CR>
map <F6> :vsplit ~/Codes/in<CR>:split ~/Codes/out<CR><C-w
    >=20<C-w><<C-w><C-h>

:autocmd BufNewFile *.cpp 0r ~/Codes/temp.cpp
set clipboard=unnamedplus
```

Stress-tester

```
#!/bin/bash
# Call as stresstester ITERATIONS [--count]

g++ gen.cpp -o gen
g++ sol.cpp -o sol
g++ brute.cpp -o brute

for i in $(seq 1 "$1") ; do
    echo "Attempt $i/$1"
    ./gen > in.txt
    ./sol < in.txt > out1.txt
    ./brute < in.txt > out2.txt
    diff -y out1.txt out2.txt > diff.txt
    if [ $? -ne 0 ] ; then
        echo "Differing Testcase Found:"; cat in.txt
        echo -e "\nOutputs:"; cat diff.txt
        break
    fi
done
```

1 All Macros

```
/*--- DEBUG TEMPLATE STARTS HERE ---*/
void show(int x) {cerr << x;}
void show(long long x) {cerr << x;}
void show(double x) {cerr << x;}
void show(char x) {cerr << '\'' << x << '\'';}
void show(const string &x) {cerr << '\"' << x << '\"';}
void show(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>
void show(pair<T, V> x) { cerr << '{'; show(x.first);
    cerr << ", "; show(x.second); cerr << '}'; }
template<typename T>
void show(T x) {int f = 0; cerr << "{"; for (auto &i: x)
    cerr << (f++ ? ", " : ""), show(i); cerr << "}";}

void debug_out(string s) {
  s.clear();
  cerr << s << '\n';
}
template <typename T, typename... V>
void debug_out(string s, T t, V... v) {
  s.erase(remove(s.begin(), s.end(), ' '), s.end());
  cerr << "        "; // 8 spaces
  cerr << s.substr(0, s.find(','));
  s = s.substr(s.find(',') + 1);
  cerr << " = ";
  show(t);
  cerr << endl;
  if(sizeof...(v)) debug_out(s, v...);
}
#define dbg(x...) cerr << "LINE: " << __LINE__ << endl;
    debug_out(#x, x); cerr << endl;
/*--- DEBUG TEMPLATE ENDS HERE ---*/
//#pragma GCC optimize("Ofast")
//#pragma GCC optimization ("O3")
//#pragma comment(linker, "/stack:200000000")
//#pragma GCC optimize("unroll-loops")
//#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm
    ,mmx,avx,tune=native")

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
  //find_by_order(k) --> returns iterator to the kth
      largest element counting from 0
  //order_of_key(val) --> returns the number of items in
      a set that are strictly smaller than our item
template <typename DT>
using ordered_set = tree <DT, null_type, less<DT>,
    rb_tree_tag,tree_order_statistics_node_update>;
mt19937 rnd(chrono::steady_clock::now().time_since_epoch
    ().count());

#ifdef LOCAL
#include "dbg.h"
#else
#define dbg(x...)
#endif

int main() {
  cin.tie(0) -> sync_with_stdio(0);
}
```

2 DP

2.1 1D-1D

```
/// Author: anachor

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

/// Solves dp[i] = min(dp[j] + cost(j+1, i)) given that
    cost() is QF
long long solve1D(int n, long long cost(int, int)) {
  vector<long long> dp(n + 1), opt(n + 1);
  deque<pair<int, int>> dq;
  dq.push_back({0, 1});
  dp[0] = 0;

  for (int i = 1; i <= n; i++) {
    opt[i] = dq.front().first;
    dp[i] = dp[opt[i]] + cost(opt[i] + 1, i);
    if (i == n) break;

    dq[0].second++;
    if (dq.size() > 1 && dq[0].second == dq[1].second) dq
        .pop_front();
```

```
    int en = n;
    while (dq.size()) {
      int o = dq.back().first, st = dq.back().second;
      if (dp[o] + cost(o + 1, st) >= dp[i] + cost(i + 1,
           st))
        dq.pop_back();
      else {
        int lo = st, hi = en;
        while (lo < hi) {
          int mid = (lo + hi + 1) / 2;
          if (dp[o] + cost(o + 1, mid) < dp[i] + cost(i +
               1, mid))
            lo = mid;
          else
            hi = mid - 1;
        }
        if (lo < n) dq.push_back({i, lo + 1});
        break;
      }
      en = st - 1;
    }
    if (dq.empty()) dq.push_back({i, i + 1});
  }
  return dp[n];
}

/// Solves https://open.kattis.com/problems/
     coveredwalkway
const int N = 1e6 + 7;
long long x[N];
int c;
long long cost(int l, int r) { return (x[r] - x[l]) * (x[
     r] - x[l]) + c; }

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

  int n;
  cin >> n >> c;
  for (int i = 1; i <= n; i++) cin >> x[i];
  cout << solve1D(n, cost) << endl;
}
```

2.2 Convex Hull Trick

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

using LL = long long;
const int N = 3e5 + 9;
const int M = 1e9 + 7;

struct CHT {
  vector<LL> m, b;
  int ptr = 0;

  bool bad(int l1, int l2, int l3) {
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2]) <= 1.0
         * (b[l2] - b[l1]) * (m[l1] - m[l3]); //(slope
         dec+query min),(slope inc+query max)
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2]) > 1.0
         * (b[l2] - b[l1]) * (m[l1] - m[l3]); //(slope dec
         +query max), (slope inc+query min)
  }

  void add(LL _m, LL _b) {
    m.push_back(_m);
    b.push_back(_b);
    int s = m.size();
    while (s >= 3 && bad(s - 3, s - 2, s - 1)) {
      s--;
      m.erase(m.end() - 2);
      b.erase(b.end() - 2);
    }
  }

  LL f(int i, LL x) { return m[i] * x + b[i]; }

  //(slope dec+query min), (slope inc+query max) -> x
       increasing
  //(slope dec+query max), (slope inc+query min) -> x
       decreasing

  LL query(LL x) {
    if (ptr >= m.size()) ptr = m.size() - 1;
    while (ptr < m.size() - 1 && f(ptr + 1, x) < f(ptr, x
         )) ptr++;
    return f(ptr, x);
  }

  LL bs(int l, int r, LL x) {
    int mid = (l + r) / 2;
    if (mid + 1 < m.size() && f(mid + 1, x) < f(mid, x))
         return bs(mid + 1, r, x); // > for max
    if (mid - 1 >= 0 && f(mid - 1, x) < f(mid, x)) return
          bs(l, mid - 1, x); // > for max
    return f(mid, x);
  }
};
```

2.3 Divide and Conquer dp

```
const int K = 805, N = 4005;
LL dp[2][N], _cost[N][N];
// 1-indexed for convenience
LL cost(int l, int r) {
  return _cost[r][r] - _cost[l - 1][r] - _cost[r][l - 1]
       + _cost[l - 1][l - 1] >> 1;
}
void compute(int cnt, int l, int r, int optl, int optr) {
  if (l > r) return;
  int mid = l + r >> 1;
  LL best = INT_MAX;
  int opt = -1;
  for (int i = optl; i <= min(mid, optr); i++) {
    LL cur = dp[cnt ^ 1][i - 1] + cost(i, mid);
    if (cur < best) best = cur, opt = i;
  }
  dp[cnt][mid] = best;
  compute(cnt, l, mid - 1, optl, opt);
  compute(cnt, mid + 1, r, opt, optr);
}
LL dnc_dp(int k, int n) {
  fill(dp[0] + 1, dp[0] + n + 1, INT_MAX);
  for (int cnt = 1; cnt <= k; cnt++) {
    compute(cnt & 1, 1, n, 1, n);
  }
  return dp[k & 1][n];
}
```

2.4 Dynamic CHT

```
typedef long long LL;

const LL IS_QUERY = -(1LL << 62);

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 HullDynamic : public multiset <line> {
  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 1.0 * (x -> b - y -> b) * (z -> m - y -> m) >=
         1.0 * (y -> b - z -> b) * (y -> m - x -> m);
  }

  void insert_line (LL m, LL b) {
    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;
  }
};
```

2.5 FFT Online

```
void fftOnline(vector <LL> &a, vector <LL> b) {
  int n = a.size();
  auto call = [&](int l, int r, auto &call){
    if(l >= r) return;
    int mid = l + r >> 1;
    call(l, mid, call);

    vector <LL> _a(a.begin() + l, a.begin() + mid + 1);
    vector <LL> _b(b.begin(), b.begin() + (r - l + 1));
    auto c = fft :: anyMod(_a, _b);

    for(int i = mid + 1; i <= r; i++) {
      a[i] += c[i - l];
      a[i] -= (a[i] >= mod) * mod;
    }
    call(mid + 1, r, call);
  };
  call(0, n - 1, call);
}
```

2.6 Knuth optimization

```
const int N = 1005;
LL dp[N][N], a[N];
int opt[N][N];
LL cost(int i, int j) { return a[j + 1] - a[i]; }
LL knuth_optimization(int n) {
  for (int i = 0; i < n; i++) {
    dp[i][i] = 0;
    opt[i][i] = i;
  }
  for (int i = n - 2; i >= 0; i--) {
    for (int j = i + 1; j < n; j++) {
      LL mn = LLONG_MAX;
      LL c = cost(i, j);
      for (int k = opt[i][j - 1]; k <= min(j - 1, opt[i +
          1][j]); k++) {
        if (mn > dp[i][k] + dp[k + 1][j] + c) {
          mn = dp[i][k] + dp[k + 1][j] + c;
          opt[i][j] = k;
        }
      }
      dp[i][j] = mn;
    }
  }
  return dp[0][n - 1];
}
```

2.7 Li Chao Tree

```
struct line {
  LL m, c;
  line(LL m = 0, LL c = 0) : m(m), c(c) {}
};
LL calc(line L, LL x) { return 1LL * L.m * x + L.c; }
struct node {
  LL m, c;
  line L;
  node *lft, *rt;
  node(LL m = 0, LL c = 0, node *lft = NULL, node *rt =
      NULL)
      : L(line(m, c)), lft(lft), rt(rt) {}
};
struct LiChao {
  node *root;
  LiChao() { root = new node(); }
  void update(node *now, int L, int R, line newline) {
    int mid = L + (R - L) / 2;
    line lo = now->L, hi = newline;
    if (calc(lo, L) > calc(hi, L)) swap(lo, hi);
    if (calc(lo, R) <= calc(hi, R)) {
      now->L = hi;
      return;
    }
    if (calc(lo, mid) < calc(hi, mid)) {
      now->L = hi;
      if (now->rt == NULL) now->rt = new node();
      update(now->rt, mid + 1, R, lo);
    } else {
      now->L = lo;
      if (now->lft == NULL) now->lft = new node();
      update(now->lft, L, mid, hi);
    }
  }
  LL query(node *now, int L, int R, LL x) {
    if (now == NULL) return -inf;
    int mid = L + (R - L) / 2;
    if (x <= mid)
      return max(calc(now->L, x), query(now->lft, L, mid,
          x));
    else
      return max(calc(now->L, x), query(now->rt, mid + 1,
          R, x));
  }
};
```

3 Data Structure

3.1 Segment Tree

```
template <typename VT>
struct segtree {
  using DT = typename VT::DT;
  using LT = typename VT::LT;

  int L, R;
  vector <VT> tr;
  segtree(int n): L(0), R(n - 1), tr(n << 2) {}
  segtree(int l, int r): L(l), R(r), tr((r - l + 1) << 2)
      {}

  void propagate(int l, int r, int u) {
    if(l == r) return;
    VT :: apply(tr[u << 1], tr[u].lz, l, (l + r) >> 1);
    VT :: apply(tr[u << 1 | 1], tr[u].lz, (l + r + 2) >>
        1, r);
    tr[u].lz = VT :: None;
  }

  void build(int l, int r, vector <DT> &v, int u = 1 ) {
    if(l == r) {
      tr[u].val = v[l];
      return;
    }
    int m = (l + r) >> 1, lft = u << 1, ryt = u << 1 | 1;
    build(l, m, v, lft);
```

```
    build(m + 1, r, v, ryt);
    tr[u].val = VT :: merge(tr[lft].val, tr[ryt].val, l,
        r);
  }

  void update(int ql,int qr, LT up, int l, int r, int u =
      1) {
    if(ql > qr) return;
    if(ql == l and qr == r) {
      VT :: apply(tr[u], up, l, r);
      return;
    }
    propagate(l, r, u);
    int m = (l + r) >> 1, lft = u << 1, ryt = u << 1 | 1;
    update(ql, min(m, qr), up, l, m, lft);
    update(max(ql, m + 1), qr, up, m + 1, r, ryt);
    tr[u].val = VT :: merge(tr[lft].val, tr[ryt].val, l,
        r);
  }

  DT query(int ql, int qr, int l, int r, int u = 1) {
    if(ql > qr) return VT::I;
    if(l == ql and r == qr)
      return tr[u].val;
    propagate(l, r, u);
    int m = (l + r) >> 1, lft = u << 1, ryt = u << 1 | 1;
    DT ansl = query(ql, min(m, qr), l, m, lft);
    DT ansr = query(max(ql, m + 1), qr, m + 1, r, ryt);
    return tr[u].merge(ansl, ansr, l, r);
  }

  void build(vector <DT> &v) { build(L, R, v); }
  void update(int ql, int qr, LT U) { update(ql, qr, U, L
      , R); }
  DT query(int ql, int qr) { return query(ql, qr, L, R);
      }
};


struct add_sum {
  using DT = LL;
  using LT = LL;
  DT val;
  LT lz;

  static constexpr DT I = 0;
  static constexpr LT None = 0;

  add_sum(DT _val = I, LT _lz = None): val(_val), lz(_lz)
      {}

  static void apply(add_sum &u, const LT &up, int l, int
      r) {
    u.val += (r - l + 1) * up;
    u.lz += up;
  }

  static DT merge(const DT &a, const DT &b, int l, int r)
      {
    return a + b;
  }
};
```

3.2 Spare Table

```
template <typename T> struct sparse_table {
  vector <vector<T>> tbl;
  function < T(T, T) > f;
  T id;

  sparse_table(const vector <T> &v, function <T(T, T)> _f
      , T _id) : f(_f), id(_id) {
    int n = (int) v.size(), b = __lg(n);
    tbl.assign(b + 1, v);
    for(int k = 1; k <= b; k++) {
      for(int i = 0; i + (1 << k) <= n; i++) {
        tbl[k][i] = f(tbl[k - 1][i], tbl[k - 1][i + (1 <<
            (k - 1))]);
      }
    }
  }
  T query(int l, int r) {
    if(l > r) return id;
    int pow = __lg(r - l + 1);
    return f(tbl[pow][l], tbl[pow][r - (1 << pow) + 1]);
  }
};
```

3.3 Persistent Segment Tree

```
struct Node {
  int l = 0, r = 0, val = 0;
} tr[20 * N];
int ptr = 0;
int build(int st, int en) {
  int u = ++ptr;
  if (st == en) return u;
  int mid = (st + en) / 2;
  auto& [l, r, val] = tr[u];
  l = build(st, mid);
  r = build(mid + 1, en);
  val = tr[l].val + tr[r].val;
  return u;
}

int update(int pre, int st, int en, int idx, int v) {
  int u = ++ptr;
  tr[u] = tr[pre];
  if (st == en) {
    tr[u].val += v;
    return u;
  }
  int mid = (st + en) / 2;
  auto& [l, r, val] = tr[u];
  if (idx <= mid) {
    r = tr[pre].r;
    l = update(tr[pre].l, st, mid, idx, v);
  } else {
    l = tr[pre].l;
    r = update(tr[pre].r, mid + 1, en, idx, v);
  }
  tr[u].val = tr[l].val + tr[r].val;
  return u;
}
// finding the kth elelment in a range
int query(int left, int right, int st, int en, int k) {
  if (st == en) return st;
  int cnt = tr[tr[right].l].val - tr[tr[left].l].val;
  int mid = (st + en) / 2;
  if (cnt >= k) return query(tr[left].l, tr[right].l, st,
      mid, k);
  else return query(tr[left].r, tr[right].r, mid + 1, en,
      k - cnt);
}
int V[N], root[N], a[N];
int main() {
  map<int, int> mp; int n, q;
  cin >> n >> q;
  for (int i = 1; i <= n; i++) cin >> a[i], mp[a[i]];
  int c = 0;
  for (auto x : mp) mp[x.first] = ++c, V[c] = x.first;
  root[0] = build(1, n);
  for (int i = 1; i <= n; i++) {
    root[i] = update(root[i - 1], 1, n, mp[a[i]], 1);
  }
  while (q--) {
    int l, r, k; cin >> l >> r >> k; l++, k++;
    cout << V[query(root[l - 1], root[r], 1, n, k)] << '\
        n';
  }
  return 0;
}
```

3.4 SegTree Beats

```
const int N = 2e5 + 5;
LL mx[4 * N], mn[4 * N], smx[4 * N], smn[4 * N], sum[4 *
    N], add[4 * N];
int mxcnt[4 * N], mncnt[4 * N];

int L, R;

void applyMax(int u, LL x) {
  sum[u] += mncnt[u] * (x - mn[u]);
  if (mx[u] == mn[u]) mx[u] = x;
  if (smx[u] == mn[u]) smx[u] = x;
  mn[u] = x;
}
void applyMin(int u, LL x) {
  sum[u] -= mxcnt[u] * (mx[u] - x);
  if (mn[u] == mx[u]) mn[u] = x;
  if (smn[u] == mx[u]) smn[u] = x;
  mx[u] = x;
}
void applyAdd(int u, LL x, int tl, int tr) {
  sum[u] += (tr - tl + 1) * x;
  add[u] += x;
  mx[u] += x, mn[u] += x;
  if (smx[u] != -INF) smx[u] += x;
  if (smn[u] != INF) smn[u] += x;
}
void push(int u, int tl, int tr) {
  int lft = u << 1, ryt = lft | 1, mid = tl + tr >> 1;
  if (add[u] != 0) {
    applyAdd(lft, add[u], tl, mid);
    applyAdd(ryt, add[u], mid + 1, tr);
    add[u] = 0;
  }
  if (mx[u] < mx[lft]) applyMin(lft, mx[u]);
  if (mx[u] < mx[ryt]) applyMin(ryt, mx[u]);

  if (mn[u] > mn[lft]) applyMax(lft, mn[u]);
  if (mn[u] > mn[ryt]) applyMax(ryt, mn[u]);
}
void merge(int u) {
  int lft = u << 1, ryt = lft | 1;
  sum[u] = sum[lft] + sum[ryt];

  mx[u] = max(mx[lft], mx[ryt]);
  smx[u] = max(smx[lft], smx[ryt]);
  if (mx[lft] != mx[ryt]) smx[u] = max(smx[u], min(mx[lft
      ], mx[ryt]));
  mxcnt[u] = (mx[u] == mx[lft]) * mxcnt[lft] + (mx[u] ==
      mx[ryt]) * mxcnt[ryt];
  mn[u] = min(mn[lft], mn[ryt]);
  smn[u] = min(smn[lft], smn[ryt]);
  if (mn[lft] != mn[ryt]) smn[u] = min(smn[u], max(mn[lft
      ], mn[ryt]));
  mncnt[u] = (mn[u] == mn[lft]) * mncnt[lft] + (mn[u] ==
      mn[ryt]) * mncnt[ryt];
}
void minimize(int l, int r, LL x, int tl = L, int tr = R,
     int u = 1) {
  if (l > tr or tl > r or mx[u] <= x) return;
  if (l <= tl and tr <= r and smx[u] < x) {
    applyMin(u, x);
    return;
  }
  push(u, tl, tr);
  int mid = tl + tr >> 1, lft = u << 1, ryt = lft | 1;
  minimize(l, r, x, tl, mid, lft);
  minimize(l, r, x, mid + 1, tr, ryt);
  merge(u);
}
void maximize(int l, int r, LL x, int tl = L, int tr = R,
     int u = 1) {
  if (l > tr or tl > r or mn[u] >= x) return;
  if (l <= tl and tr <= r and smn[u] > x) {
    applyMax(u, x);
    return;
  }
  push(u, tl, tr);
  int mid = tl + tr >> 1, lft = u << 1, ryt = lft | 1;
  maximize(l, r, x, tl, mid, lft);
  maximize(l, r, x, mid + 1, tr, ryt);
  merge(u);
}
void increase(int l, int r, LL x, int tl = L, int tr = R,
     int u = 1) {
  if (l > tr or tl > r) return;
  if (l <= tl and tr <= r) {
    applyAdd(u, x, tl, tr);
    return;
  }
  push(u, tl, tr);
  int mid = tl + tr >> 1, lft = u << 1, ryt = lft | 1;
  increase(l, r, x, tl, mid, lft);
  increase(l, r, x, mid + 1, tr, ryt);
  merge(u);
}
LL getSum(int l, int r, int tl = L, int tr = R, int u =
    1) {
  if (l > tr or tl > r) return 0;
  if (l <= tl and tr <= r) return sum[u];
  push(u, tl, tr);
  int mid = tl + tr >> 1, lft = u << 1, ryt = lft | 1;
  return getSum(l, r, tl, mid, lft) + getSum(l, r, mid +
      1, tr, ryt);
}
void build(LL a[], int tl = L, int tr = R, int u = 1) {
  if (tl == tr) {
    sum[u] = mn[u] = mx[u] = a[tl];
    mxcnt[u] = mncnt[u] = 1;
    smx[u] = -INF;
    smn[u] = INF;
    return;
  }
  int mid = tl + tr >> 1, lft = u << 1, ryt = lft | 1;
  build(a, tl, mid, lft);
  build(a, mid + 1, tr, ryt);
  merge(u);
}
void init(LL a[], int _L, int _R) {
  L = _L, R = _R;
  build(a);
}
```

3.5 HashTable

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

const int RANDOM = chrono::high_resolution_clock::now().
    time_since_epoch().count();
unsigned hash_f(unsigned x) {
  x = ((x >> 16) ^ x) * 0x45d9f3b;
  x = ((x >> 16) ^ x) * 0x45d9f3b;
  return x = (x >> 16) ^ x;
}

unsigned hash_combine(unsigned a, unsigned b) { return a
    * 31 + b; }
struct chash {
  int operator()(int x) const { return hash_f(x); }
};
typedef gp_hash_table<int, int, chash> gp;
gp table;
```

3.6 DSU With Rollbacks

```
struct Rollback_DSU {
  int n;
  vector<int> par, sz;
  vector<pair<int, int>> op;
  Rollback_DSU(int n) : par(n), sz(n, 1) {
```

```
    iota(par.begin(), par.end(), 0);
    op.reserve(n);
  }
  int Anc(int node) {
    for (; node != par[node]; node = par[node])
      ;  // no path compression
    return node;
  }
  void Unite(int x, int y) {
    if (sz[x = Anc(x)] < sz[y = Anc(y)]) swap(x, y);
    op.emplace_back(x, y);
    par[y] = x;
    sz[x] += sz[y];
  }
  void Undo(int t) {
    for (; op.size() > t; op.pop_back()) {
      par[op.back().second] = op.back().second;
      sz[op.back().first] -= sz[op.back().second];
    }
  }
};
```

3.7 Binary Trie

```
const int N = 1e7 + 5, b = 30;
int tc = 1;
struct node {
  int vis = 0;
  int to[2] = {0, 0};
  int val[2] = {0, 0};
  void update() {
    to[0] = to[1] = 0;
    val[0] = val[1] = 0;
    vis = tc;
  }
} T[N + 2];
node *root = T;
int ptr = 0;
node *nxt(node *cur, int x) {
  if (cur->to[x] == 0) cur->to[x] = ++ptr;
  assert(ptr < N);
  int idx = cur->to[x];
  if (T[idx].vis < tc) T[idx].update();
  return T + idx;
}
int query(int j, int aj) {
  int ans = 0, jaj = j ^ aj;
  node *cur = root;
  for (int k = b - 1; ~k; k--) {
    maximize(ans, nxt(cur, (jaj >> k & 1) ^ 1)->val[1 ^ (
        aj >> k & 1)]);
    cur = nxt(cur, (jaj >> k & 1));
  }
  return ans;
}
void insert(int j, int aj, int val) {
  int jaj = j ^ aj;
  node *cur = root;
  for (int k = b - 1; ~k; k--) {
    cur = nxt(cur, (jaj >> k & 1));
    maximize(cur->val[j >> k & 1], val);
  }
}
void clear() {
  tc++;
  ptr = 0;
  root->update();
}
```

3.8 BIT-2D

```
const int N = 1008;
int bit[N][N], n, m;
int a[N][N], q;
void update(int x, int y, int val) {
  for (; x < N; x += -x & x)
    for (int j = y; j < N; j += -j & j) bit[x][j] += val;
}
int get(int x, int y) {
  int ans = 0;
  for (; x; x -= x & -x)
    for (int j = y; j; j -= j & -j) ans += bit[x][j];
  return ans;
}
int get(int x1, int y1, int x2, int y2) {
  return get(x2, y2) - get(x1 - 1, y2) - get(x2, y1 - 1)
      + get(x1 - 1, y1 - 1);
}
```

3.9 Divide And Conquer Query Offline

```
namespace up {
int l[N], r[N], u[N], v[N], tm;
void push(int _l, int _r, int _u, int _v) {
  l[tm] = _l, r[tm] = _r, u[tm] = _u, v[tm] = _v;
  tm++;
}
}  // namespace up
namespace que {
int node[N], tm;
LL ans[N];
void push(int _node) { node[++tm] = _node; }
}  // namespace que
namespace edge_set {
void push(int i) { dsu ::merge(up ::u[i], up ::v[i]); }
void pop(int t) { dsu ::rollback(t); }
int time() { return dsu ::op.size(); }
LL query(int u) { return a[dsu ::root(u)]; }
}  // namespace edge_set
namespace dncq {
vector<int> tree[4 * N];
void update(int idx, int l = 0, int r = que ::tm, int
    node = 1) {
  int ul = up ::l[idx], ur = up ::r[idx];
  if (r < ul or ur < l) return;
  if (ul <= l and r <= ur) {
    tree[node].push_back(idx);
    return;
  }
  int m = l + r >> 1;
  update(idx, l, m, node << 1);
  update(idx, m + 1, r, node << 1 | 1);
}
void dfs(int l = 0, int r = que ::tm, int node = 1) {
  int cur = edge_set ::time();
  for (int e : tree[node]) edge_set ::push(e);
  if (l == r) {
    que ::ans[l] = edge_set ::query(que ::node[l]);
  } else {
    int m = l + r >> 1;
    dfs(l, m, node << 1);
    dfs(m + 1, r, node << 1 | 1);
  }
  edge_set ::pop(cur);
}
}  // namespace dncq
void push_updates() {
  for (int i = 0; i < up ::tm; i++) dncq ::update(i);
}
```

3.10 MO with Update

```
const int N = 1e5 + 5, sz = 2700, bs = 25;
int arr[N], freq[2 * N], cnt[2 * N], id[N], ans[N];
struct query {
    int l, r, t, L, R;
    query(int l = 1, int r = 0, int t = 1, int id = -1)
            : l(l), r(r), t(t), L(l / sz), R(r / sz) {}
    bool operator<(const query &rhs) const {
        return (L < rhs.L) or (L == rhs.L and R < rhs.R)
            or
                  (L == rhs.L and R == rhs.R and t <
                    rhs.t);
    }
```

```
} Q[N];
struct update {
    int idx, val, last;
} Up[N];
int qi = 0, ui = 0;
int l = 1, r = 0, t = 0;

void add(int idx) {
    --cnt[freq[arr[idx]]];
    freq[arr[idx]]++;
    cnt[freq[arr[idx]]]++;
}
void remove(int idx) {
    --cnt[freq[arr[idx]]];
    freq[arr[idx]]--;
    cnt[freq[arr[idx]]]++;
}
void apply(int t) {
    const bool f = l <= Up[t].idx and Up[t].idx <= r;
    if (f) remove(Up[t].idx);
    arr[Up[t].idx] = Up[t].val;
    if (f) add(Up[t].idx);
}
void undo(int t) {
    const bool f = l <= Up[t].idx and Up[t].idx <= r;
    if (f) remove(Up[t].idx);
    arr[Up[t].idx] = Up[t].last;
    if (f) add(Up[t].idx);
}
int mex() {
    for (int i = 1; i <= N; i++)
        if (!cnt[i]) return i;
    assert(0);
}
int main() {
    sort(id + 1, id + qi + 1, [&](int x, int y) { return
         Q[x] < Q[y]; });
    for (int i = 1; i <= qi; i++) {
        int x = id[i];
        while (Q[x].t > t) apply(++t);
        while (Q[x].t < t) undo(t--);
        while (Q[x].l < l) add(--l);
        while (Q[x].r > r) add(++r);
        while (Q[x].l > l) remove(l++);
        while (Q[x].r < r) remove(r--);
        ans[x] = mex();
    }
}
```

3.11 SparseTable (Rectangle Query)

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

const int MAXN = 505;
const int LOGN = 9;

// O(n^2 (logn)^2
// Supports Rectangular Query
int A[MAXN][MAXN];
int M[MAXN][MAXN][LOGN][LOGN];

void Build2DSparse(int N) {
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            M[i][j][0][0] = A[i][j];
        }
        for (int q = 1; (1 << q) <= N; q++) {
            int add = 1 << (q - 1);
            for (int j = 1; j + add <= N; j++) {
                M[i][j][0][q] = max(M[i][j][0][q - 1], M[i
                    ][j + add][0][q - 1]);
            }
        }
    }

    for (int p = 1; (1 << p) <= N; p++) {
        int add = 1 << (p - 1);
        for (int i = 1; i + add <= N; i++) {
            for (int q = 0; (1 << q) <= N; q++) {
                for (int j = 1; j <= N; j++) {
                    M[i][j][p][q] = max(M[i][j][p - 1][q],
                        M[i + add][j][p - 1][q]);
                }
            }
        }
    }
}

// returns max of all A[i][j], where x1<=i<=x2 and y1<=j
   <=y2
int Query(int x1, int y1, int x2, int y2) {
    int kX = log2(x2 - x1 + 1);
    int kY = log2(y2 - y1 + 1);
    int addX = 1 << kX;
    int addY = 1 << kY;

    int ret1 = max(M[x1][y1][kX][kY], M[x1][y2 - addY +
        1][kX][kY]);
    int ret2 = max(M[x2 - addX + 1][y1][kX][kY],
                    M[x2 - addX + 1][y2 - addY
                     + 1][kX][kY]);
    return max(ret1, ret2);
}
```

4 Geometry

4.1 Point

```
typedef double Tf;
typedef Tf Ti;    /// use long long for exactness
const Tf PI = acos(-1), EPS = 1e-9;
int dcmp(Tf x) { return abs(x) < EPS ? 0 : (x<0 ? -1 : 1)
    ;}

struct Pt {
  Ti x, y;
  Pt(Ti x = 0, Ti y = 0) : x(x), y(y) {}
  Pt operator + (const Pt& u) const { return Pt(x + u.x,
      y + u.y); }
  Pt operator - (const Pt& u) const { return Pt(x - u.x,
      y - u.y); }
  Pt operator * (const long long u) const { return Pt(x *
       u, y * u); }
  Pt operator * (const Tf u) const { return Pt(x * u, y *
       u); }
  Pt operator / (const Tf u) const { return Pt(x / u, y /
       u); }

  bool operator == (const Pt& u) const { return dcmp(x -
      u.x) == 0 && dcmp(y - u.y) == 0; }
  bool operator != (const Pt& u) const { return !(*this
      == u); }
  bool operator < (const Pt& u) const { return dcmp(x - u
      .x) < 0 || (dcmp(x - u.x) == 0 && dcmp(y - u.y) <
      0); }
  friend istream &operator >> (istream &is, Pt &p) {
      return is >> p.x >> p.y; }
  friend ostream &operator << (ostream &os, const Pt &p)
      { return os << p.x << " " << p.y; }
};
using vec_p = vector<Pt>;

Ti dot(Pt a, Pt b) { return a.x * b.x + a.y * b.y; }
Ti crs(Pt a, Pt b) { return a.x * b.y - a.y * b.x; }
Tf len(Pt a) { return sqrt(dot(a, a)); }
Ti sqlen(Pt a) { return dot(a, a); }
Tf dis(Pt a, Pt b) {return len(a-b);}
Tf angle(Pt u) { return atan2(u.y, u.x); }
// returns angle between oa, ob in (-PI, PI]
Tf angleBetween(Pt a, Pt b) {
  double ans = angle(b) - angle(a);
```

```
  return ans <= -PI ? ans + 2*PI : (ans > PI ? ans - 2*PI
       : ans);
}
// Rotate a ccw by rad radians
Pt rotate(Pt a, Tf rad) {
  return Pt(a.x * cos(rad) - a.y * sin(rad), a.x * sin(
       rad) + a.y * cos(rad));
}
// rotate a ccw by angle th with cos(th) = co && sin(th)
    = si
Pt rotatePrecise(Pt a, Tf co, Tf si) {
  return Pt(a.x * co - a.y * si, a.y * co + a.x * si);
}
Pt rotate90(Pt a) { return Pt(-a.y, a.x); }
// scales vector a by s such that len of a becomes s
Pt scale(Pt a, Tf s) {
  return a / len(a) * s;
}
// returns an unit vector perpendicular to vector a
Pt normal(Pt a) {
  Tf l = len(a);
  return Pt(-a.y / l, a.x / l);
}
int ornt(Pt a, Pt b, Pt c) {
  return dcmp(crs(b - a, c - a));
}
bool half(Pt p){    // returns true for pt above x axis
    or on negative x axis
  return p.y > 0 || (p.y == 0 && p.x < 0);
}
bool polarComp(Pt p, Pt q){ //to be used in sort()
    function
  return make_tuple(half(p), 0) < make_tuple(half(q), crs
      (p, q));
}
struct Seg {
  Pt a, b;
  Seg(Pt aa, Pt bb) : a(aa), b(bb) {}
};
typedef Seg Line;
struct Crc {
    Pt o;
    Tf r;
    Crc(Pt o = Pt(0, 0), Tf r = 0) : o(o), r(r) {}
    // returns true if pt p is in || on the crc
    bool contains(Pt p) {
      return dcmp(sqlen(p - o) - r * r) <= 0;
    }
    // returns a pt on the crc rad radians away from +X
        CCW
    Pt pt(Tf rad) {
      return Pt(o.x + cos(rad) * r, o.y + sin(rad) * r);
    }
    // area of a circular sector with central angle rad
    Tf area(Tf rad = PI + PI) { return rad * r * r / 2; }
    // area of the circular sector cut by a chord with
        central angle alpha
    Tf sector(Tf alpha) { return r * r * 0.5 * (alpha -
        sin(alpha)); }
};
```

4.2 Linear

```
// returns true if pt p is on segs s
bool onSeg(Pt p, Seg s) {
  return dcmp(crs(s.a - p, s.b - p)) == 0 && dcmp(dot(s.a
       - p, s.b - p)) <= 0;
}
// returns true if segs p && q touch or intersect
bool segssIntersect(Seg p, Seg q) {
  if(onSeg(p.a, q) || onSeg(p.b, q)) return true;
  if(onSeg(q.a, p) || onSeg(q.b, p)) return true;
  Ti c1 = crs(p.b - p.a, q.a - p.a);
  Ti c2 = crs(p.b - p.a, q.b - p.a);
  Ti c3 = crs(q.b - q.a, p.a - q.a);
  Ti c4 = crs(q.b - q.a, p.b - q.a);
  return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) <
       0;
}
bool linesParallel(Line p, Line q) {
  return dcmp(crs(p.b - p.a, q.b - q.a)) == 0;
}
// lines are represented as a ray from a pt: (pt, vector)
// returns false if two lines (p, v) && (q, w) are
    parallel or collinear
// true otherwise, intersection pt is stored at o via
    reference
bool lineLineIntscn(Pt p, Pt v, Pt q, Pt w, Pt& o) {
  if(dcmp(crs(v, w)) == 0) return false;
  Pt u = p - q;
  o = p + v * (crs(w,u)/crs(v,w));
  return true;
}
// returns false if two lines p && q are parallel or
    collinear
// true otherwise, intersection pt is stored at o via
    reference
bool lineLineIntscn(Line p, Line q, Pt& o) {
  return lineLineIntscn(p.a, p.b - p.a, q.a, q.b - q.a, o
      );
}
// returns the dis from pt a to line l
Tf disPtLine(Pt p, Line l) {
  return abs(crs(l.b - l.a, p - l.a) / len(l.b - l.a));
}
// returns the shortest dis from pt a to segs s
Tf disPtSeg(Pt p, Seg s) {
  if(s.a == s.b) return len(p - s.a);
  Pt v1 = s.b - s.a, v2 = p - s.a, v3 = p - s.b;
  if(dcmp(dot(v1, v2)) < 0)  return len(v2);
  else if(dcmp(dot(v1, v3)) > 0) return len(v3);
  else return abs(crs(v1, v2) / len(v1));
}
// returns the shortest dis from segs p to segs q
Tf disSegSeg(Seg p, Seg q) {
  if(segssIntersect(p, q)) return 0;
  Tf ans = disPtSeg(p.a, q);
  ans = min(ans, disPtSeg(p.b, q));
  ans = min(ans, disPtSeg(q.a, p));
  ans = min(ans, disPtSeg(q.b, p));
  return ans;
}
// returns the projection of pt p on line l
Pt projectPtLine(Pt p, Line l) {
  Pt v = l.b - l.a;
  return l.a + v * ((Tf) dot(v, p - l.a) / dot(v, v));
}
```

4.3 Polygon

```
using Poly = vector<Pt>;
Tf signedPolyArea(Poly p) {
  Tf ret = 0;
  for(int i = 0; i < (int) p.size() - 1; i++)
    ret += crs(p[i]-p[0], p[i+1]-p[0]);
  return ret / 2;
}
// given a polygon p of n vertices, generates the convex
    hull in ch
// in CCW && returns the number of vertices in the convex
     hull
int convexHull(Poly p, Poly &ch) {
  sort(p.begin(), p.end());
  int n = p.size();
  ch.resize(n + n);
  int m = 0;   // preparing lower hull
  for(int i = 0; i < n; i++) {
    while(m > 1 && dcmp(crs(ch[m - 1] - ch[m - 2], p[i] -
         ch[m - 1])) <= 0) m--;
    ch[m++] = p[i];
  }
  int k = m;   // preparing upper hull
```

```cpp
  for(int i = n - 2; i >= 0; i--) {
    while(m > k && dcmp(crs(ch[m - 1] - ch[m - 2], p[i] -
          ch[m - 2])) <= 0) m--;
    ch[m++] = p[i];
  }
  if(n > 1) m--;
  ch.resize(m);
  return m;
}
// for a pt o and polygon p returns:
//   -1 if o is strictly inside p
//  0 if o is on a segs of p
//  1 if o is strictly outside p
// computes via winding numbers
int ptInPoly(Pt o, Poly p) {
  using Linear::onSeg;
  int wn = 0, n = p.size();
  for(int i = 0; i < n; i++) {
    int j = (i + 1) % n;
    if(onSeg(o, Seg(p[i], p[j])) || o == p[i]) return 0;
    int k = dcmp(crs(p[j] - p[i], o - p[i]));
    int d1 = dcmp(p[i].y - o.y);
    int d2 = dcmp(p[j].y - o.y);
    if(k > 0 && d1 <= 0 && d2 > 0) wn++;
    if(k < 0 && d2 <= 0 && d1 > 0) wn--;
  }
  return wn ? -1 : 1;
}
// returns the longest line segs of l that is inside or
    on the
// simply polygon p. O(n lg n). TESTED: TIMUS 1955
Tf longestSegInPoly(Line l, const Poly &p) {
  using Linear::lineLineIntscn;
  int n = p.size();
  vector<pair<Tf, int>> ev;
  for(int i=0; i<n; ++i) {
    Pt a = p[i], b = p[(i + 1) % n], z = p[(i - 1 + n) %
        n];
    int ora = ornt(l.a, l.b, a), orb = ornt(l.a, l.b, b),
          orz = ornt(l.a, l.b, z);
    if(!ora) {
      Tf d = dot(a - l.a, l.b - l.a);
      if(orz && orb) {
        if(orz != orb) ev.emplace_back(d, 0);
      }
      else if(orz) ev.emplace_back(d, orz);
      else if(orb) ev.emplace_back(d, orb);
    }
    else if(ora == -orb) {
      Pt ins;
      lineLineIntscn(l, Line(a, b), ins);
      ev.emplace_back(dot(ins - l.a, l.b - l.a), 0);
    }
  }
  sort(ev.begin(), ev.end());
  Tf ret = 0, cur = 0, pre = 0;
  bool active = false;
  int sign = 0;
  for(auto &qq : ev) {
    int tp = qq.second;
    Tf d = qq.first;
    if(sign) {
      cur += d - pre;
      ret = max(ret, cur);
      if(tp != sign) active = !active;
      sign = 0;
    }
    else {
      if(active) cur += d - pre, ret = max(ret, cur);
      if(tp == 0) active = !active;
      else sign = tp;
    }
    pre = d;
    if(!active) cur = 0;
  }
  ret /= len(l.b - l.a);
  return ret;
}
```

4.4 Convex

```cpp
/// minkowski sum of two polygons in O(n)
Poly minkowskiSum(Poly A, Poly B) {
  int n = A.size(), m = B.size();
  rotate(A.begin(), min_element(A.begin(), A.end()), A.
      end());
  rotate(B.begin(), min_element(B.begin(), B.end()), B.
      end());
  A.push_back(A[0]);
  B.push_back(B[0]);
  for (int i = 0; i < n; i++) A[i] = A[i + 1] - A[i];
  for (int i = 0; i < m; i++) B[i] = B[i + 1] - B[i];
  Poly C(n + m + 1);
  C[0] = A.back() + B.back();
  merge(A.begin(), A.end() - 1, B.begin(), B.end() - 1, C
      .begin() + 1, polarComp(Pt(0, 0), Pt(0, -1)));
  for (int i = 1; i < C.size(); i++) C[i] = C[i] + C[i -
      1];
  C.pop_back();
  return C;
}
void rotatingCalipersGetRectangle(Pt* p, int n, Tf& area,
      Tf& perimeter) {
  using Linear::disPtLine;
  p[n] = p[0];
  int l = 1, r = 1, j = 1;
  area = perimeter = 1e100;
  for(int i = 0; i < n; i++) {
    Pt v = (p[i + 1] - p[i]) / len(p[i + 1] - p[i]);
    while(dcmp(dot(v, p[r % n] - p[i]) - dot(v, p[(r + 1)
          % n] - p[i])) < 0) r++;
    while(j < r || dcmp(crs(v, p[j % n] - p[i]) - crs(v,
          p[(j + 1) % n] - p[i])) < 0) j++;
    while(l < j || dcmp(dot(v, p[l % n] - p[i]) - dot(v,
          p[(l + 1) % n] - p[i])) > 0) l++;
    Tf w = dot(v, p[r % n] - p[i]) - dot(v, p[l % n] - p[
          i]);
    Tf h = disPtLine(p[j % n], Line(p[i], p[i + 1]));
    area = min(area, w * h);
    perimeter = min(perimeter, 2 * w + 2 * h);
  }
}
// returns the left side of polygon u after cutting it by
      ray a->b
Poly cutPoly(Poly u, Pt a, Pt b) {
  using Linear::lineLineIntscn, Linear::onSeg;
  Poly ret;
  int n = u.size();
  for(int i = 0; i < n; i++) {
    Pt c = u[i], d = u[(i + 1) % n];
    if(dcmp(crs(b-a, c-a)) >= 0) ret.push_back(c);
    if(dcmp(crs(b-a, d-c)) != 0) {
      Pt t;
      lineLineIntscn(a, b - a, c, d - c, t);
      if(onSeg(t, Seg(c, d))) ret.push_back(t);
    }
  }
  return ret;
}
// returns true if pt p is in or on tri abc
bool ptInTri(Pt a, Pt b, Pt c, Pt p) {
  return dcmp(crs(b - a, p - a)) >= 0 && dcmp(crs(c - b,
      p - b)) >= 0 && dcmp(crs(a - c, p - c)) >= 0;
}
// pt must be in ccw order with no three collinear pts
// returns inside = -1, on = 0, outside = 1
int ptInConvexPoly(const Poly &pt, Pt p) {
  int n = pt.size();
  assert(n >= 3);
  int lo = 1, hi = n - 1;
  while(hi - lo > 1) {
```

```
    int mid = (lo + hi) / 2;
    if(dcmp(crs(pt[mid] - pt[0], p - pt[0])) > 0) lo =
        mid;
    else hi = mid;
  }
  bool in = ptInTri(pt[0], pt[lo], pt[hi], p);
  if(!in) return 1;
  if(dcmp(crs(pt[lo] - pt[lo - 1], p - pt[lo - 1])) == 0)
      return 0;
  if(dcmp(crs(pt[hi] - pt[lo], p - pt[lo])) == 0) return
      0;
  if(dcmp(crs(pt[hi] - pt[(hi + 1) % n], p - pt[(hi + 1)
      % n])) == 0) return 0;
  return -1;
}
// Extreme Pt for a direction is the farthest pt in that
    direction
// poly is a convex polygon, sorted in CCW, doesn't
    contain redundant pts
// u is the direction for extremeness
int extremePt(const Poly &poly, Pt u = Pt(0, 1)) {
  int n = (int) poly.size();
  int a = 0, b = n;
  while(b - a > 1) {
    int c = (a + b) / 2;
    if(dcmp(dot(poly[c] - poly[(c + 1) % n], u)) >= 0 &&
        dcmp(dot(poly[c] - poly[(c - 1 + n) % n], u)) >=
        0) {
      return c;
    }
    bool a_up = dcmp(dot(poly[(a + 1) % n] - poly[a], u))
        >= 0;
    bool c_up = dcmp(dot(poly[(c + 1) % n] - poly[c], u))
        >= 0;
    bool a_above_c = dcmp(dot(poly[a] - poly[c], u)) > 0;
    if(a_up && !c_up) b = c;
    else if(!a_up && c_up) a = c;
    else if(a_up && c_up) {
      if(a_above_c) b = c;
      else a = c;
    }
    else {
      if(!a_above_c) b = c;
      else a = c;
    }
  }
  if(dcmp(dot(poly[a] - poly[(a + 1) % n], u)) > 0 &&
      dcmp(dot(poly[a] - poly[(a - 1 + n) % n], u)) > 0)
    return a;
  return b % n;
}
// For a convex polygon p and a line l, returns a list of
    segss
// of p that are touch or intersect line l.
// the i'th segs is considered (p[i], p[(i + 1) modulo |p
    |])
// #1 If a segs is collinear with the line, only that is
    returned
// #2 Else if l goes through i'th pt, the i'th segs is
    added
// If there are 2 or more such collinear segss for #1,
// any of them (only one, not all) should be returned (
    not tested)
// Complexity: O(lg |p|)
vector<int> lineConvexPolyIntscn(const Poly &p, Line l) {
  assert((int) p.size() >= 3);
  assert(l.a != l.b);

  int n = p.size();
  vector<int> ret;

  Pt v = l.b - l.a;
  int lf = extremePt(p, rotate90(v));
  int rt = extremePt(p, rotate90(v) * Ti(-1));
  int olf = ornt(l.a, l.b, p[lf]);
  int ort = ornt(l.a, l.b, p[rt]);

  if(!olf || !ort) {
    int idx = (!olf ? lf : rt);
    if(ornt(l.a, l.b, p[(idx - 1 + n) % n]) == 0)
      ret.push_back((idx - 1 + n) % n);
    else ret.push_back(idx);
    return ret;
  }
  if(olf == ort) return ret;
  for(int i=0; i<2; ++i) {
    int lo = i ? rt : lf;
    int hi = i ? lf : rt;
    int olo = i ? ort : olf;

    while(true) {
      int gap = (hi - lo + n) % n;
      if(gap < 2) break;
      int mid = (lo + gap / 2) % n;
      int omid = ornt(l.a, l.b, p[mid]);
      if(!omid) {
        lo = mid;
        break;
      }
      if(omid == olo) lo = mid;
      else hi = mid;
    }
    ret.push_back(lo);
  }
  return ret;
}
// Calculate [ACW, CW] tangent pair from an external pt
constexpr int CW = -1, ACW = 1;
bool isGood(Pt u, Pt v, Pt Q, int dir) { return ornt(Q, u
    , v) != -dir; }
Pt better(Pt u, Pt v, Pt Q, int dir) { return ornt(Q, u,
    v) == dir ? u : v; }
Pt ptPolyTng(const Poly &pt, Pt Q, int dir, int lo, int
    hi) {
  while(hi - lo > 1) {
    int mid = (lo + hi) / 2;
    bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);
    bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);
    if(pvs && nxt) return pt[mid];
    if(!(pvs || nxt)) {
      Pt p1 = ptPolyTng(pt, Q, dir, mid + 1, hi);
      Pt p2 = ptPolyTng(pt, Q, dir, lo, mid - 1);
      return better(p1, p2, Q, dir);
    }
    if(!pvs) {
      if(ornt(Q, pt[mid], pt[lo]) == dir)     hi = mid -
          1;
      else if(better(pt[lo], pt[hi], Q, dir) == pt[lo])
          hi = mid - 1;
      else lo = mid + 1;
    }
    if(!nxt) {
      if(ornt(Q, pt[mid], pt[lo]) == dir)     lo = mid +
          1;
      else if(better(pt[lo], pt[hi], Q, dir) == pt[lo])
          hi = mid - 1;
      else lo = mid + 1;
    }
  }

  Pt ret = pt[lo];
  for(int i = lo + 1; i <= hi; i++) ret = better(ret, pt[
      i], Q, dir);
  return ret;
}
// [ACW, CW] Tng
pair<Pt, Pt> ptPolyTngs(const Poly &pt, Pt Q) {
  int n = pt.size();
  Pt acw_tan = ptPolyTng(pt, Q, ACW, 0, n - 1);
  Pt cw_tan = ptPolyTng(pt, Q, CW, 0, n - 1);
```

```
  return make_pair(acw_tan, cw_tan);
}
```

4.5 Circular

```
// Extremely inaccurate for finding near touches
// compute intersection of line l with crc c
// The intersections are given in order of the ray (l.a,
    l.b)
vec_p crcLineIntscn(Crc c, Line l) {
  static_assert(is_same<Tf, Ti>::value);
  vec_p ret;
  Pt b = l.b - l.a, a = l.a - c.o;
  Tf A = dot(b, b), B = dot(a, b);
  Tf C = dot(a, a) - c.r * c.r, D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(l.a + b * (-B - sqrt(D + EPS)) / A);
  if (D > EPS)
    ret.push_back(l.a + b * (-B + sqrt(D)) / A);
  return ret;
}
// signed area of intersection of crc(c.o, c.r) &&
// tri(c.o, s.a, s.b) [crs(a-o, b-o)/2]
Tf crcTriIntscnArea(Crc c, Seg s) {
  using Linear::disPtSeg;
  Tf OA = len(c.o - s.a);
  Tf OB = len(c.o - s.b);
  // sector
  if(dcmp(disPtSeg(c.o, s) - c.r) >= 0)
    return angleBetween(s.a-c.o, s.b-c.o) * (c.r * c.r) /
        2.0;
  // tri
  if(dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)
    return crs(c.o - s.b, s.a - s.b) / 2.0;
  // three part: (A, a) (a, b) (b, B)
  vec_p Sect = crcLineIntscn(c, s);
  return crcTriIntscnArea(c, Seg(s.a, Sect[0])) +
      crcTriIntscnArea(c, Seg(Sect[0], Sect[1])) +
      crcTriIntscnArea(c, Seg(Sect[1], s.b));
}
// area of intersecion of crc(c.o, c.r) && simple polyson
    (p[])
// Tested : https://codeforces.com/gym/100204/problem/F -
     Little Mammoth
Tf crcPolyIntscnArea(Crc c, Poly p) {
  Tf res = 0;
  int n = p.size();
  for(int i = 0; i < n; ++i)
    res += crcTriIntscnArea(c, Seg(p[i], p[(i + 1) % n]))
        ;
  return abs(res);
}
// locates crc c2 relative to c1
// interior       (d < R - r)    ----> -2
// interior tangents (d = R - r) ----> -1
// concentric   (d = 0)
// secants      (R - r < d < R + r) ----> 0
// exterior tangents (d = R + r)  ----> 1
// exterior      (d > R + r)    ----> 2
int crcCrcPosition(Crc c1, Crc c2) {
  Tf d = len(c1.o - c2.o);
  int in = dcmp(d - abs(c1.r - c2.r)), ex = dcmp(d - (c1.
      r + c2.r));
  return in < 0 ? -2 : in == 0 ? -1 : ex == 0 ? 1 : ex >
      0 ? 2 : 0;
}
// compute the intersection pts between two crcs c1 && c2
vec_p crcCrcIntscn(Crc c1, Crc c2) {
  vec_p ret;
  Tf d = len(c1.o - c2.o);
  if(dcmp(d) == 0) return ret;
  if(dcmp(c1.r + c2.r - d) < 0) return ret;
  if(dcmp(abs(c1.r - c2.r) - d) > 0) return ret;
  Pt v = c2.o - c1.o;
  Tf co = (c1.r * c1.r + sqlen(v) - c2.r * c2.r) / (2 *
      c1.r * len(v));
  Tf si = sqrt(abs(1.0 - co * co));
  Pt p1 = scale(rotatePrecise(v, co, -si), c1.r) + c1.o;
  Pt p2 = scale(rotatePrecise(v, co, si), c1.r) + c1.o;
  ret.push_back(p1);
  if(p1 != p2) ret.push_back(p2);
  return ret;
}
// intersection area between two crcs c1, c2
Tf crcCrcIntscnArea(Crc c1, Crc c2) {
  Pt AB = c2.o - c1.o;
  Tf d = len(AB);
  if(d >= c1.r + c2.r) return 0;
  if(d + c1.r <= c2.r) return PI * c1.r * c1.r;
  if(d + c2.r <= c1.r) return PI * c2.r * c2.r;
  Tf alpha1 = acos((c1.r * c1.r + d * d - c2.r * c2.r) /
      (2.0 * c1.r * d));
  Tf alpha2 = acos((c2.r * c2.r + d * d - c1.r * c1.r) /
      (2.0 * c2.r * d));
  return c1.sector(2 * alpha1) + c2.sector(2 * alpha2);
}
// returns tangents from a pt p to crc c
vec_p ptCrcTngs(Pt p, Crc c) {
  vec_p ret;
  Pt u = c.o - p;
  Tf d = len(u);
  if(d < c.r) ;
  else if(dcmp(d - c.r) == 0) {
    ret = { rotate(u, PI / 2) };
  }
  else {
    Tf ang = asin(c.r / d);
    ret = { rotate(u, -ang), rotate(u, ang) };
  }
  return ret;
}
// returns the pts on tangents that touches the crc
vec_p ptCrcTngPts(Pt p, Crc c) {
  Pt u = p - c.o;
  Tf d = len(u);
  if(d < c.r) return {};
  else if(dcmp(d - c.r) == 0) return {c.o + u};
  else {
    Tf ang = acos(c.r / d);
    u = u / len(u) * c.r;
    return { c.o + rotate(u, -ang), c.o + rotate(u, ang)
        };
  }
}
// for two crcs c1 && c2, returns two list of pts a && b
// such that a[i] is on c1 && b[i] is c2 && for every i
// Line(a[i], b[i]) is a tangent to both crcs
// CAUTION: a[i] = b[i] in case they touch | -1 for c1 =
    c2
int crcCrcTngPts(Crc c1, Crc c2, vec_p &a, vec_p &b) {
  a.clear(), b.clear();
  int cnt = 0;
  if(dcmp(c1.r - c2.r) < 0) {
    swap(c1, c2); swap(a, b);
  }
  Tf d2 = sqlen(c1.o - c2.o);
  Tf rdif = c1.r - c2.r, rsum = c1.r + c2.r;
  if(dcmp(d2 - rdif * rdif) < 0) return 0;
  if(dcmp(d2) == 0 && dcmp(c1.r - c2.r) == 0) return -1;
  Tf base = angle(c2.o - c1.o);
  if(dcmp(d2 - rdif * rdif) == 0) {
    a.push_back(c1.pt(base));
    b.push_back(c2.pt(base));
    cnt++;
    return cnt;
  }
  Tf ang = acos((c1.r - c2.r) / sqrt(d2));
  a.push_back(c1.pt(base + ang));
  b.push_back(c2.pt(base + ang));
  cnt++;
  a.push_back(c1.pt(base - ang));
```

```
  b.push_back(c2.pt(base - ang));
  cnt++;
  if(dcmp(d2 - rsum * rsum) == 0) {
    a.push_back(c1.pt(base));
    b.push_back(c2.pt(PI + base));
    cnt++;
  }
  else if(dcmp(d2 - rsum * rsum) > 0) {
    Tf ang = acos((c1.r + c2.r) / sqrt(d2));
    a.push_back(c1.pt(base + ang));
    b.push_back(c2.pt(PI + base + ang));
    cnt++;
    a.push_back(c1.pt(base - ang));
    b.push_back(c2.pt(PI + base - ang));
    cnt++;
  }
  return cnt;
}
```

4.6 Half Plane

```
using Linear::lineLineIntscn;
struct DirLine {
  Pt p, v;
  Tf ang;
  DirLine() {}
  /// Directed line containing pt P in the direction v
  DirLine(Pt p, Pt v) : p(p), v(v) { ang = atan2(v.y, v.x
      ); }
  bool operator<(const DirLine& u) const { return ang < u
      .ang; }
};
// returns true if pt p is on the ccw-left side of ray l
bool onLeft(DirLine l, Pt p) { return dcmp(crs(l.v, p-l.p
    )) >= 0; }
// Given a set of directed lines returns a polygon such
    that
// the polygon is the intersection by halfplanes created
    by the
// left side of the directed lines. MAY CONTAIN DUPLICATE
     ptS
int halfPlaneIntscn(vector<DirLine> &li, Poly &poly) {
  int n = li.size();
  sort(li.begin(), li.end());
  int first, last;
  Pt* p = new Pt[n];
  DirLine* q = new DirLine[n];
  q[first = last = 0] = li[0];
  for(int i = 1; i < n; i++) {
    while(first < last && !onLeft(li[i], p[last - 1]))
        last--;
    while(first < last && !onLeft(li[i], p[first])) first
        ++;
    q[++last] = li[i];
    if(dcmp(crs(q[last].v, q[last-1].v)) == 0) {
      last--;
      if(onLeft(q[last], li[i].p)) q[last] = li[i];
    }
    if(first < last)
      lineLineIntscn(q[last - 1].p, q[last - 1].v, q[last
          ].p, q[last].v, p[last - 1]);
  }
  while(first < last && !onLeft(q[first], p[last - 1]))
      last--;
  if(last - first <= 1) {
    delete[] p;
    delete[] q;
    poly.clear();
    return 0;
  }
  lineLineIntscn(q[last].p, q[last].v, q[first].p, q[
      first].v, p[last]);
  int m = 0;
  poly.resize(last - first + 1);
  for(int i = first; i <= last; i++) poly[m++] = p[i];
  delete[] p;
  delete[] q;
  return m;
}
```

5 Graph

5.1 Graph Template

```
struct edge {
  int u, v;
  edge(int u = 0, int v = 0) : u(u), v(v) {}
  int to(int node) { return u ^ v ^ node; }
};
struct graph {
  int n;
  vector<vector<int>> adj;
  vector<edge> edges;
  graph(int n = 0) : n(n), adj(n) {}
  void addEdge(int u, int v, bool dir = 1) {
    adj[u].push_back(edges.size());
    if (dir) adj[v].push_back(edges.size());
    edges.emplace_back(u, v);
  }
  int addNode() {
    adj.emplace_back();
    return n++;
  }
  edge &operator()(int idx) { return edges[idx]; }
  vector<int> &operator[](int u) { return adj[u]; }
};
```

5.2 Lifting, LCA, HLD

```
using Tree = vector<vector<int>>;
int anc[B][N], sz[N], lvl[N], st[N], en[N], nxt[N], t =
    0;

void initLifting(int n) {
  for (int b = 1; b < B; b++) {
    for (int i = 0; i < n; i++) {
      anc[b][i] = anc[b - 1][anc[b - 1][i]];
    }
  }
}

int kthAncestor(int u, int k) {
  for (int b = 0; b < B; b++) {
    if (k >> b & 1) u = anc[b][u];
  }
  return u;
}

int lca(int u, int v) {
  if (lvl[u] > lvl[v]) swap(u, v);
  v = kthAncestor(v, lvl[v] - lvl[u]);

  if (u == v) return u;

  for (int b = B - 1; b >= 0; b--) {
    if (anc[b][u] != anc[b][v]) u = anc[b][u], v = anc[b
        ][v];
  }
  return anc[0][u];
}

int dis(int u, int v) {
  int g = lca(u, v);
  return lvl[u] + lvl[v] - 2 * lvl[g];
}
bool isAncestor(int u, int v) { return st[v] <= st[u] and
     en[u] <= en[v]; }

void tour(int u, int p, Tree &T) {
  st[u] = t++;
  int idx = 0;
  for (int v : T[u]) {
    if (v == p) continue;
    nxt[v] = (idx++ ? v : nxt[u]); // only for hld
```

```cpp
    anc[0][v] = u, lvl[v] = lvl[u] + 1;
    tour(v, u, T);
  }
  en[u] = t; // [st, en] contains subtree range
}

void hld(int u, int p, Tree &T) {
  sz[u] = 1;

  int eld = 0, mx = 0, idx = 0;
  for (int i = 0; i < T[u].size(); i++) {
    int v = T[u][i];
    if (v == p) continue;
    hld(v, u, T);

    if (sz[v] > mx) mx = sz[v], eld = i;
    sz[u] += sz[v];
  }
  swap(T[u][0], T[u][eld]);
}

LL climbQuery(int u, int g) {
  LL ans = -INF;
  while (1) {
    int _u = nxt[u];
    if (isAncestor(g, _u)) _u = g;
    ans = max(ans, rmq ::query(st[_u], st[u]));

    if (_u == g) break;
    u = anc[0][_u];
  }
  return ans;
}

LL pathQuery(int u, int v) {
  int g = lca(u, v);
  return max(climbQuery(u, g), climbQuery(v, g));
}

void init(int u, Tree &T) {
  int n = T.size();
  anc[0][u] = nxt[u] = u;
  lvl[u] = 0;
  hld(u, u, T);
  tour(u, u, T);
  initLifting(n);
}
```

5.3 SCC

```cpp
vector<int> order, comp, idx;
vector<bool> vis;
vector<vector<int>> comps;
Graph dag;

void dfs1(int u, Graph &G, string s = "") {
  vis[u] = 1;
  for (int e : G[u]) {
    int v = G(e).to(u);
    if (!vis[v]) dfs1(v, G, s);
  }
  order.push_back(u);
}
void dfs2(int u, Graph &R) {
  comp.push_back(u);
  idx[u] = comps.size();

  for (int e : R[u]) {
    int v = R(e).to(u);
    if (idx[v] == -1) dfs2(v, R);
  }
}

void init(Graph &G) {
  int n = G.n;
  vis.assign(n, 0);
  idx.assign(n, -1);

  for (int i = 0; i < n; i++) {
    if (!vis[i]) dfs1(i, G);
  }
  reverse(order.begin(), order.end());

  Graph R(n);
  for (auto &e : G.edges) R.addEdge(e.v, e.u, 0);

  for (int u : order) {
    if (idx[u] != -1) continue;
    comp.clear();
    dfs2(u, R);
    comps.push_back(comp);
  }
}

Graph &createDAG(Graph &G) {
  int sz = comps.size();
  dag = Graph(sz);

  vector<bool> taken(sz);
  vector<int> cur;
  for (int i = 0; i < sz; i++) {
    cur.clear();
    taken[i] = 1;
    for (int u : comps[i]) {
      for (int e : G[u]) {
        int v = G(e).to(u);
        int j = idx[v];
        if (taken[j]) continue; // rejects multi-edge
        dag.addEdge(i, j, 0);
        taken[j] = 1;
        cur.push_back(j);
      }
    }
    for (int j : cur) taken[j] = 0;
  }
  return dag;
}
```

5.4 Centroid Decompose

```cpp
namespace ct {
int par[N], cnt[N], cntp[N];
LL sum[N], sump[N];
void activate(int u) {
  int v = u, _u = u;

  ans += sum[u];
  cnt[u]++;
  while (par[u] != -1) {
    u = par[u];
    LL d = ta ::dis(_u, u);
    ans += sum[u] - sump[v];
    ans += d * (cnt[u] - cntp[v]);

    sum[u] += d;
    cnt[u]++;

    sump[v] += d;
    cntp[v]++;

    v = u;
  }
}
}
namespace ctrd {
int sz[N];
bool blk[N];

int szCalc(Tree &T, int u, int p = -1) {
  sz[u] = 1;
  for (int v : T[u]) {
```

```
    if (v == p or blk[v]) continue;
    sz[u] += szCalc(T, v, u);
  }
  return sz[u];
}
int getCentroid(Tree &T, int u, int s, int p = -1) {
  for (int v : T[u]) {
    if (v == p or blk[v]) continue;
    if (sz[v] * 2 >= s) return getCentroid(T, v, s, u);
  }
  return u;
}

void decompose(Tree &T, int u, int p = -1) {
  szCalc(T, u);
  u = getCentroid(T, u, sz[u]);
  ct ::par[u] = p;

  blk[u] = 1;
  for (int v : T[u]) {
    if (!blk[v]) decompose(T, v, u);
  }
}
}
```

5.5 Euler Tour on Edge

```
// for simplicity, G[idx] contains the adjacency list of
    a node
// while G(e) is a reference to the e-th edge.
const int N = 2e5 + 5;
int in[N], out[N], fwd[N], bck[N];
int t = 0;
void dfs(graph &G, int node, int par) {
  out[node] = t;
  for (int e : G[node]) {
    int v = G(e).to(node);
    if (v == par) continue;
    fwd[e] = t++;
    dfs(G, v, node);
    bck[e] = t++;
  }
  in[node] = t - 1;
}
void init(graph &G, int node) {
  t = 0;
  dfs(G, node, node);
}
```

5.6 Virtual Tree

```
namespace lca1 {
int st[N], lvl[N];
int tbl[B][2 * N];
int t = 0;

void dfs(int u, int p, Tree &T) {
  st[u] = t;
  tbl[0][t++] = u;
  for(int v: T[u]) {
    if(v == p) continue;
    lvl[v] = lvl[u] + 1;
    dfs(v, u, T);
    tbl[0][t++] = u;
  }
}
int low(int u, int v) {
  return make_pair(lvl[u], u) < make_pair(lvl[v], v) ? u
      : v;
}

void makeTable(int n) {
  int m = 2 * n - 1;
  for(int b = 1; b < B; b++) {
    for(int i = 0; i < m; i++) {
      tbl[b][i] = low(tbl[b - 1][i], tbl[b - 1][i + (1 <<
          b - 1)]);
    }
  }
}

int lca(int u, int v) {
  int l = st[u], r = st[v];
  if(l > r) swap(l, r);
  int k = __lg(r - l + 1);
  return low(tbl[k][l], tbl[k][r - (1 << k) + 1]);
}
void init(int root, Tree &T) {
  lvl[root] = 0;
  t = 0;
  dfs(root, root, T);
  makeTable(T.size());
}
}
namespace vt {
int st[N], en[N], t;
vector <int> adj[N];

void dfs(int u, int p, Tree &T) {
  st[u] = t++;
  for(int v: T[u]) if(v != p) dfs(v, u, T);
  en[u] = t++;
}
bool comp(int u, int v) {
  return st[u] < st[v];
}
bool isAncestor(int u, int p) {
  return st[p] <= st[u] and en[u] <= en[p];
}

void construct(vector <int> &nodes) {
  sort(nodes.begin(), nodes.end(), comp);
  int n = nodes.size();
  for(int i = 0; i + 1 < n; i++) {
    nodes.push_back(lca1 :: lca(nodes[i], nodes[i + 1]));
  }
  sort(nodes.begin(), nodes.end(), comp);
  nodes.erase(unique(nodes.begin(), nodes.end()), nodes.
      end());
  n = nodes.size();
  stack <int> s;
  s.push(nodes[0]);
  for(int i = 1; i < n; i++) {
    int u = nodes[i];
    while(not isAncestor(u, s.top())) s.pop();
    adj[s.top()].push_back(u);
    s.push(u);
  }
}
void clear(vector <int> &nodes) {
  for(int u: nodes) {
      adj[u].clear();
  }
}

void init(int root, Tree &T) {
  lca1 :: init(root, T);
  t = 0;
  dfs(root, root, T);
}
}
```

5.7 Dinic Max Flow

```
/// flow with demand(lower bound) only for DAG
// create new src and sink
// add_edge(new src, u, sum(in_demand[u]))
// add_edge(u, new sink, sum(out_demand[u]))
// add_edge(old sink, old src, inf)
//  if (sum of lower bound == flow) then demand satisfied
// flow in every edge i = demand[i] + e.flow

using Ti = long long;
```

```
const Ti INF = 1LL << 60;
struct edge {
  int v, u;
  Ti cap, flow = 0;
  edge(int v, int u, Ti cap) : v(v), u(u), cap(cap) {}
};
const int N = 1e5 + 50;
vector<edge> edges;
vector<int> adj[N];
int m = 0, n;
int level[N], ptr[N];
queue<int> q;
bool bfs(int s, int t) {
  for (q.push(s), level[s] = 0; !q.empty(); q.pop()) {
    for (int id : adj[q.front()]) {
      auto &ed = edges[id];
      if (ed.cap - ed.flow > 0 and level[ed.u] == -1)
        level[ed.u] = level[ed.v] + 1, q.push(ed.u);
    }
  }
  return level[t] != -1;
}
Ti dfs(int v, Ti pushed, int t) {
  if (pushed == 0) return 0;
  if (v == t) return pushed;
  for (int &cid = ptr[v]; cid < adj[v].size(); cid++) {
    int id = adj[v][cid];
    auto &ed = edges[id];
    if (level[v] + 1 != level[ed.u] || ed.cap - ed.flow <
        1) continue;
    Ti tr = dfs(ed.u, min(pushed, ed.cap - ed.flow), t);
    if (tr == 0) continue;
    ed.flow += tr;
    edges[id ^ 1].flow -= tr;
    return tr;
  }
  return 0;
}
void init(int nodes) {
  m = 0, n = nodes;
  for (int i = 0; i < n; i++) level[i] = -1, ptr[i] = 0,
      adj[i].clear();
}
void addEdge(int v, int u, Ti cap) {
  edges.emplace_back(v, u, cap), adj[v].push_back(m++);
  edges.emplace_back(u, v, 0), adj[u].push_back(m++);
}
Ti maxFlow(int s, int t) {
  Ti f = 0;
  for (auto &ed : edges) ed.flow = 0;
  for (; bfs(s, t); memset(level, -1, n * 4)) {
    for (memset(ptr, 0, n * 4); Ti pushed = dfs(s, INF, t
        ); f += pushed)
      ;
  }
  return f;
}
```

5.8 Min Cost Max Flow

```
mt19937 rnd(chrono::steady_clock::now().time_since_epoch
    ().count());
const LL inf = 1e9;
struct edge {
  int v, rev;
  LL cap, cost, flow;
  edge() {}
  edge(int v, int rev, LL cap, LL cost)
      : v(v), rev(rev), cap(cap), cost(cost), flow(0) {}
};
struct mcmf {
  int src, sink, n;
  vector<int> par, idx, Q;
  vector<bool> inq;
  vector<LL> dis;
  vector<vector<edge>> g;
  mcmf() {}
  mcmf(int src, int sink, int n)
      : src(src),
        sink(sink),
        n(n),
        par(n),
        idx(n),
        inq(n),
        dis(n),
        g(n),
        Q(10000005) {} // use Q(n) if not using random
  void add_edge(int u, int v, LL cap, LL cost, bool
      directed = true) {
    edge _u = edge(v, g[v].size(), cap, cost);
    edge _v = edge(u, g[u].size(), 0, -cost);
    g[u].pb(_u);
    g[v].pb(_v);
    if (!directed) add_edge(v, u, cap, cost, true);
  }
  bool spfa() {
    for (int i = 0; i < n; i++) {
      dis[i] = inf, inq[i] = false;
    }
    int f = 0, l = 0;
    dis[src] = 0, par[src] = -1, Q[l++] = src, inq[src] =
        true;
    while (f < l) {
      int u = Q[f++];
      for (int i = 0; i < g[u].size(); i++) {
        edge &e = g[u][i];
        if (e.cap <= e.flow) continue;
        if (dis[e.v] > dis[u] + e.cost) {
          dis[e.v] = dis[u] + e.cost;
          par[e.v] = u, idx[e.v] = i;
          if (!inq[e.v]) inq[e.v] = true, Q[l++] = e.v;
          // if (!inq[e.v]) {
          //   inq[e.v] = true;
          //   if (f && rnd() & 7) Q[--f] = e.v;
          //   else Q[l++] = e.v;
          // }
        }
      }
      inq[u] = false;
    }
    return (dis[sink] != inf);
  }
  pair<LL, LL> solve() {
    LL mincost = 0, maxflow = 0;
    while (spfa()) {
      LL bottleneck = inf;
      for (int u = par[sink], v = idx[sink]; u != -1; v =
          idx[u], u = par[u]) {
        edge &e = g[u][v];
        bottleneck = min(bottleneck, e.cap - e.flow);
      }
      for (int u = par[sink], v = idx[sink]; u != -1; v =
          idx[u], u = par[u]) {
        edge &e = g[u][v];
        e.flow += bottleneck;
        g[e.v][e.rev].flow -= bottleneck;
      }
      mincost += bottleneck * dis[sink], maxflow +=
          bottleneck;
    }
    return make_pair(mincost, maxflow);
  }
};
// want to minimize cost and don't care about flow
// add edge from sink to dummy sink (cap = inf, cost = 0)
// add edge from source to sink (cap = inf, cost = 0)
// run mcmf, cost returned is the minimum cost
```

5.9 Block Cut Tree

```
vector<vector<int> > components;
```

```cpp
vector<int> cutpoints, start, low;
vector<bool> is_cutpoint;
stack<int> st;
void find_cutpoints(int node, graph &G, int par = -1, int d = 0) {
  low[node] = start[node] = d++;
  st.push(node);
  int cnt = 0;
  for (int e : G[node])
    if (int to = G(e).to(node); to != par) {
      if (start[to] == -1) {
        find_cutpoints(to, G, node, d + 1);
        cnt++;
        if (low[to] >= start[node]) {
          is_cutpoint[node] = par != -1 or cnt > 1;
          components.push_back({node}); // starting a new block with the point
          while (st.top() != node)
            components.back().push_back(st.top()), st.pop();
        }
      }
      low[node] = min(low[node], low[to]);
    }
}
graph tree;
vector<int> id;
void init(graph &G) {
  int n = G.n;
  start.assign(n, -1), low.resize(n), is_cutpoint.resize(n), id.assign(n, -1);
  find_cutpoints(0, G);
  for (int u = 0; u < n; ++u)
    if (is_cutpoint[u]) id[u] = tree.addNode();
  for (auto &comp : components) {
    int node = tree.addNode();
    for (int u : comp)
      if (!is_cutpoint[u])
        id[u] = node;
      else
        tree.addEdge(node, id[u]);
  }
  if (id[0] == -1) // corner - 1
    id[0] = tree.addNode();
}
```

5.10 Bridge Tree

```cpp
vector<vector<int>> comps;
vector<int> depth, low, id;
stack<int> st;
vector<Edge> bridges;
Graph tree;

void dfs(int u, Graph &G, int ed = -1, int d = 0) {
  low[u] = depth[u] = d;
  st.push(u);
  for (int e : G[u]) {
    if (e == ed) continue;
    int v = G(e).to(u);
    if (depth[v] == -1) dfs(v, G, e, d + 1);
    low[u] = min(low[u], low[v]);

    if (low[v] <= depth[u]) continue;
    bridges.emplace_back(u, v);
    comps.emplace_back();
    do {
      comps.back().push_back(st.top()), st.pop();
    } while (comps.back().back() != v);
  }
  if (ed == -1) {
    comps.emplace_back();
    while (!st.empty()) comps.back().push_back(st.top()), st.pop();
  }
}
Graph &createTree() {
  for (auto &comp : comps) {
    int idx = tree.addNode();
    for (auto &e : comp) id[e] = idx;
  }
  for (auto &[l, r] : bridges) tree.addEdge(id[l], id[r]);
  return tree;
}

void init(Graph &G) {
  int n = G.n;
  depth.assign(n, -1), id.assign(n, -1), low.resize(n);
  for (int i = 0; i < n; i++) {
    if (depth[i] == -1) dfs(i, G);
  }
}
```

5.11 Tree Isomorphism

```cpp
mp["01"] = 1;
ind = 1;
int dfs(int u, int p) {
  int cnt = 0;
  vector<int> vs;
  for (auto v : g1[u]) {
    if (v != p) {
      int got = dfs(v, u);
      vs.pb(got);
      cnt++;
    }
  }
  if (!cnt) return 1;

  sort(vs.begin(), vs.end());
  string s = "0";
  for (auto i : vs) s += to_string(i);
  vs.clear();
  s.pb('1');
  if (mp.find(s) == mp.end()) mp[s] = ++ind;
  int ret = mp[s];
  return ret;
}
```

6 Math

6.1 Combi

```cpp
array<int, N + 1> fact, inv, inv_fact;
void init() {
  fact[0] = inv_fact[0] = 1;
  for (int i = 1; i <= N; i++) {
    inv[i] = i == 1 ? 1 : (LL)inv[i - mod % i] * (mod / i + 1) % mod;
    fact[i] = (LL)fact[i - 1] * i % mod;
    inv_fact[i] = (LL)inv_fact[i - 1] * inv[i] % mod;
  }
}
LL C(int n, int r) {
  return (r < 0 or r > n) ? 0 : (LL)fact[n] * inv_fact[r] % mod * inv_fact[n - r] % mod;
}
```

6.2 Linear Sieve

```cpp
const int N = 1e7;
vector<int> primes;
int spf[N + 5], phi[N + 5], NOD[N + 5], cnt[N + 5], POW[N + 5];
bool prime[N + 5];
int SOD[N + 5];
void init() {
  fill(prime + 2, prime + N + 1, 1);
  SOD[1] = NOD[1] = phi[1] = spf[1] = 1;
  for (LL i = 2; i <= N; i++) {
    if (prime[i]) {
      primes.push_back(i), spf[i] = i;
      phi[i] = i - 1;
      NOD[i] = 2, cnt[i] = 1;
```

```
      SOD[i] = i + 1, POW[i] = i;
    }
    for (auto p : primes) {
      if (p * i > N or p > spf[i]) break;
      prime[p * i] = false, spf[p * i] = p;
      if (i % p == 0) {
        phi[p * i] = p * phi[i];
        NOD[p * i] = NOD[i] / (cnt[i] + 1) * (cnt[i] + 2), cnt[p * i] = cnt[i] + 1;
        SOD[p * i] = SOD[i] / SOD[POW[i]] * (SOD[POW[i]] + p * POW[i]), POW[p * i] = p * POW[i];
        break;
      } else {
        phi[p * i] = phi[p] * phi[i];
        NOD[p * i] = NOD[p] * NOD[i], cnt[p * i] = 1;
        SOD[p * i] = SOD[p] * SOD[i], POW[p * i] = p;
      }
    }
  }
}
```

6.3 Pollard Rho

```
LL mul(LL a, LL b, LL mod) {
  return (__int128) a * b % mod;
  // LL ans = a * b - mod * (LL) (1.L / mod * a * b);
  // return ans + mod * (ans < 0) - mod * (ans >= (LL) mod);
}
LL bigmod(LL num, LL pow, LL mod) {
  LL ans = 1;
  for (; pow > 0; pow >>= 1, num = mul(num, num, mod)){
    if (pow & 1) ans = mul(ans, num, mod);
  }
  return ans;
}
bool is_prime(LL n) {
  if (n < 2 or n % 6 % 4 != 1) return (n | 1) == 3;
  LL a[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
  LL s = __builtin_ctzll(n - 1), d = n >> s;
  for (LL x : a) {
    LL p = bigmod(x % n, d, n), i = s;
    for (; p != 1 and p != n - 1 and x % n and i--; p = mul(p, p, n));
    if (p != n - 1 and i != s) return false;
  }
  return true;
}
LL get_factor(LL n) {
  auto f = [&](LL x) { return mul(x, x, n) + 1; };
  LL x = 0, y = 0, t = 0, prod = 2, i = 2, q;
  for (; t++ % 40 or gcd(prod, n) == 1; x = f(x), y = f(f(y))) {
    (x == y) ? x = i++, y = f(x) : 0;
    prod = (q = mul(prod, max(x, y) - min(x, y), n)) ? q : prod;
  }
  return gcd(prod, n);
}
map<LL, int> factorize(LL n) {
  map<LL, int> res;
  if (n < 2) return res;
  LL small_primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97};
  for (LL p : small_primes) {
    for (; n % p == 0; n /= p) res[p]++;
  }
  auto _factor = [&](LL n, auto &_factor) {
    if (n == 1) return;
    if (is_prime(n))
      res[n]++;
    else {
      LL x = get_factor(n);
      _factor(x, _factor);
      _factor(n / x, _factor);
    }
  };
  _factor(n, _factor);
  return res;
}
```

6.4 Chinese Remainder Theorem

```
// given a, b will find solutions for
// ax + by = 1
tuple<LL, LL, LL> EGCD(LL a, LL b) {
  if (b == 0)
    return {1, 0, a};
  else {
    auto [x, y, g] = EGCD(b, a % b);
    return {y, x - a / b * y, g};
  }
}
// given modulo equations, will apply CRT
PLL CRT(vector<PLL> &v) {
  LL V = 0, M = 1;
  for (auto &[v, m] : v) { // value % mod
    auto [x, y, g] = EGCD(M, m);
    if ((v - V) % g != 0) return {-1, 0};
    V += x * (v - V) / g % (m / g) * M, M *= m / g;
    V = (V % M + M) % M;
  }
  return make_pair(V, M);
}
```

6.5 Mobius Function

```
const int N = 1e6 + 5;
int mob[N];
void mobius() {
  memset(mob, -1, sizeof mob);
  mob[1] = 1;
  for (int i = 2; i < N; i++)
    if (mob[i]) {
      for (int j = i + i; j < N; j += i) mob[j] -= mob[i];
    }
}
```

6.6 FFT

```
using CD = complex<double>;
typedef long long LL;
const double PI = acos(-1.0L);

int N;
vector<int> perm;
vector<CD> wp[2];
void precalculate(int n) {
  assert((n & (n - 1)) == 0), N = n;
  perm = vector<int>(N, 0);
  for (int k = 1; k < N; k <<= 1) {
    for (int i = 0; i < k; i++) {
      perm[i] <<= 1;
      perm[i + k] = 1 + perm[i];
    }
  }
  wp[0] = wp[1] = vector<CD>(N);
  for (int i = 0; i < N; i++) {
    wp[0][i] = CD(cos(2 * PI * i / N), sin(2 * PI * i / N));
    wp[1][i] = CD(cos(2 * PI * i / N), -sin(2 * PI * i / N));
  }
}
void fft(vector<CD> &v, bool invert = false) {
  if (v.size() != perm.size()) precalculate(v.size());
  for (int i = 0; i < N; i++)
    if (i < perm[i]) swap(v[i], v[perm[i]]);
  for (int len = 2; len <= N; len *= 2) {
```

```
    for (int i = 0, d = N / len; i < N; i += len) {
      for (int j = 0, idx = 0; j < len / 2; j++, idx += d
          ) {
        CD x = v[i + j];
        CD y = wp[invert][idx] * v[i + j + len / 2];
        v[i + j] = x + y;
        v[i + j + len / 2] = x - y;
      }
    }
  }
  if (invert) {
    for (int i = 0; i < N; i++) v[i] /= N;
  }
}
void pairfft(vector<CD> &a, vector<CD> &b, bool invert =
     false) {
  int N = a.size();
  vector<CD> p(N);
  for (int i = 0; i < N; i++) p[i] = a[i] + b[i] * CD(0,
      1);
  fft(p, invert);
  p.push_back(p[0]);
  for (int i = 0; i < N; i++) {
    if (invert) {
      a[i] = CD(p[i].real(), 0);
      b[i] = CD(p[i].imag(), 0);
    } else {
      a[i] = (p[i] + conj(p[N - i])) * CD(0.5, 0);
      b[i] = (p[i] - conj(p[N - i])) * CD(0, -0.5);
    }
  }
}
vector<LL> multiply(const vector<LL> &a, const vector<LL>
     &b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  vector<CD> fa(a.begin(), a.end()), fb(b.begin(), b.end
      ());
  fa.resize(n);
  fb.resize(n);
  //      fft(fa); fft(fb);
  pairfft(fa, fb);
  for (int i = 0; i < n; i++) fa[i] = fa[i] * fb[i];
  fft(fa, true);
  vector<LL> ans(n);
  for (int i = 0; i < n; i++) ans[i] = round(fa[i].real()
      );
  return ans;
}
const int M = 1e9 + 7, B = sqrt(M) + 1;
vector<LL> anyMod(const vector<LL> &a, const vector<LL> &
    b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  vector<CD> al(n), ar(n), bl(n), br(n);
  for (int i = 0; i < a.size(); i++) al[i] = a[i] % M / B
      , ar[i] = a[i] % M % B;
  for (int i = 0; i < b.size(); i++) bl[i] = b[i] % M / B
      , br[i] = b[i] % M % B;
  pairfft(al, ar);
  pairfft(bl, br);
  //      fft(al); fft(ar); fft(bl); fft(br);
  for (int i = 0; i < n; i++) {
    CD ll = (al[i] * bl[i]), lr = (al[i] * br[i]);
    CD rl = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
    al[i] = ll;
    ar[i] = lr;
    bl[i] = rl;
    br[i] = rr;
  }
  pairfft(al, ar, true);
  pairfft(bl, br, true);
  //      fft(al, true); fft(ar, true); fft(bl, true);
      fft(br, true);
  vector<LL> ans(n);
  for (int i = 0; i < n; i++) {
    LL right = round(br[i].real()), left = round(al[i].
        real());
    ;
    LL mid = round(round(bl[i].real()) + round(ar[i].real
        ()));
    ans[i] = ((left % M) * B * B + (mid % M) * B + right)
         % M;
  }
  return ans;
}
```

6.7 NTT

```
const LL N = 1 << 18;
const LL MOD = 786433;

vector<LL> P[N];
LL rev[N], w[N | 1], a[N], b[N], inv_n, g;
LL Pow(LL b, LL p) {
  LL ret = 1;
  while (p) {
    if (p & 1) ret = (ret * b) % MOD;
    b = (b * b) % MOD;
    p >>= 1;
  }
  return ret;
}
LL primitive_root(LL p) {
  vector<LL> factor;
  LL phi = p - 1, n = phi;
  for (LL i = 2; i * i <= n; i++) {
    if (n % i) continue;
    factor.emplace_back(i);
    while (n % i == 0) n /= i;
  }
  if (n > 1) factor.emplace_back(n);
  for (LL res = 2; res <= p; res++) {
    bool ok = true;
    for (LL i = 0; i < factor.size() && ok; i++)
      ok &= Pow(res, phi / factor[i]) != 1;
    if (ok) return res;
  }
  return -1;
}
void prepare(LL n) {
  LL sz = abs(31 - __builtin_clz(n));
  LL r = Pow(g, (MOD - 1) / n);
  inv_n = Pow(n, MOD - 2);
  w[0] = w[n] = 1;
  for (LL i = 1; i < n; i++) w[i] = (w[i - 1] * r) % MOD;
  for (LL i = 1; i < n; i++)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz - 1));
}
void NTT(LL *a, LL n, LL dir = 0) {
  for (LL i = 1; i < n - 1; i++)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (LL m = 2; m <= n; m <<= 1) {
    for (LL i = 0; i < n; i += m) {
      for (LL j = 0; j < (m >> 1); j++) {
        LL &u = a[i + j], &v = a[i + j + (m >> 1)];
        LL t = v * w[dir ? n - n / m * j : n / m * j] %
            MOD;
        v = u - t < 0 ? u - t + MOD : u - t;
        u = u + t >= MOD ? u + t - MOD : u + t;
      }
    }
  }
  if (dir)
    for (LL i = 0; i < n; i++) a[i] = (inv_n * a[i]) %
        MOD;
}
vector<LL> mul(vector<LL> p, vector<LL> q) {
  LL n = p.size(), m = q.size();
  LL t = n + m - 1, sz = 1;
  while (sz < t) sz <<= 1;
```

```
  prepare(sz);

  for (LL i = 0; i < n; i++) a[i] = p[i];
  for (LL i = 0; i < m; i++) b[i] = q[i];
  for (LL i = n; i < sz; i++) a[i] = 0;
  for (LL i = m; i < sz; i++) b[i] = 0;

  NTT(a, sz);
  NTT(b, sz);
  for (LL i = 0; i < sz; i++) a[i] = (a[i] * b[i]) % MOD;
  NTT(a, sz, 1);

  vector<LL> c(a, a + sz);
  while (c.size() && c.back() == 0) c.pop_back();
  return c;
}
```

6.8 WalshHadamard

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
#define bitwiseXOR 1
// #define bitwiseAND 2
// #define bitwiseOR 3
const LL MOD = 30011;

LL BigMod(LL b, LL p) {
  LL ret = 1;
  while (p > 0) {
    if (p % 2 == 1) {
      ret = (ret * b) % MOD;
    }
    p = p / 2;
    b = (b * b) % MOD;
  }
  return ret % MOD;
}

void FWHT(vector<LL>& p, bool inverse) {
  LL n = p.size();
  assert((n & (n - 1)) == 0);

  for (LL len = 1; 2 * len <= n; len <<= 1) {
    for (LL i = 0; i < n; i += len + len) {
      for (LL j = 0; j < len; j++) {
        LL u = p[i + j];
        LL v = p[i + len + j];

#ifdef bitwiseXOR
        p[i + j] = (u + v) % MOD;
        p[i + len + j] = (u - v + MOD) % MOD;
#endif // bitwiseXOR

#ifdef bitwiseAND
        if (!inverse) {
          p[i + j] = v % MOD;
          p[i + len + j] = (u + v) % MOD;
        } else {
          p[i + j] = (-u + v) % MOD;
          p[i + len + j] = u % MOD;
        }
#endif // bitwiseAND

#ifdef bitwiseOR
        if (!inverse) {
          p[i + j] = u + v;
          p[i + len + j] = u;
        } else {
          p[i + j] = v;
          p[i + len + j] = u - v;
        }
#endif // bitwiseOR
      }
    }
  }

#ifdef bitwiseXOR
  if (inverse) {
    LL val = BigMod(n, MOD - 2); // Option 2: Exclude
    for (LL i = 0; i < n; i++) {
      // assert(p[i]%n==0); //Option 2: Include
      p[i] = (p[i] * val) % MOD; // Option 2: p[i]/=n;
    }
  }
#endif // bitwiseXOR
}
```

6.9 Berlekamp Massey

```
struct berlekamp_massey { // for linear recursion
  typedef long long LL;
  static const int SZ = 2e5 + 5;
  static const int MOD = 1e9 + 7; /// mod must be a prime
  LL m , a[SZ] , h[SZ] , t_[SZ] , s[SZ] , t[SZ];
  // bigmod goes here
  inline vector <LL> BM( vector <LL> &x ) {
    LL lf , ld;
    vector <LL> ls , cur;
    for ( int i = 0; i < int(x.size()); ++i ) {
      LL t = 0;
      for ( int j = 0; j < int(cur.size()); ++j ) t = (t
          + x[i - j - 1] * cur[j]) % MOD;
      if ( (t - x[i]) % MOD == 0 ) continue;
      if ( !cur.size() ) {
        cur.resize( i + 1 );
        lf = i; ld = (t - x[i]) % MOD;
        continue;
      }
      LL k = -(x[i] - t) * bigmod( ld , MOD - 2 , MOD ) %
          MOD;
      vector <LL> c(i - lf - 1);
      c.push_back( k );
      for ( int j = 0; j < int(ls.size()); ++j ) c.
          push_back(-ls[j] * k % MOD);
      if ( c.size() < cur.size() ) c.resize( cur.size() )
          ;
      for ( int j = 0; j < int(cur.size()); ++j ) c[j] =
          (c[j] + cur[j]) % MOD;
      if (i - lf + (int)ls.size() >= (int)cur.size() ) ls
          = cur, lf = i, ld = (t - x[i]) % MOD;
      cur = c;
    }
    for ( int i = 0; i < int(cur.size()); ++i ) cur[i] =
        (cur[i] % MOD + MOD) % MOD;
    return cur;
  }
  inline void mull( LL *p , LL *q ) {
    for ( int i = 0; i < m + m; ++i ) t_[i] = 0;
    for ( int i = 0; i < m; ++i ) if ( p[i] )
      for ( int j = 0; j < m; ++j ) t_[i + j] = (t_[i +
          j] + p[i] * q[j]) % MOD;
    for ( int i = m + m - 1; i >= m; --i ) if ( t_[i] )
      for ( int j = m - 1; ~j; --j ) t_[i - j - 1] = (
          t_[i - j - 1] + t_[i] * h[j]) % MOD;
    for ( int i = 0; i < m; ++i ) p[i] = t_[i];
  }
  inline LL calc( LL K ) {
    for ( int i = m; ~i; --i ) s[i] = t[i] = 0;
    s[0] = 1; if ( m != 1 ) t[1] = 1; else t[0] = h[0];
    while ( K ) {
      if ( K & 1 ) mull( s , t );
      mull( t , t ); K >>= 1;
    }
    LL su = 0;
    for ( int i = 0; i < m; ++i ) su = (su + s[i] * a[i])
        % MOD;
    return (su % MOD + MOD) % MOD;
  }
  /// already calculated upto k , now calculate upto n.
```

```
  inline vector <LL> process( vector <LL> &x , int n ,
      int k ) {
    auto re = BM( x );
    x.resize( n + 1 );
    for ( int i = k + 1; i <= n; i++ ) {
      for ( int j = 0; j < re.size(); j++ ) {
        x[i] += 1LL * x[i - j - 1] % MOD * re[j] % MOD; x
            [i] %= MOD;
      }
    }
    return x;
  }
  inline LL work( vector <LL> &x , LL n ) {
    if ( n < int(x.size()) ) return x[n] % MOD;
    vector <LL> v = BM( x ); m = v.size(); if ( !m )
        return 0;
    for ( int i = 0; i < m; ++i ) h[i] = v[i], a[i] = x[i
        ];
    return calc( n ) % MOD;
  }
} rec;
vector <LL> v;
void solve() {
  int n;
  cin >> n;
  cout << rec.work(v, n - 1) << endl;
}
```

6.10 Lagrange

```
// p is a polynomial with n points.
// p(0), p(1), p(2), ... p(n-1) are given.
// Find p(x).
LL Lagrange(vector<LL> &p, LL x) {
  LL n = p.size(), L, i, ret;
  if (x < n) return p[x];
  L = 1;
  for (i = 1; i < n; i++) {
    L = (L * (x - i)) % MOD;
    L = (L * bigmod(MOD - i, MOD - 2)) % MOD;
  }
  ret = (L * p[0]) % MOD;
  for (i = 1; i < n; i++) {
    L = (L * (x - i + 1)) % MOD;
    L = (L * bigmod(x - i, MOD - 2)) % MOD;
    L = (L * bigmod(i, MOD - 2)) % MOD;
    L = (L * (MOD + i - n)) % MOD;
    ret = (ret + L * p[i]) % MOD;
  }
  return ret;
}
```

6.11 Shanks' Baby Step, Giant Step

```
// Finds a^x = b (mod p)

LL bigmod(LL b, LL p, LL m) {}

LL babyStepGiantStep(LL a, LL b, LL p) {
  LL i, j, c, sq = sqrt(p);
  map<LL, LL> babyTable;

  for (j = 0, c = 1; j <= sq; j++, c = (c * a) % p)
      babyTable[c] = j;

  LL giant = bigmod(a, sq * (p - 2), p);

  for (i = 0, c = 1; i <= sq; i++, c = (c * giant) % p) {
    if (babyTable.find((c * b) % p) != babyTable.end())
      return i * sq + babyTable[(c * b) % p];
  }

  return -1;
}
```

6.12 Xor Basis

```
struct XorBasis {
  static const int sz = 64;
  array<ULL, sz> base = {0}, back;
  array<int, sz> pos;
  void insert(ULL x, int p) {
    ULL cur = 0;
    for (int i = sz - 1; ~i; i--)
      if (x >> i & 1) {
        if (!base[i]) {
          base[i] = x, back[i] = cur, pos[i] = p;
          break;
        } else x ^= base[i], cur |= 1ULL << i;
      }
  }
  pair<ULL, vector<int>> construct(ULL mask) {
    ULL ok = 0, x = mask;
    for (int i = sz - 1; ~i; i--)
      if (mask >> i & 1 and base[i]) mask ^= base[i], ok
          |= 1ULL << i;
    vector<int> ans;
    for (int i = 0; i < sz; i++)
      if (ok >> i & 1) {
        ans.push_back(pos[i]);
        ok ^= back[i];
      }
    return {x ^ mask, ans};
  }
};
```

7 String

7.1 Aho Corasick

```
struct AC {
int N, P;
const int A = 26;
vector<vector<int>> next;
vector<int> link, out_link;
vector<vector<int>> out;
AC() : N(0), P(0) { node(); }
int node() {
  next.emplace_back(A, 0);
  link.emplace_back(0);
  out_link.emplace_back(0);
  out.emplace_back(0);
  return N++;
}
inline int get(char c) { return c - 'a'; }
int add_pattern(const string T) {
  int u = 0;
  for (auto c : T) {
    if (!next[u][get(c)]) next[u][get(c)] = node();
    u = next[u][get(c)];
  }
  out[u].push_back(P);
  return P++;
}
void compute() {
  queue<int> q;
  for (q.push(0); !q.empty();) {
    int u = q.front(); q.pop();
    for (int c = 0; c < A; ++c) {
      int v = next[u][c];
      if (!v) next[u][c] = next[link[u]][c];
      else {
        link[v] = u ? next[link[u]][c] : 0;
        out_link[v] = out[link[v]].empty() ? out_link[
            link[v]] : link[v];
        q.push(v);
      }
    }
  }
}
int advance(int u, char c) {
  while (u && !next[u][get(c)]) u = link[u];
  u = next[u][get(c)];
  return u;
}
void match(const string S) {
```

```
  int u = 0;
  for (auto c : S) {
    u = advance(u, c);
    for (int v = u; v; v = out_link[v]) {
      for (auto p : out[v]) cout << "match " << p << endl
          ;
    }
  }
}
};
int main() {
  AC aho; int n; cin >> n;
  while (n--) {
    string s; cin >> s;
    aho.add_pattern(s);
  }
  aho.compute(); string text;
  cin >> text; aho.match(text);
  return 0;
}
```

7.2 Double hash

```
// define +, -, * for (PLL, LL) and (PLL, PLL), % for (
    PLL, PLL);
PLL base(1949313259, 1997293877);
PLL mod(2091573227, 2117566807);

PLL power(PLL a, LL p) {
  PLL ans = PLL(1, 1);
  for(; p; p >>= 1, a = a * a % mod) {
    if(p & 1) ans = ans * a % mod;
  }
  return ans;
}

PLL inverse(PLL a) { return power(a, (mod.ff - 1) * (mod.
    ss - 1) - 1); }
PLL inv_base = inverse(base);
PLL val;
vector<PLL> P;

void hash_init(int n) {
  P.resize(n + 1);
  P[0] = PLL(1, 1);
  for (int i = 1; i <= n; i++) P[i] = (P[i - 1] * base) %
       mod;
}
PLL append(PLL cur, char c) { return (cur * base + c) %
    mod; }
/// prepends c to string with size k
PLL prepend(PLL cur, int k, char c) { return (P[k] * c +
     cur) % mod; }
/// replaces the i-th (0-indexed) character from right
     from a to b;
PLL replace(PLL cur, int i, char a, char b) {
  cur = (cur + P[i] * (b - a)) % mod;
  return (cur + mod) % mod;
}
/// Erases c from the back of the string
PLL pop_back(PLL hash, char c) {
  return (((hash - c) * inv_base) % mod + mod) % mod;
}
/// Erases c from front of the string with size len
PLL pop_front(PLL hash, int len, char c) {
  return ((hash - P[len - 1] * c) % mod + mod) % mod;
}
/// concatenates two strings where length of the right is
     k
PLL concat(PLL left, PLL right, int k) { return (left * P
     [k] + right) % mod; }
/// Calculates hash of string with size len repeated cnt
     times
/// This is O(log n). For O(1), pre-calculate inverses
PLL repeat(PLL hash, int len, LL cnt) {
  PLL mul = (P[len * cnt] - 1) * inverse(P[len] - 1);
  mul = (mul % mod + mod) % mod;
  PLL ret = (hash * mul) % mod;
  if (P[len].ff == 1) ret.ff = hash.ff * cnt;
  if (P[len].ss == 1) ret.ss = hash.ss * cnt;
  return ret;
}
LL get(PLL hash) { return ((hash.ff << 32) ^ hash.ss); }
struct hashlist {
  int len;
  vector<PLL> H, R;
  hashlist() {}
  hashlist(string& s) {
    len = (int)s.size();
    hash_init(len);
    H.resize(len + 1, PLL(0, 0)), R.resize(len + 2, PLL
         (0, 0));
    for (int i = 1; i <= len; i++) H[i] = append(H[i -
         1], s[i - 1]);
    for (int i = len; i >= 1; i--) R[i] = append(R[i +
         1], s[i - 1]);
  }
  /// 1-indexed
  PLL range_hash(int l, int r) {
    return ((H[r] - H[l - 1] * P[r - l + 1]) % mod + mod)
         % mod;
  }
  PLL reverse_hash(int l, int r) {
    return ((R[l] - R[r + 1] * P[r - l + 1]) % mod + mod)
         % mod;
  }
  PLL concat_range_hash(int l1, int r1, int l2, int r2) {
    return concat(range_hash(l1, r1), range_hash(l2, r2),
         r2 - l2 + 1);
  }
  PLL concat_reverse_hash(int l1, int r1, int l2, int r2)
       {
    return concat(reverse_hash(l2, r2), reverse_hash(l1,
        r1), r1 - l1 + 1);
  }
};
```

7.3 Manacher's

```
vector<int> d1(n);
// d[i] = number of palindromes taking s[i] as center
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
  while (0 <= i - k && i + k < n && s[i - k] == s[i + k])
       k++;
  d1[i] = k--;
  if (i + k > r) l = i - k, r = i + k;
}
vector<int> d2(n);
// d[i] = number of palindromes taking s[i-1] and s[i] as
     center
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1)
      ;
  while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s
      [i + k]) k++;
  d2[i] = k--;
  if (i + k > r) l = i - k - 1, r = i + k;
}
```

7.4 Suffix Array

```
vector<VI> c;
VI sort_cyclic_shifts(const string &s) {
  int n = s.size();
  const int alphabet = 256;
  VI p(n), cnt(alphabet, 0);

  c.clear();
  c.emplace_back();
  c[0].resize(n);

  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[0][p[0]] = 0;
  int classes = 1;

  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i - 1]]) classes++;
    c[0][p[i]] = classes - 1;
  }

  VI pn(n), cn(n);
  cnt.resize(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.end(), 0);
    /// radix sort
    for (int i = 0; i < n; i++) cnt[c[h][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[h][pn[i
        ]]]] = pn[i];

    cn[p[0]] = 0;
    classes = 1;

    for (int i = 1; i < n; i++) {
      PII cur = {c[h][p[i]], c[h][(p[i] + (1 << h)) % n
          ]};
      PII prev = {c[h][p[i - 1]], c[h][(p[i - 1] + (1 <<
          h)) % n]};
      if (cur != prev) ++classes;
      cn[p[i]] = classes - 1;
    }
    c.push_back(cn);
  }
  return p;
}
VI suffix_array_construction(string s) {
  s += "!";
  VI sorted_shifts = sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin());
  return sorted_shifts;
}
/// LCP between the ith and jth (i != j) suffix of the
    STRING
int suffixLCP(int i, int j) {
  assert(i != j);
  int log_n = c.size() - 1;

  int ans = 0;
  for (int k = log_n; k >= 0; k--) {
    if (c[k][i] == c[k][j]) {
      ans += 1 << k;
      i += 1 << k;
      j += 1 << k;
    }
  }
  return ans;
}

VI lcp_construction(const string &s, const VI &sa) {
  int n = s.size();
  VI rank(n, 0);
  VI lcp(n - 1, 0);

  for (int i = 0; i < n; i++) rank[sa[i]] = i;

  for (int i = 0, k = 0; i < n; i++, k -= (k != 0)) {
    if (rank[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = sa[rank[i] + 1];
    while (i + k < n && j + k < n && s[i + k] == s[j + k
        ]) k++;
    lcp[rank[i]] = k;
  }
  return lcp;
}
```

7.5 Z Algo

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

8 Equations and Formulas

8.1 Catalan Numbers

$C_n = \frac{1}{n+1}\binom{2n}{n}$ $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$

The number of ways to completely parenthesize n+1 factors.
The number of triangulations of a convex polygon with n+2 sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.
The number of rooted full binary trees with n+1 leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
Number of permutations of $1, , n$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For $n = 3$, these permutations are 132, 213, 231, 312 and 321.

8.2 Stirling Numbers First Kind

The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
$S(n, k)$ counts the number of permutations of n elements with k disjoint cycles.
$S(n, k) = (n-1) \cdot S(n-1, k) + S(n-1, k-1),\ where,\ S(0, 0) = 1, S(n, 0) = S(0, n) = 0$ $\sum_{k=0}^{n} S(n, k) = n!$

The unsigned Stirling numbers may also be defined algebraically, as the coefficient of the rising factorial:

$$x^{\bar{n}} = x(x+1)...(x+n-1) = \sum_{k=0}^{n} S(n, k)x^k$$

Lets $[n, k]$ be the stirling number of the first kind, then

$$\left[n \overset{n}{-} k\right] = \sum_{0 \le i_1 < i_2 < i_k < n} i_1 i_2 .... i_k.$$

8.3 Stirling Numbers Second Kind

Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.
$S(n, k) = k \cdot S(n-1, k) + S(n-1, k-1),\ where\ S(0, 0) = 1, S(n, 0) = S(0, n) = 0$ $S(n, 2) = 2^{n-1} - 1$ $S(n, k) \cdot k! =$ number of ways to color n nodes using colors from 1 to k such that each color is used at least once.
An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n, k)$ and obeys the recurrence relation. $S_r(n+1, k) = kS_r(n, k) + \binom{n}{r-1} S_r(n-r+1, k-1)$
Denote the n objects to partition by the integers $1, 2, ., n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n, k)$, to be the number of ways to partition the integers $1, 2, ., n$ into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i - j| \ge d$. It has been shown that these numbers satisfy, $S^d(n, k) = S(n-d+1, k-d+1), n \ge k \ge d$

8.4 Other Combinatorial Identities

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$$

$$\sum_{i=0}^{k} \binom{n+i}{i} = \sum_{i=0}^{k} \binom{n+i}{n} = \binom{n+k+1}{k}$$

$$n, r \in N, n > r, \sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$$

If $P(n) = \sum_{k=0}^{n} \binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} \cdot P(k)$$

If $P(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cdot P(k)$$

8.5 Different Math Formulas

Picks Theorem : $A = i + b/2 - 1$
Deragements : $d(i) = (i-1) \times (d(i-1) + d(i-2))$

$$\frac{n}{ab} \quad - \quad \left\{\frac{b\prime n}{a}\right\} \quad - \quad \left\{\frac{a\prime n}{b}\right\} \quad + \quad 1$$

8.6 GCD and LCM

if m is any integer, then $\gcd(a + m{\cdot}b, b) = \gcd(a, b)$
The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$.
$\gcd(a, \operatorname{lcm}(b, c)) = \operatorname{lcm}(\gcd(a, b), \gcd(a, c))$.
$\operatorname{lcm}(a, \gcd(b, c)) = \gcd(\operatorname{lcm}(a, b), \operatorname{lcm}(a, c))$.
For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$

$$\gcd(a, b) = \sum_{k|a \text{ and } k|b} \phi(k)$$

$$\sum_{i=1}^{n} [\gcd(i, n) = k] = \phi\left(\frac{n}{k}\right)$$

$$\sum_{k=1}^{n} \gcd(k, n) = \sum_{d|n} d \cdot \phi\left(\frac{n}{d}\right)$$

$$\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi\left(\frac{n}{d}\right)$$

$$\sum_{k=1}^{n} \frac{1}{\gcd(k, n)} = \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{1}{n} \sum_{d|n} d \cdot \phi(d)$$

$$\sum_{k=1}^{n} \frac{k}{\gcd(k, n)} = \frac{n}{2} \cdot \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$$

$$\sum_{k=1}^{n} \frac{n}{\gcd(k, n)} = 2 * \sum_{k=1}^{n} \frac{k}{\gcd(k, n)} - 1, \text{ for } n > 1$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} [\gcd(i, j) = 1] = \sum_{d=1}^{n} \mu(d) \lfloor \frac{n}{d} \rfloor^2$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \gcd(i, j) = \sum_{d=1}^{n} \phi(d) \lfloor \frac{n}{d} \rfloor^2$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} i \cdot j [\gcd(i, j) = 1] = \sum_{i=1}^{n} \phi(i) i^2$$

$$F(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \operatorname{lcm}(i, j) = \sum_{l=1}^{n} \left(\frac{\left(1 + \lfloor \frac{n}{l} \rfloor\right) \left(\lfloor \frac{n}{l} \rfloor\right)}{2} \right)^2 \sum_{d|l} \mu(d) l d$$