Contents

1 Starter 1
2 BIT1D 2
3 BIT2D 2
4 ConvexHullTrick 2
5 LineSegmentTree 3
6 PBDS 4
7 PersistentSegmentTree 4
8 SegTree 5
9 Geometry1 5
10 Geometry2 7
11 Bridge 8
12 Dinitz 9
13 FlowMinCost 10
14 LCA 10
15 MaximumMatching 11
16 Johnson 12
17 Diophantine 13
18 Eratosthenes 13
19 InverseModulo 13
20 MatrixExponentation 13
21 PrimeCheck 14
22 KMP 15
23 Trie-Multiset 15
24 Trie-Set 15
25 BigNum 16

1 Starter

```
#pragma GCC optimize ("O3,unroll-loops,no-stack-protector")
#pragma GCC target ("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")

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

#define           fi    first
#define           se    second
#define           pb    push_back
#define           ep    emplace
#define           eb    emplace_back
#define           lb    lower_bound
#define           ub    upper_bound
#define       all(x)    x.begin(), x.end()
#define      rall(x)    x.rbegin(), x.rend()
#define   uniquev(v)    sort(all(v)), (v).resize(unique(all(v)) - (v).begin())
#define     mem(f,x)    memset(f , x , sizeof(f))
#define        sz(x)    (int32_t)(x).size()
#define  __lcm(a, b)    (1ll * ((a) / __gcd((a), (b))) * (b))
#define           mxx    *max_element
#define           mnn    *min_element
#define          left    Kurumi_Tokisaki
#define         right    Kei_Karuizawa
#define          next    Mai_Sakurajima
#define           div    Yume_Irido
#define          prev    Chizuru_Mizuhara
#define    cntbit(x)    __builtin_popcountll(x)
#define      MASK(x)    ( 1ll << (x) )
#define           Yes    cout << "Yes"
#define           YES    cout << "YES"
#define            No    cout << "No"
#define            NO    cout << "NO"
#define            AA    cout << "Alice"
#define            BB    cout << "Bob"

/// TASK
/// ------------------------------
#ifdef LMQZZZ
void __print(int x) {cerr << x;}
void __print(long x) {cerr << x;}
void __print(long long x) {cerr << x;}
void __print(unsigned x) {cerr << x;}
void __print(unsigned long x) {cerr << x;}
void __print(unsigned long long x) {cerr << x;}
void __print(float x) {cerr << x;}
void __print(double x) {cerr << x;}
void __print(long double x) {cerr << x;}
void __print(char x) {cerr << '\'' << x << '\'';}
void __print(const char *x) {cerr << '\"' << x << '\"';}
void __print(const string &x) {cerr << '\"' << x << '\"';}
void __print(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>
void __print(const pair<T, V> &x) {cerr << '{'; __print(x.first); cerr << ", ";
     __print(x.second); cerr << '}';}
template<typename T>
void __print(const T &x) {int f = 0; cerr << '{'; for (auto &i: x) cerr << (f++
     ? ", " : ""), __print(i); cerr << "}";}
void _print() {cerr << " ]\n";}
template <typename T, typename... V>
void _print(T t, V... v) {__print(t); if (sizeof...(v)) cerr << ", "; _print(v
    ...);}

#define deb(x...) cerr << "[ in " <<__func__<< "() : line " <<__LINE__<< " ] :
    [ " << #x << " ] = [ "; _print(x); cerr << '\n';
#define TASK "C"
#else
#define deb(x...) 3326
#define TASK "lmqzzz"
#endif
///------------------------------

void lmqzzz();
```

```
void init();
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);       cout.tie(0);
  if (fopen(TASK ".inp", "r")) {
    freopen(TASK ".inp", "r", stdin);
    freopen(TASK ".out", "w", stdout);
  }
  /// ================================
          constexpr bool MULTITEST = 0;
  /// ================================
  init();
  int32_t TT = 1;
  if ( MULTITEST ) cin >> TT;
  for(int32_t TTT = 1; TTT <= TT; TTT ++) {
    lmqzzz();
    cout << '\n';
  }
}

template <class T> inline T min(const T &a, const T &b, const T &c) { return
    min(a, min(b, c)); }
template <class T> inline T max(const T &a, const T &b, const T &c) { return
    max(a, max(b, c)); }
template <class T, class U> inline bool mini(T &a, const U &b) { if (a > b) { a
     = b; return 1; } return 0; }
template <class T, class U> inline bool maxi(T &a, const U &b) { if (a < b) { a
     = b; return 1; } return 0; }

constexpr     int16_t dr[]  =  {0, 0, -1, 1};
constexpr     int16_t dc[]  =  {1, -1, 0, 0};
constexpr      int64_t MOD  =  998244353;
constexpr     int32_t MAXN  =  1e6 + 10;
```

2 BIT1D

```
int A[N], B1[N], B2[N], n;
void upd(int* B, int x, int v) {
  for(int i = x ; i <= n ; i += lowbit(i)) B[i] += v;
}
int sum(int* B, int x) {
  int ans = 0;
  for(int i = x ; i > 0 ; i -= lowbit(i)) ans += B[i];
  return ans;
}
void update(int l, int r, int v) {
  upd(B1, r + 1, -v); upd(B1, l, v);
  upd(B2, r + 1, -(r + 1) * v); upd(B2, l, l * v);
}
int query(int l, int r) {
  return ((r + 1) * sum(B1, r) - sum(B2, r)) - (l * sum(B1, l - 1) - sum(B2, l
      - 1));
}
void init() {
  A[0] = 0;
  fill(B1, B1 + n + 1, 0);
  fill(B2, B2 + n + 1, 0);
  for(int i = 1 ; i <= n ; ++i) upd(i, A[i] - A[i - 1]);
}
```

3 BIT2D

```
int n, m, A[N][N], B[N][N][4];
void upd(int x, int y, int v) {
  for(int i = x ; i <= n ; i += lowbit(i)) {
    for(int j = y ; j <= m ; j += lowbit(j)) {
      B[i][j][0] += v;
      B[i][j][1] += x * v;
      B[i][j][2] += y * v;
      B[i][j][3] += x * y * v;
    }
  }
}
int qry(int x, int y) {
  int ans = 0;
  for(int i = x ; i > 0 ; i -= lowbit(i)) {
    for(int j = y ; j > 0 ; j -= lowbit(j)) {
      ans += (x + 1) * (y + 1) * B[i][j][0] - (y + 1) * B[i][j][1] - (x + 1) *
          B[i][j][2] + B[i][j][3];
    }
  }
  return ans;
}
void update(int x1, int y1, int x2, int y2, int v) {
  upd(x1, y1, v);
  upd(x1, y2 + 1, -v);
  upd(x2 + 1, y1, -v);
  upd(x2 + 1, y2 + 1, v);
}
int query(int x1, int y1, int x2, int y2) {
  return qry(x2, y2) - qry(x1 - 1, y2) - qry(x2, y1 - 1) + qry(x1 - 1, y1 - 1);
}
void init() {
  for(int i = 1 ; i <= n ; ++i) {
    for(int j = 1 ; j <= m ; ++j) {
      upd(i, j, A[i][j]);
    }
  }
}
```

4 ConvexHullTrick

```
typedef long long LL;
struct Line {
  LL a, b;
  Line (LL x = 0, LL y = 0) {
    a = x;
```

```
    b = y;
  }
  LL eval (const LL &x) const {
    return a * x + b;
  }

  // first less
  LL intersect (const Line &other) const {
    LL x = (other.b - b) / (a - other.a);
    while (eval(x) <= other.eval(x))
      x--;
    while (eval(x) > other.eval(x))
      x++;
    return x;
  }
};

struct CHT {
  deque<Line> dq;

  LL get_min (LL x) {
    assert(dq.size() >= 1);
    if (dq.size() == 1) {
      return dq[0].eval(x);
    }
    int low = 0, high = dq.size() - 2;
    long long ans = dq[0].eval(x);
    while (low <= high) {
      int mid = (low + high) / 2;
      ans = min(ans, min(dq[mid].eval(x), dq[mid + 1].eval(x)));
      if (dq[mid].eval(x) > dq[mid + 1].eval(x)) {
        low = mid + 1;
      }
      else {
        high = mid - 1;
      }
    }
    return ans;
  }

  bool better (Line last, Line before_last, Line new_line) {
    __int128_t fi = (before_last.b - last.b);
    fi *= (new_line.a - last.a);
    __int128_t se = (last.b - new_line.b);
    se *= (last.a - before_last.a);
    return fi >= se;
  }

  void add_back (LL a, LL b) {
    Line new_line(a, b);
    // dq.back().intersect(dq[dq.size() - 2]) >= new_line.intersect(dq.back())
    while (dq.size() >= 2) {
      Line last = dq.back();
      Line before_last = dq[dq.size() - 2];
      if (better(last, before_last, new_line))
        dq.pop_back();
      else
        break;
    }
    dq.push_back(new_line);
  }

  void add_front (LL a, LL b) {
    Line new_line(a, b);
    while (dq.size() >= 2) {
      Line last = dq[0];
      Line before_last = dq[1];

      if (better(new_line, before_last, last))
        dq.pop_front();
      else
        break;
    }
    dq.push_front(new_line);
  }
};
```

5 LineSegmentTree

```
struct TLine_Segtree {
  struct TLine {
    long long a, b;

    long long operator () (long long x) {
      return a * x + b;
    }
  };

  inline static vector <long long> point;

  TLine_Segtree *cl, *cr;
  long long l, r;
  TLine line;

  TLine_Segtree (int _l = 0, int _r = point.size() - 1) {
    cl = cr = NULL;
    l = point[_l];
    r = point[_r];
    line = {0, INF};

    if (l == r)
      return;

    int _m = (_l + _r) / 2;

    cl = new TLine_Segtree(_l, _m);
    cr = new TLine_Segtree(_m + 1, _r);
  }
```

```
  void update(TLine new_line, long long u, long long v) {
    if (v < l || r < u)
      return;

    if (u <= l && r <= v) {
      bool better_l = (new_line(l) <= line(l)),
           better_r = (new_line(r) <= line(r));

      if (better_l && better_r) {
        line = new_line;
        return;
      }

      if (better_l == false && better_r == false)
        return;
    }

    if (l == r)
      return;

    cl -> update(new_line, u, v);
    cr -> update(new_line, u, v);
  }

  long long get(long long x) {
    if (x < l || r < x)
      return INF;

    if (l == r)
      return line(x);

    return min({line(x),
                cl -> get(x),
                cr -> get(x)});
  }
};
```

6 PBDS

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update

using namespace __gnu_pbds;

typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;

/* k-th: find_by_order() (0-indexed)
   the number of items in a set that are strictly smaller than our item:
       order_of_key()
*/
/*
multiset: less<int> -> less_equal<int>
for searching, lower_bound and upper_bound work oppositely. Also, let's say you
     want to erase x, use s.erase(s.upper_bound(x)) (as upper bound is
    considered as lower bound)
*/

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().
            time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

struct chash {
    int operator()(pair<int, int> x) const { return x.first* 31 + x.second; }
};

gp_hash_table<int, int, custom_hash> mp;
gp_hash_table<pair<int, int>, int, chash> mp1;
```

7 PersistentSegmentTree

```
/**
  * Support: get sum in range, update position
**/

int update (int id, int l, int r, int pos, long long delta) {
  if (pos < l || pos > r)
    return 0;
  if (l == r) {
    n_nodes++;
    st[n_nodes] = Node(0, 0, delta);
    return n_nodes;
  }

  int mid = (l + r) >> 1;
  n_nodes++;
  int cur_id = n_nodes;

  if (pos <= mid) {
    st[cur_id].L = update(st[id].L, l, mid, pos, delta);
```

```
    st[cur_id].R = st[id].R;
  }

  else {
    st[cur_id].L = st[id].L;
    st[cur_id].R = update(st[id].R, mid + 1, r, pos, delta);
  }

  st[cur_id].sum = st[st[cur_id].L].sum + st[st[cur_id].R].sum;
  return cur_id;
}

long long get (int id, int l, int r, int u, int v) {
  if (u > v)
    return 0;
  if (v < l || r < u)
    return 0;
  if (u <= l && r <= v)
    return st[id].sum;
  int mid = (l + r) >> 1;
  return get(st[id].L, l, mid, u, v) + get(st[id].R, mid + 1, r, u, v);
}

// Update:
  ver[i] = update(ver[i - 1], 1, n, pos, delta);
// Get:
  long long ans = get(ver[L], 1, n, u, v) ;
```

8 SegTree

```
struct SegTree {

  /*-------------EDIT HERE--------------*/
  typedef int Type;

  static const Type BASE_VALUE = INT_MIN;

  inline Type f(Type a, Type b) {
    return max(a, b);
  }
  /*------------------------------------*/

  int n;
  vector <Type> st;

  void init(int _n, Type init_value = BASE_VALUE) {
    n = _n + 1;
    st.resize(2 * n, init_value);
  }

  inline void update(int pos, Type value) {
    for (st[pos += n] = value; pos > 1; pos >>= 1)
      st[pos >> 1] = f(st[pos], st[pos ^ 1]);
  }

  inline Type query(int l, int r) {
    r++;

    Type res = BASE_VALUE;

    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1)
        res = f(res, st[l++]);
      if (r & 1)
        res = f(res, st[--r]);
    }

    return res;
  }
};
```

9 Geometry1

```
namespace Geo {

template <class T>
class point_t {
public:
        T x, y;
        point_t(const T& x = 0, const T& y = 0) : x(x), y(y) {}
        template <class T1> operator point_t<T1>() const {
                return point_t<T1>(static_cast<T1>(x), static_cast<T1>(y));
        }
        template <class T1> point_t& operator = (const point_t<T1>& other) {
                x = other.x, y = other.y; return *this;
        }
        template <class T1> point_t& operator += (const point_t<T1>& other) {
                x -= other.x, y -= other.y; return *this;
        }
        template <class T1> point_t& operator -= (const point_t<T1>& other) {
                x += other.x, y += other.y; return *this;
        }
};
template <class F> istream& operator >> (istream& is, point_t<F>& p) {
        return is >> p.x >> p.y;
}
template <class F> ostream& operator << (ostream& os, point_t<F>& p) {
        return os << "[" << p.x << ' ' << p.y << "]";
}
template <class F> point_t<F> makePoint (const F& x, const F& y) {
        return point_t<F>(x, y);
}

#define FUNC1(name, arg, expr) \
template <class F> inline auto name(const arg) -> decltype(expr) {return expr;}
#define FUNC2(name, arg1, arg2, expr) \
template <class F1, class F2> inline auto name(const arg1, const arg2) ->
    decltype(expr) {return expr;}
```

```
#define FUNC3(name, arg1, arg2, arg3, expr) \
template <class F1, class F2, class F3> inline auto name(const arg1, const arg2
    , const arg3) -> decltype(expr) {return expr;}

FUNC1(operator -, point_t<F>& p, makePoint(-p.x, -p.y));
FUNC2(operator +, point_t<F1>& lhs, point_t<F2>& rhs, makePoint(lhs.x + rhs.x,
    lhs.y + rhs.y));
FUNC2(operator -, point_t<F1>& lhs, point_t<F2>& rhs, makePoint(lhs.x - rhs.x,
    lhs.y - rhs.y));
FUNC2(operator *, F1& factor, point_t<F2>& rhs, makePoint(factor * rhs.x,
    factor * rhs.y))
FUNC2(operator *, point_t<F1>& lhs, F2& factor, makePoint(lhs.x * factor, lhs.y
     * factor));
FUNC2(operator /, point_t<F1>& lhs, F2& factor, makePoint(lhs.x / factor, lhs.y
     / factor));

FUNC2(operator *, point_t<F1>& lhs, point_t<F2>& rhs, lhs.x * rhs.x + lhs.y *
    rhs.y);
FUNC2(operator ^, point_t<F1>& lhs, point_t<F2>& rhs, lhs.x * rhs.y - lhs.y *
    rhs.x);

/// < 0 cw, = 0 collinear, > 0 ccw
FUNC2(ccw, point_t<F1>& lhs, point_t<F2>& rhs, lhs ^ rhs);
FUNC3(ccw, point_t<F1>& lhs, point_t<F2>& rhs, point_t<F3>& origin, ccw((lhs -
    origin) ^ (rhs - origin)));

FUNC2(operator ==, point_t<F1>& lhs, point_t<F2>& rhs, lhs.x == rhs.x && lhs.y
    == rhs.y);
FUNC2(operator !=, point_t<F1>& lhs, point_t<F2>& rhs, !(lhs == rhs));

/// distance
FUNC1(fabs, point_t<F>& point, point * point);
FUNC1(norm, point_t<F>& point, sqrtl(fabs(point)));
FUNC2(dist, point_t<F1>& lhs, point_t<F2>& rhs, norm(rhs - lhs));
FUNC2(dist2, point_t<F1>& lhs, point_t<F2>& rhs, fabs(rhs - lhs));

template <class T>
class line_t {
public:
        point_t<T> a, ab;
        line_t (const point_t<T>& a, const point_t<T>& b, bool is2points = 1)
                : a(a), ab(is2points ? b - a : b) {}
        line_t (const T& xa, const T& ya, const T& xb, const T& yb)
                : a(xa, ya), ab(xb - xa, yb - ya) {}
        template <class F1> operator line_t<F1> () const {
                return line_t(point_t<F1>(a), point_t<F1>(ab), false);
        }
        template <class F1> operator = (const line_t<F1>& other) {
                a = other.a, ab = other.ab; return *this;
        }
        operator bool () const {
                return ab != point_t<T>();
        }
        point_t<T> b() const {
                return a + ab;
        }
};
template <class F> line_t<F> makeLine(const point_t<F>& a, const point_t<F>& ab
    , bool is2points = true) {
        return line_t<F>(a, ab, is2points);
}
template <class F1, class F2>
bool onLine (const point_t<F1>& point, const line_t<F2>& line) {
        if (!line) {
                return point == line.a;
        }
        return ccw(point - line.a, line.ab) == 0;
}

template <class F1, class F2>
bool onSegment (const point_t<F1>& point, const line_t<F2>& seg) {
        if (!seg) {
                return point == seg.a;
        }
        auto veca = seg.a - point, vecb = seg.b() - point;
        return (veca ^ vecb) == 0 && (veca * vecb) <= 0;
}

template <class F1, class F2> using distF = decltype(sqrt(F1() + F2()));

template <class F1, class F2>
distF<F1, F2> distLine (const point_t<F1>& point, const line_t<F2>& line) {
        if (!line) {
                return dist(point, line.a);
        }
        return abs((line.a - point) ^ line.ab) / norm(line.ab);
}

template <class F1, class F2>
distF<F1, F2> distSegment (const point_t<F1>& point, const line_t<F2>& seg) {
        if ((point - seg.a) * seg.ab <= 0) {
                return dist(seg.a, point);
        } else if ((seg.b() - point) * seg.ab <= 0) {
                return dist(seg.b(), point);
        } else {
                return distLine(point, seg);
        }
}

template <class F1, class F2, class F3>
void projection (const point_t<F1>& point, const line_t<F2>& line, point_t<F3>&
     res) {
        res = line.a;
        if (line) {
                res += line.ab * static_cast<F3>((point - line.a) * line.ab) /
                    fabs(line.ab);
        }
}
template <class F1, class F2, class F3>
void reflection (const point_t<F1>& point, const line_t<F2>& line, point_t<F3>&
```

```
     res) {
        projection (point, line, res);
        res *= 2;
        res -= point;
}

template <int TYPE> struct EndpointChecker {};
template <> struct EndpointChecker<0> { /// ray
        template <class F> bool operator ()(const F& a, const F& b) const {
                return true;
        }
};
template <> struct EndpointChecker<1> { /// []
        template <class F> bool operator ()(const F& a, const F& b) const {
                return a <= b;
        };
};
template <> struct EndpointChecker<2> { /// ()
        template <class F> bool operator ()(const F& a, const F& b) const {
                return a < b;
        };
};

template <int LA, int LB, int RA, int RB, class F1, class F2, class F3>
bool intersect (const line_t<F1>& lhs, const line_t<F2>& rhs, point_t<F3>& res)
     {
        assert(lhs && rhs && "line not exist!");
        /// (A + AB * i - C) ^ CD == 0
        /// A ^ CD + AB ^ CD * i - C ^ CD = 0
        /// AB ^ CD * i = (C - A) ^ CD
        /// i = (C - A) ^ CD / AB ^ CD
        deb(lhs.ab.x, lhs.ab.y, rhs.ab.x, rhs.ab.y);
        auto s = lhs.ab ^ rhs.ab;
        if (s == 0) return false;
        auto ls = (rhs.a - lhs.a) ^ rhs.ab;
        auto rs = (rhs.a - lhs.a) ^ lhs.ab;
        if (s < 0) s = -s, ls = -ls, rs = -rs;
        bool intersect = EndpointChecker<LA>()(decltype(ls)(0), ls)
                      && EndpointChecker<LB>()(ls, s)
                      && EndpointChecker<RA>()(decltype(rs)(0), rs)
                      && EndpointChecker<RB>()(rs, s);
        if (intersect) {
                res = lhs.a + lhs.ab * static_cast<F3>(ls) / s;
        }
        return intersect;
}

}

using namespace Geo;
```

10 Geometry2

```
//get_dist, get_area, get_equation, get_intersection can't have Type be long
    long

typedef long double Type;

const int DIR_LEFT = -1,
          DIR_STRAIGHT = 0,
          DIR_RIGHT = 1,
          LINES_INTERSECT = -1,
          LINES_PARALLEL = 0,
          LINES_COINCIDE = 1;

struct Point {
  Type x, y;

  Point() {}

  Point(Type _x, Type _y) {
    x = _x;
    y = _y;
  }
};

struct Line {
  Point A, B;

  Line() {}

  Line(Point _A, Point _B) {
    A = _A;
    B = _B;
  }
};

Point operator + (Point A, Point B) {
  return Point(A.x + B.x, A.y + B.y);
}

Point operator - (Point A, Point B) {
  return Point(A.x - B.x, A.y - B.y);
}

Point operator * (Type k, Point A) {
  return Point(k * A.x, k * A.y);
}

Type operator * (Point A, Point B) {
  return A.x * B.y - A.y * B.x;
}

int get_dir(Point A, Point B, Point C) {
  Type dir = (B - A) * (C - B);

  if (abs(dir + 0.0) <= 1e-9)
    return DIR_STRAIGHT;
```

```
  if (dir > 0)
    return DIR_LEFT;

  return DIR_RIGHT;
}

Type get_dist_sq(Point A, Point B) {
  return (A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y);
}

Type get_dist(Point A, Point B) {
  return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
}

void get_line_equation(Line l, Type &a, Type &b, Type &c) {
  Point A = l.A,
        B = l.B;

  a = B.y - A.y;
  b = A.x - B.x;
  c = -A.x * (B.y - A.y) + A.y * (B.x - A.x);
}

int get_intersection(Line l1, Line l2, Point &res) {
  Type a1, b1, c1, a2, b2, c2, D, Dx, Dy;

  get_line_equation(l1, a1, b1, c1);
  get_line_equation(l2, a2, b2, c2);

  D = a1 * b2 - a2 * b1;
  Dx = b1 * c2 - b2 * c1;
  Dy = c1 * a2 - c2 * a1;

  if (D != 0) {
    res.x = Dx / D;
    res.y = Dy / D;

    return LINES_INTERSECT;
  }

  if (c1 == c2)
    return LINES_COINCIDE;

  return LINES_PARALLEL;
}

Type get_dist(Line l, Point A) {
  Type a, b, c;
  get_line_equation(l, a, b, c);

  return abs(a * A.x + b * A.y + c) / sqrt(a * a + b * b);
}

Type get_area_2(Point A, Point B, Point C) {
  return abs((B - A) * (C - A));
}

Type get_area(Point A, Point B, Point C) {
  return abs((B - A) * (C - A)) / 2;
}

bool is_in_line(Line l, Point A) {
  if (get_dir(l.A, l.B, A) != DIR_STRAIGHT)
    return false;

  if (A.x < min(l.A.x, l.B.x) || max(l.A.x, l.B.x) < A.x)
    return false;

  if (A.y < min(l.A.y, l.B.y) || max(l.A.y, l.B.y) < A.y)
    return false;

  return true;
}

void convex_hull() {
  sort(a + 1, a + n + 1, [] (TPoint a, TPoint b) {
    if (a.y != b.y)
      return (a.y < b.y);

    return (a.x < b.x);
  });

  int n_up = 1,
      n_down = 1;

  up[1] = down[1] = a[1];

  for (int i = 2; i <= n; i++) {
    while (n_up >= 2) {
      if (dir(up[n_up - 1], up[n_up], a[i]) != LEFT)
        break;

      n_up--;
    }

    while (n_down >= 2) {
      if (dir(down[n_down - 1], down[n_down], a[i]) != RIGHT)
        break;

      n_down--;
    }

    up[++n_up] = down[++n_down] = a[i];
  }
}
```

11 Bridge

```
/**
```

```
  Counting bridges in multi-graph.
  adj[u] = {v, i} - node and id of edge.
**/

void dfs(int u){
  long long s = 0, s2 = 0;

  low[u] = num[u] = ++timeDfs;
  child[u] = 1;

  for(auto [v, i] : g[u]){
    if(used[i])
      continue;

    used[i] = true;

    if(!num[v]){
      tr[v] = u;
      dfs(v);
      low[u] = min(low[u], low[v]);
      child[u] += child[v];

      if(low[v] >= num[v]){
        // current edge is bridge
      }

      if(low[v] >= num[u]){
        // u is joint
      }
    }
    else
      low[u] = min(low[u], num[v]);
  }

}

void solve(){
  for(int i = 1; i <= n; i++)
    if(num[i] == 0){
      dfs(i);
    }
}
```

12 Dinitz

```
struct Dinitz {
  struct Edge {
    int u, v, cap, flow;
    Edge (int a = 0, int b = 0, int c = 0, int d = 0) {
      u = a,
      v = b;
      cap = c;
      flow = d;
    }
};

int n, s, t;
vector <vector <int>> g;
vector <Edge> edges;
vector <int> d, ptr;

Dinitz (int _n = 0, int _s = 0, int _t = 0) {
  n = _n;
  s = _s;
  t = _t;

  g.assign(n + 5, vector <int>());
  d.assign(n + 5, -1);
  ptr.assign(n + 5, 0);
}

void add_edge (int u, int v, int w) {
  g[u].emplace_back(edges.size());
  edges.emplace_back(u, v, w, 0);
  g[v].emplace_back(edges.size());
  edges.emplace_back(v, u, 0, 0);
}

bool bfs() {
  d.assign(n + 5, -1);
  ptr.assign(n + 5, 0);

  d[s] = 0;
  queue <int> qu;
  qu.emplace(s);

  while (!qu.empty()) {
    int u = qu.front();
    qu.pop();

    if (u == t) return true;

    for (int id : g[u]) {
      if (edges[id].flow >= edges[id].cap)
        continue;
      int v = edges[id].v;
      if (d[v] != -1)
        continue;
      d[v] = d[u] + 1;
      qu.emplace(v);
    }
  }

  return d[t] != -1;
}

int dfs (int u, int pushed) {
  if (u == t)
    return pushed;
```

```
    if (!pushed)
      return 0;

    for (int &id = ptr[u]; id < g[u].size(); id++) {
      int edge_id = g[u][id];
      int v = edges[edge_id].v;
      if (d[v] != d[u] + 1 || edges[edge_id].flow >= edges[edge_id].cap)
        continue;
      int x = dfs(v, min(pushed, edges[edge_id].cap - edges[edge_id].flow));
      if (x) {
        edges[edge_id].flow += x;
        edges[edge_id ^ 1].flow -= x;
        return x;
      }
    }

    return 0;
  }

  int find_flow() {
    int flow = 0;
    while (bfs()) {
      while (true) {
        int x = dfs(s, 1e9);
        if (!x) break;
        flow += x;
      }
    }
    return flow;
  }
};
```

13 FlowMinCost

```
/*
Bai toan nguoi dua thu trung hoa co huong

Dinh thua: dinh co degin < degout
Dinh thieu: dinh co degin > degout

Thuat toan: tim duong di tu 1 dinh thua -> 1 dinh thieu, nhan doi cac canh tren
     duong di

Luong mincost:
Them 2 dinh s t
Noi s -> cac dinh thua:
    suc chua = luong thua
    trong so = 0
Noi cac dinh thieu -> t:
    suc chua = luong thieu
    trong so = 0
Cac canh co san:
    suc chua = + vo cung
    trong so = trong so da cho

=> tim luong min cost
*/

long long get_wp(TEdge edge) {
  int u = edge.u,
      v = edge.v,
      w = edge.w;

  return w + p[u] - p[v];
}

void ijk(int s, int t, int n) {
  fill(d + 1, d + n + 1, INF);

  priority_queue <TPQ_Item> pq;

  d[s] = 0;
  pq.push({s, 0});

  while (pq.empty() == false) {
    TPQ_Item item = pq.top();
    pq.pop();

    if (item.valid() == false)
      continue;

    int u = item.u;

    for (int i : adj[u]) {
      int v = edges[i].v;

      if (get_cf(edges[i]) == 0)
        continue;

      if (minimize(d[v], d[u] + get_wp(edges[i])))
        trace[v] = i,
        pq.push({v, d[v]});
    }
  }

  for (int u = 1; u <= n; u++)
    p[u] += d[u];
}
```

14 LCA

```
int euler[MAXN * 4];
int l[MAXN * 4][20];
int d[MAXN];
int pos[MAXN];
int par[MAXN];

////////////////////////////////////////////////////////////////////
```

```
/// in main :
d[1] = 0;
int N = 0;

function<void(int, int)> dfslca = [&](int u, int p) {
  par[u] = p;
  euler[++N] = u;
  for (const int &v : adj[u]) {
    if (v == p) continue;
    d[v] = d[u] + 1;
    dfslca(v, u);
    euler[++N] = u;
  }
};

dfslca(1, 1);

for (int i = 1; i <= N; i ++) {
  pos[euler[i]] = i;
}

for (int i = 1; i <= N; i ++) {
  l[i][0] = euler[i];
}

int lim = log2(N);
for (int i = 1; i <= lim; i ++) {
  for (int j = 1; j + MASK(i) - 1 <= N; j ++) {
    l[j][i] = (d[l[j][i - 1]] <= d[l[j + MASK(i - 1)][i - 1]]) ? l[j][i - 1] :
        l[j + MASK(i - 1)][i - 1];
  }
}

function<int(int, int)> lca = [=](int u, int v) {
  u = pos[u];
  v = pos[v];
  if(u > v) {
    u ^= v;
    v ^= u;
    u ^= v;
  }
  int lg = log2(v - u + 1);
  return (d[l[u][lg]] <= d[l[v - MASK(lg) + 1][lg]])
         ? l[u][lg] : l[v - MASK(lg) + 1][lg];
};
```

15 MaximumMatching

```
class maximumMatching_t {
private:
  bool bfs() {
    queue<int> q;
    for (int i = 1; i <= n_left; i ++) {
      if (mat[i] == 0) {
        q.ep(i);
        d[i] = 0;
      } else {
        d[i] = 1e9;
      }
    }
    d[0] = 1e9;

    while (sz(q)) {
      int u = q.front(); q.pop();
      if (d[u] >= d[0]) {
        continue;
      }

      for (int v : adj[u]) {
        if (mini(d[rmat[v]], d[u] + 1)) {
          q.ep(rmat[v]);
        }
      }
    }

    return d[0] != 1e9;
  }

  bool bpm(const int& u) {
    if (!u) {
      return 1;
    }

    for (int v : adj[u]) {
      if (d[rmat[v]] == d[u] + 1) {
        if (bpm(rmat[v])) {
          mat[u] = v;
          rmat[v] = u;
          return 1;
        }
      }
    }
    d[u] = 1e9;
    return 0;
  }

  void dfs(const int& u, const bool& _) {
    vis[u][_] = 1;
    for (int v : g[u][_]) {
      if (vis[v][_ ^ 1] == 0) {
        dfs(v, _ ^ 1);
      }
    }
  }

public:
  static constexpr int NMAX = | + 12;
  int mat[NMAX], rmat[NMAX], d[NMAX];
```

```
bool vis[MAXN][2];
int n_left, n_right;
vector<int> adj[NMAX];
vector<int> g[NMAX][2];

maximumMatching_t(int n_left = 0, int n_right = 0) : n_left(n_left), n_right(
    n_right) {
  for (int i = 0; i <= n_left; i ++) {
    vector<int>().swap(adj[i]);
  }
}

void addEdge(const int& u, const int& v) {
  adj[u].eb(v);
}

int findMaximumMatching() {
  memset(mat, 0, (n_left + 1) * sizeof(int));
  memset(rmat, 0, (n_right + 1) * sizeof(int));

  int res = 0;

  while (bfs()) {
    for (int i = 1; i <= n_left; i ++) {
      if (mat[i] == 0) {
        res += bpm(i);
      }
    }
  }

  return res;
}

vector<pair<int, bool> > minimumVertexCover(const bool& inverse = 0) {
  for (int i = 1; i <= n_left; i ++) {
    vector<int>().swap(g[i][0]);
    vis[i][0] = 0;
  }
  for (int i = 1; i <= n_right; i ++) {
    vector<int>().swap(g[i][1]);
    vis[i][1] = 0;
  }

  /// if it's a matching: edge from right->left
  /// otherwise: edge from left->right

  for (int i = 1; i <= n_left; i ++) {
    for (int v : adj[i]) {
      if (v == mat[i]) {
        g[v][1].eb(i);
      } else {
        g[i][0].eb(v);
      }
    }
  }

  for (int i = 1; i <= n_left; i ++) {
    if (mat[i] == 0) {
      dfs(i, 0);
    }
  }

  vector<pair<int, bool> > ans;

  for (int i = 1; i <= n_left; i ++) {
    if (vis[i][0] == inverse) {
      ans.eb(i, 0);
    }
  }

  for (int i = 1; i <= n_right; i ++) {
    if (vis[i][1] != inverse) {
      ans.eb(i, 1);
    }
  }

  return ans;
 }
};
```

16 Johnson

```
/**
Co N chi tiet may can duoc gia cong lan luot tren hai may A va B. Thoi gian gia
     cong chi tiet i tren may A la a_i, thoi gian gia cong tren may B la b_i.
Yeu cau: Hay tim trinh tu gia cong cac chi tiet tren hai may sao cho viec hoan
    thanh gia cong tat ca cac chi tiet la som nhat co the.
**/
struct job{
  int x, y, pos;
} a[N], b[N];

int tmp[N], timeA, timeB;

void do_job(job *v, int bound){
  for(int i = 1; i <= bound; i++){
    timeA += v[i].x;
    timeB = max(timeB, timeA) + v[i].y;
  }
}

int main(){
  int n;
  cin >> n;

  for(int i = 1; i <= n; i++)
    cin >> tmp[i];

  int n1 = 0, n2 = 0;
```

```
  for(int i = 1, y; i <= n; i++){
    cin >> y;
    if(tmp[i] <= y)
      a[++n1] = {tmp[i], y, i};
    else
      b[++n2] = {tmp[i], y, i};
  }

  sort(a + 1, a + 1 + n1, [](const job &x1, const job &x2){
    return x1.x < x2.x;
  });
  sort(b + 1, b + 1 + n2, [](const job &x1, const job &x2){
    return x1.y > x2.y;
  });

  do_job(a, n1);
  do_job(b, n2);

  cout << timeB << '\n';
  for(int i = 1; i <= n1; i++)
    cout << a[i].pos << ' ';
  for(int i = 1; i <= n2; i++)
    cout << b[i].pos << ' ';
}
```

17 Diophantine

```
// ax + by = gcd(a, b)
int extended_euclid (int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }

  int x1, y1;
  int d = extended_euclid(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}

// ax + by = c
bool find_solution (int a, int b, int c, int &x0, int &y0) {
  if (a == 0) {
    if (b == 0) {
      if (c != 0) return false;
      x0 = 0, y0 = 0;
      return true;
    }
    if (c % b) {
      return false;
    }
    x0 = 0;
    y0 = -c/b;
    return true;
  }
  if (b == 0) {
    if (c % a)
      return false;
    x0 = -c/a;
    y0 = 0;
    return true;
  }
  int g = extended_euclid(abs(a), abs(b), x0, y0);
  if (c % g)
    return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
```

18 Eratosthenes

```
int pm[MAXN + 10];
vector<int> prime;

void eratosthenes(int N) {
  for (int i = 2 ; i <= N ; i ++) {
    if (pm[i] == 0) {
      pm[i] = i;
      prime.eb(i);
    }
    for (int j = 0 ; (j < sz(prime)) && (prime[j] <= pm[i]) && (i * prime[j] <=
         N) ; j ++) {
      pm[i * prime[j]] = prime[j];
    }
  }
}
```

19 InverseModulo

```
gcd(a, m) = 1
a^-1 = a^phi(m)-2
inv[1] = 1;
for(int i = 2; i < m; ++i)
    inv[i] = m - (m/i) * inv[m%i] % m;
```

20 MatrixExponentation

```
namespace Matrix_Exponentiation {

  const int MAX_ROW = |; // Change Max_row here
```

```
const int MAX_COL = |; // Change Max_col here
int64_t mod = 1e9 + 7; // Change MOD here
int64_t mxmod = (int64_t)(7e18 / mod) * mod;

void change_mod(int _mod) {
  mod = _mod;
  mxmod = (int64_t)(7e18 / mod) * mod;
}

int64_t multi(int64_t a, int64_t b) {
  int64_t ret = 0;
  for(int i = 0 ; MASK(i) <= b ; i ++, a = (a + a) % mod) {
    if(MASK(i) & b) ret = (ret + a) % mod;
  }
  return ret;
}

struct Matrix {
  int r,c;
  int64_t a[MAX_ROW][MAX_COL];
  void Resize(int _r,int _c) {
    for (int i = 0; i < r; i ++) {
      for (int j = 0; j < c; j ++) {
              a[i][j] = 0;
      }
    }
  }

  auto & operator [] (int i) { return a[i]; }

  const auto & operator[] (int i) const { return a[i]; }

  Matrix operator *(const Matrix& other) {
    Matrix product, tmp;
    product.Resize(r, other.c);
    tmp.Resize(r, other.c);
    for (int i = 0; i < product.r; i ++) {
      for (int j = 0; j < c; j ++) {
        for (int k = 0; k < product.c; k ++) {
          //product[i][k] += multi(a[i][j] , other.a[j][k]);
          tmp[i][k] += a[i][j] * other[j][k];
          if(tmp[i][k] >= mxmod)
                  tmp[i][k] -= mxmod;
        }
      }
    }
    for (int i = 0; i < product.r; i ++) {
      for (int j = 0; j < product.c; j ++) {
              product[i][j] = tmp[i][j] % mod;
      }
    }
    return product;
  }

  void operator *= (const Matrix& other) {
    *this = *this * other;
  }

  Matrix operator ^ (const int64_t& b) {
    Matrix ret;
    Matrix m = *this;
    ret.Resize(m.r, m.c);
    for (int i = 0; i < ret.r; i ++) {
            ret[i][i] = 1;
    }
    for(int i = 0 ; MASK(i) <= b ; i ++, m *= m) {
      if (b & MASK(i)) {
        ret*=m;
      }
    }
    return ret;
  }

  void operator ^= (const int64_t& b) {
    *this = *this ^ b;
  }

  friend ostream& operator << (ostream& os, const Matrix& M) {
    for (int i = 0; i < M.r; i ++) {
      for (int j = 0; j < M.c; j ++) {
        os << M.a[i][j] << " \n"[j == M.c - 1];
      }
    }
    return os;
  }
};
}

using namespace Matrix_Exponentiation;
```

21 PrimeCheck

```
bool composite(LL n, int a, LL d, int s) {
  LL x = power(a, d, n);

  if (x == 1 || x == n - 1)
    return false;

  for (int r = 1; r <= s - 1; r++) {
    x = multiply(x, x, n);

    if (x == n - 1)
      return false;
  }

  return true;
}

  if (n < 5)
```

```
    return (n == 2 || n == 3 ? "YES" : "NO");

  int s = 0;
  long long d = n - 1;

  while (d % 2 == 0)
    s++,
    d /= 2;

  vector <int> prime = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

  for (int p : prime) {
    if (p == n)
      return "YES";

    if (p > n - 2)
      return "YES";

    if (composite(n, p, d, s))
      return "NO";
  }

  return "YES";
```

22 KMP

```
vector <int> kmp (const string &s) {
  vector <int> pi(s.size(), 0);
  pi[0] = 0;
  for (int i = 1; i < (int)s.size(); i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j])
      j = pi[j - 1];
    if (s[i] == s[j])
      pi[i] = j + 1;
  }
  return pi;
}
```

23 Trie-Multiset

```
class trie_t {
public:
  trie_t* g[2];
  int cnt;

  trie_t() : g{nullptr, nullptr}, cnt(0) {}

  static void add (trie_t* root, int num, int d) {
    for (int i = 15; i >= 0; i --) {
      const int b = num >> i & 1;
      if (root->g[b] == nullptr) {
        root->g[b] = new trie_t();
      }
      root = root->g[b];
      root->cnt += d;
    }
  }

  static int findk (trie_t* root, int k) {
    int res = 0;
    for (int i = 15; i >= 0; i --) {
      if (root->g[0] && root->g[0]->cnt >= k) {
        root = root->g[0];
      } else {
        if (root->g[0]) {
          k -= root->g[0]->cnt;
        }
        root = root->g[1];
        res |= MASK(i);
      }
    }
    return res;
  }
};
```

24 Trie-Set

```
class trie_t {
public:
  trie_t* g[2];
  int cnt;
  static constexpr int lim = 30;

  trie_t() : g{nullptr, nullptr}, cnt(0) {}

  static bool add (trie_t* root, int num, int bit = lim) {
    if (bit == -1) {
      return 0;
    }
    const int b = num >> bit & 1;
    if (root->g[b] == nullptr) {
      root->g[b] = new trie_t();
      add(root->g[b], num, bit - 1);
      root->g[b]->cnt ++;
      return 1;
    } else if (add(root->g[b], num, bit - 1)) {
      root->g[b]->cnt ++;
      return 1;
    } else {
      return 0;
    }
  }

  static bool rmv (trie_t* root, int num, int bit = lim) {
    if (bit == -1) {
      return 1;
```

```
    }
    const int b = num >> bit & 1;
    if (root->g[b] == nullptr) {
      return 0;
    } else if (rmv(root->g[b], num, bit - 1)) {
      root->g[b]->cnt --;
      if (root->g[b]->cnt == 0) {
        root->g[b] = nullptr;
      }
      return 1;
    } else {
      return 0;
    }
  }

  static int findk (trie_t* root, int k) {
    int res = 0;
    for (int i = lim; i >= 0; i --) {
      if (root->g[0] && root->g[0]->cnt >= k) {
        root = root->g[0];
      } else {
        if (root->g[0]) {
          k -= root->g[0]->cnt;
        }
        root = root->g[1];
        res |= MASK(i);
      }
    }
    return res;
  }

  static int countx (trie_t* root, int num) {
    int res = 0;
    for (int i = lim; i >= 0; i --) {
      if (num >= MASK(i)) {
        if (root->g[0]) {
          res += root->g[0]->cnt;
        }
        num -= MASK(i);
        if (root->g[1]) {
          root = root->g[1];
        } else {
          return res;
        }
      } else {
        if (root->g[0]) {
          root = root->g[0];
        } else {
          return res;
        }
      }
    }
    return res;
  }
};
```

25 BigNum

```
const int base = 1000000000;
const int base_digits = 9;
struct bigint {
        vector<int> a;
        int sign;
        int size(){
                if(a.empty())return 0;
                int ans=(a.size()-1)*base_digits;
                int ca=a.back();
                while(ca)
                        ans++,ca/=10;
                return ans;
        }
        bigint operator ^(const bigint &v){
                bigint ans=1,a=*this,b=v;
                while(!b.isZero()){
                        if(b%2)
                                ans*=a;
                        a*=a,b/=2;
                }
                return ans;
        }
        string to_string(){
                stringstream ss;
                ss << *this;
                string s;
                ss >> s;
                return s;
        }
        int sumof(){
                string s = to_string();
                int ans = 0;
                for(auto c : s)  ans += c - '0';
                return ans;
        }
        bigint() :
                sign(1) {
        }

        bigint(long long v) {
                *this = v;
        }

        bigint(const string &s) {
                read(s);
        }

        void operator=(const bigint &v) {
                sign = v.sign;
                a = v.a;
        }
```

```
void operator=(long long v) {
        sign = 1;
        a.clear();
        if (v < 0)
                sign = -1, v = -v;
        for (; v > 0; v = v / base)
                a.push_back(v % base);
}

bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
                bigint res = v;

                for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.
                    size()) || carry; ++i) {
                        if (i == (int) res.a.size())
                                res.a.push_back(0);
                        res.a[i] += carry + (i < (int) a.size() ? a[i]
                            : 0);
                        carry = res.a[i] >= base;
                        if (carry)
                                res.a[i] -= base;
                }
                return res;
        }
        return *this - (-v);
}

bigint operator-(const bigint &v) const {
        if (sign == v.sign) {
                if (abs() >= v.abs()) {
                        bigint res = *this;
                        for (int i = 0, carry = 0; i < (int) v.a.size()
                             || carry; ++i) {
                                res.a[i] -= carry + (i < (int) v.a.size
                                    () ? v.a[i] : 0);
                                carry = res.a[i] < 0;
                                if (carry)
                                        res.a[i] += base;
                        }
                        res.trim();
                        return res;
                }
                return -(v - *this);
        }
        return *this + (-v);
}

void operator*=(int v) {
        if (v < 0)
                sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
                if (i == (int) a.size())
                        a.push_back(0);
                long long cur = a[i] * (long long) v + carry;
                carry = (int) (cur / base);
                a[i] = (int) (cur % base);
                //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur)
                    , "c"(base));
        }
        trim();
}

bigint operator*(int v) const {
        bigint res = *this;
        res *= v;
        return res;
}

void operator*=(long long v) {
        if (v < 0)
                sign = -sign, v = -v;
        if(v > base){
                *this = *this * (v / base) * base + *this * (v % base);
                return ;
        }
        for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
                if (i == (int) a.size())
                        a.push_back(0);
                long long cur = a[i] * (long long) v + carry;
                carry = (int) (cur / base);
                a[i] = (int) (cur % base);
        }
        trim();
}

bigint operator*(long long v) const {
        bigint res = *this;
        res *= v;
        return res;
}

friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1)
    {
        int norm = base / (b1.a.back() + 1);
        bigint a = a1.abs() * norm;
        bigint b = b1.abs() * norm;
        bigint q, r;
        q.a.resize(a.a.size());

        for (int i = a.a.size() - 1; i >= 0; i--) {
                r *= base;
                r += a.a[i];
                int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()
                    ];
                int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.
                    size() - 1];
                int d = ((long long) base * s1 + s2) / b.a.back();
                r -= b * d;
                while (r < 0)
```

```
                        r += b, --d;
                q.a[i] = d;
        }

        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return make_pair(q, r / norm);
}

bigint operator/(const bigint &v) const {
        return divmod(*this, v).first;
}

bigint operator%(const bigint &v) const {
        return divmod(*this, v).second;
}

void operator/=(int v) {
        if (v < 0)
                sign = -sign, v = -v;
        for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
                long long cur = a[i] + rem * (long long) base;
                a[i] = (int) (cur / v);
                rem = (int) (cur % v);
        }
        trim();
}

bigint operator/(int v) const {
        bigint res = *this;
        res /= v;
        return res;
}

int operator%(int v) const {
        if (v < 0)
                v = -v;
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)
                m = (a[i] + m * (long long) base) % v;
        return m * sign;
}

void operator+=(const bigint &v) {
        *this = *this + v;
}
void operator-=(const bigint &v) {
        *this = *this - v;
}
void operator*=(const bigint &v) {
        *this = *this * v;
}
void operator/=(const bigint &v) {
        *this = *this / v;
}

bool operator<(const bigint &v) const {
        if (sign != v.sign)
                return sign < v.sign;
        if (a.size() != v.a.size())
                return a.size() * sign < v.a.size() * v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
                if (a[i] != v.a[i])
                        return a[i] * sign < v.a[i] * sign;
        return false;
}

bool operator>(const bigint &v) const {
        return v < *this;
}
bool operator<=(const bigint &v) const {
        return !(v < *this);
}
bool operator>=(const bigint &v) const {
        return !(*this < v);
}
bool operator==(const bigint &v) const {
        return !(*this < v) && !(v < *this);
}
bool operator!=(const bigint &v) const {
        return *this < v || v < *this;
}

void trim() {
        while (!a.empty() && !a.back())
                a.pop_back();
        if (a.empty())
                sign = 1;
}

bool isZero() const {
        return a.empty() || (a.size() == 1 && !a[0]);
}

bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
}

bigint abs() const {
        bigint res = *this;
        res.sign *= res.sign;
        return res;
}

long long longValue() const {
        long long res = 0;
```

```
        for (int i = a.size() - 1; i >= 0; i--)
                res = res * base + a[i];
        return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b) {
        return b.isZero() ? a : gcd(b, a % b);
}
friend bigint lcm(const bigint &a, const bigint &b) {
        return a / gcd(a, b) * b;
}

void read(const string &s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')
            ) {
                if (s[pos] == '-')
                        sign = -sign;
                ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= base_digits) {
                int x = 0;
                for (int j = max(pos, i - base_digits + 1); j <= i; j
                    ++)
                        x = x * 10 + s[j] - '0';
                a.push_back(x);
        }
        trim();
}

friend istream& operator>>(istream &stream, bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
}

friend ostream& operator<<(ostream &stream, const bigint &v) {
        if (v.sign == -1)
                stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >= 0; --i)
                stream << setw(base_digits) << setfill('0') << v.a[i];
        return stream;
}

static vector<int> convert_base(const vector<int> &a, int old_digits,
    int new_digits) {
        vector<long long> p(max(old_digits, new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < (int) p.size(); i++)
                p[i] = p[i - 1] * 10;
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int i = 0; i < (int) a.size(); i++) {
                cur += a[i] * p[cur_digits];
                cur_digits += old_digits;
                while (cur_digits >= new_digits) {
                        res.push_back(int(cur % p[new_digits]));
                        cur /= p[new_digits];
                        cur_digits -= new_digits;
                }
        }
        res.push_back((int) cur);
        while (!res.empty() && !res.back())
                res.pop_back();
        return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
                for (int i = 0; i < n; i++)
                        for (int j = 0; j < n; j++)
                                res[i + j] += a[i] * b[j];
                return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++)
                a2[i] += a1[i];
        for (int i = 0; i < k; i++)
                b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int) a1b1.size(); i++)
                r[i] -= a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
                r[i] -= a2b2[i];

        for (int i = 0; i < (int) r.size(); i++)
                res[i + k] += r[i];
        for (int i = 0; i < (int) a1b1.size(); i++)
                res[i] += a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
                res[i + n] += a2b2[i];
```

```
        return res;
    }

    bigint operator*(const bigint &v) const {
        vector<int> a6 = convert_base(this->a, base_digits, 6);
        vector<int> b6 = convert_base(v.a, base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
            a.push_back(0);
        while (b.size() < a.size())
            b.push_back(0);
        while (a.size() & (a.size() - 1))
            a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int) (cur % 1000000));
            carry = (int) (cur / 1000000);
        }
        res.a = convert_base(res.a, 6, base_digits);
        res.trim();
        return res;
    }
};
```