Team notebook

Takik Hasan

July 8, 2020

Contents

1 Data Structure

1.1 All purpose Segment Tree

```
/** All purpose segment tree */

/**
 * Source Inspiration: KACTL github
 * Description: Segment tree with ability to add or set values
      of large intervals, and compute max of intervals.
 * Can be changed to other things.
 * Use with a bump allocator for better performance, and
      SmallPtr or implicit indices to save memory.
 * Time: O(\log N).
 * Usage: Node* tr = new Node(v, 0, sz(v));
 * Range(L, R) means range (L to R-1) inclusive
 */

/*
    #pragma once

    #define size_0 (300 << 19) // 150 MEGABYTES
    #define size_1 (350 << 19) // 175 MEGABYTES -> Safe with
         moderate capacity
    #define size_2 (400 << 19) // 200 MEGABYTES -> Safe with
         huge capacity
    #define size_3 (450 << 19) // 225 MEGABYTES

    static char buf[size_1];
    void* operator new(size_t s) {
        static size_t i = sizeof buf;
        assert(s < i);
        return (void*)&buf[i -= s];
    }
    void operator delete(void*) {}
*/

LL f(LL a, LL b)
{
    return max(a, b); /// CHANGE IF NEEDED
}

const int OSUM = 1, OMIN = 2, OMAX = 3, OXOR = 4, OOR = 5, OAND
     = 6, OGCD = 7, OLCM = 8;

LL f2(LL x, LL len, int OPTION)
{
    /*
        f2 function should work wonderfully for any kind of
             tree's SET operation,
```

```
        and for SUM, MIN, MAX type tree's ADD operation, but
             for other types - doubtful (maybe improved later).
    */
    if (OPTION == OSUM) {
        return len * x;
    }
    else if (OPTION == OXOR) {
        if (len % 2 == 0) return 0;
        else return x;
    }
    else {
        return x;
    }
}

const LL SUM = 0, MIN = LLONG_MAX, MAX = LLONG_MIN, XOR = 0, OR
     = 0, GCD = 0, LCM = 1;
const LL AND = (1LL << 60) - 1; /// 60 on bits

struct Node {
        Node *l = 0, *r = 0;
        int lo, hi;
        LL mset = inf, madd = 0, val = MAX; /// CHANGE IF NEEDED
        Node(int lo, int hi) : lo(lo), hi(hi) {} // Large
             interval of MAX
        Node(vector<LL>& v, int lo, int hi) : lo(lo), hi(hi) {
                if (lo + 1 < hi) {
                        int mid = lo + (hi - lo)/2;
                        l = new Node(v, lo, mid); r = new Node(v,
                             mid, hi);
                        val = f(l->val, r->val);
                }
                else val = v[lo];
        }
        LL query(int L, int R) {
                if (R <= lo || hi <= L) return MAX; /// CHANGE
                     IF NEEDED
                if (L <= lo && hi <= R) return val;
                push();
                return f(l->query(L, R), r->query(L, R));
        }
        void set(int L, int R, LL x) {
                if (R <= lo || hi <= L) return;
                if (L <= lo && hi <= R) {
                    mset = x, madd = 0;
                    val = f2(x, hi - lo, OMAX); /// CHANGE IF NEEDED
                }
                else {
                        push(), l->set(L, R, x), r->set(L, R, x);
                        val = f(l->val, r->val);
                }
        }
        void add(int L, int R, LL x) {
                if (R <= lo || hi <= L) return;
                if (L <= lo && hi <= R) {
                        if (mset != inf) mset += x;
                        else madd += x;
                        val += f2(x, hi - lo, OMAX); /// CHANGE
                             IF NEEDED
                }
                else {
                        push(), l->add(L, R, x), r->add(L, R, x);
                        val = f(l->val, r->val);
                }
        }
        void push() {
                if (!l) {
                        int mid = lo + (hi - lo)/2;
                        l = new Node(lo, mid); r = new Node(mid,
                             hi);
                }
                if (mset != inf)
                        l->set(lo,hi,mset), r->set(lo,hi,mset),
                             mset = inf;
                else if (madd)
                        l->add(lo,hi,madd), r->add(lo,hi,madd),
                             madd = 0;
        }
};

/** All purpose segment tree */
```

1.2 LCA Extended Upgraded

```
/*
    #1. T1 -> Data type required for query
    #2. T2 -> Data type required for edge weight
    #3. Introduced two data types for memory efficiency
*/
template<typename T1, typename T2>
struct LCA
{
    typedef pair<T2, int> ii;
    const int MIN = 1, MAX = 2, SUM = 3;
    /* 0 indexed */
    int n;
    vector<int> l;
    vector<vector<int>> p;
    vector<vector<T1>> val;
    int OPTION;
    T1 OPTION_VAL;
    /* parent, val, option */
    int LOG;
    vector<int> logs, powr;
    vector<vector<ii>> g;
    LCA(int n, int option) : n(n), l(n), g(n), OPTION(option)
    {
        LOG = log2(n) + 2;
        OPTION_VAL = (OPTION == SUM) ? 0 : ((OPTION == MIN) ?
             numeric_limits<T1>::max() :
             numeric_limits<T1>::min());
    }
    LCA(int n, vector<vector<ii>> g, int option) : n(n), l(n),
         g(g), OPTION(option)
    {
        LOG = log2(n) + 2;
        OPTION_VAL = (OPTION == SUM) ? 0 : ((OPTION == MIN) ?
             numeric_limits<T1>::max() :
             numeric_limits<T1>::min());
    }
    void addEdge(int u, int v, T2 w)
    {
        g[u].pb(mp(v, w));
        g[v].pb(mp(u, w));
    }
    void dfs(int u, int pp, int ll)
    {
        l[u] = ll++;
        p[u][0] = pp;
        for (ii x : g[u]) {
            if (x.ff != pp) {
                val[x.ff][0] = x.ss;
                dfs(x.ff, u, ll);
            }
        }
    }
    T1 func(T1 x, T2 y)
    {
        if (OPTION == SUM) return x + y;
        else if (OPTION == MIN) return min(x, y);
        else if (OPTION == MAX) return max(x, y);
    }
    void build()
    {
        p = vector<vector<int>> (n, vector<int> (LOG));
        val = vector<vector<T1>> (n, vector<T1> (LOG));
        dfs(0, -1, 0);
        for (int i = 1; i < LOG; i++) {
            for (int j = 0; j < n; j++) {
                if (p[j][i-1] != -1) {
                    val[j][i] = func(val[j][i-1],
                         val[p[j][i-1]][i-1]);
                    p[j][i] = p[p[j][i-1]][i-1];
                }
                else {
                    p[j][i] = -1;
                    val[j][i] = OPTION_VAL;
                }
            }
        }
    }
    int lca(int u, int v)
    {
        if (l[u] > l[v]) swap(u, v);
        int d = l[v] - l[u];
        for (int i = 0; i < LOG; i++) {
            if (CHECK(d, i)) {
                v = p[v][i];
            }
        }
        if (u == v) return u;
        for (int i = LOG - 1; i >= 0; i--) {
            if (p[u][i] != p[v][i]) {
                u = p[u][i];
                v = p[v][i];
            }
```

```
    }
    return p[u][0];
}
T1 query(int u, int v)
{
    T1 ret = OPTION_VAL;
    if (l[u] > l[v]) swap(u, v);
    int d = l[v] - l[u];
    for (int i = 0; i < LOG; i++) {
        if (CHECK(d, i)) {
            ret = func(ret, val[v][i]);
            v = p[v][i];
        }
    }
    if (u == v) return ret;
    for (int i = LOG - 1; i >= 0; i--) {
        if (p[u][i] != p[v][i]) {
            ret = func(ret, val[u][i]);
            ret = func(ret, val[v][i]);
            u = p[u][i];
            v = p[v][i];
        }
    }
    ret = func(ret, val[u][0]);
    ret = func(ret, val[v][0]);
    return ret;
}
/*
    To use array queries:

        Initialize for an array a[n]:
            LCA<T1, T2> lca(n + 1); // T1 -> Data type for
                 Query, T2 -> Data type for array element
            for (int i = 0; i < n; i++) {
                lca.addEdge(i, i + 1, a[i]);
            }
            lca.build();
            lca.fast_g(); // Only for O(1) queries

        For query(l, r) where l and r are 0-indexed but any
              kind of query
        including O(logn) and O(1) ones:
            O(logn):
                cout << lca.query(l, r + 1) << endl;
            O(1):
                cout << lca.fast_query(l, r + 1) << endl;

*/
void fast_g()
{
    logs.resize(n); powr.resize(LOG);
    logs[1] = 0;   powr[0] = 1;
    for (int i = 1; i < LOG; i++) powr[i] = powr[i-1] * 2;
    for (int i = 2; i < n ; i++) logs[i] = logs[i/2] + 1;
}
T1 fast_query(int l, int r)
{
    int j = logs[r-l];
    return func(val[r][j], val[l+powr[j]][j]);
}
```

```
};
```

1.3 Ordered set

```
// requires c++11 or higher

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

int main()
{
    ordered_set<int> s;
    s.insert(1);
    s.insert(12);
    s.insert(123);
    s.insert(1234);
    s.erase(123);
    cout << s.size() << endl;
    cout << s.order_of_key(1234) << endl; // how many numbers
         less than 1234
    cout << *s.find_by_order(1) << endl; // the value at index
         1 (0 indexed)
    cout << (end(s) == s.find_by_order(100)) << endl;

    return 0;
}
```

1.4 Segment Tree Variation - Leftmost or Rightmost index with Min or Max Value

```
/** All purpose segment tree - VARIATION */

/**
 * Source Inspiration: KACTL github
 * Description: Segment tree with ability to add or set values
       of large intervals, and compute max of intervals.
 * Can be changed to other things.
 * Use with a bump allocator for better performance, and
       SmallPtr or implicit indices to save memory.
 * Time: O(\log N).
 * Usage: Node* tr = new Node(v, 0, sz(v));
 * Range(L, R) means range (L to R-1) inclusive
 */

/**
    IMPORANT!!!
    Segment Tree Variation Description:
        Tree type: MIN tree
        Updates: Range or point SET or ADD operation
        Query: For any range, the min value and the last index
              with that value
               Can be changed to find first index and/or to find
                    max value

    Problems (easy to hard) - (role model submission / problem
         page):
        i.
              https://codeforces.com/contest/1208/submission/83386421
              (direct template)
*/

/*
    #pragma once

    #define size_0 (300 << 19) // 150 MEGABYTES
    #define size_1 (350 << 19) // 175 MEGABYTES -> Safe with
         moderate capacity
    #define size_2 (400 << 19) // 200 MEGABYTES -> Safe with
         huge capacity
    #define size_3 (450 << 19) // 225 MEGABYTES

    static char buf[size_1];
    void* operator new(size_t s) {
        static size_t i = sizeof buf;
        assert(s < i);
        return (void*)&buf[i -= s];
    }
    void operator delete(void*) {}
*/

LL f(LL a, LL b)
{
    return min(a, b); /// CHANGE IF NEEDED
}

const int OSUM = 1, OMIN = 2, OMAX = 3, OXOR = 4, OOR = 5, OAND
     = 6, OGCD = 7, OLCM = 8;

LL f2(LL x, LL len, int OPTION)
{
    /*
        f2 function should work wonderfully for any kind of
             tree's SET operation,
        and for SUM, MIN, MAX type tree's ADD operation, but
             for other types - doubtful (maybe improved later).
    */
    if (OPTION == OSUM) {
        return len * x;
    }
    else if (OPTION == OXOR) {
        if (len % 2 == 0) return 0;
        else return x;
    }
    else {
        return x;
    }
}
```

```
const LL SUM = 0, MIN = LLONG_MAX, MAX = LLONG_MIN, XOR = 0, OR
     = 0, GCD = 0, LCM = 1;
const LL AND = (1LL << 60) - 1; /// 60 on bits

struct Node {
       Node *l = 0, *r = 0;
       int lo, hi;
       LL mset = inf, madd = 0, val = MIN; /// CHANGE IF NEEDED
       Node(int lo, int hi) : lo(lo), hi(hi) {} // Large
            interval of MAX
       Node(vector<LL>& v, int lo, int hi) : lo(lo), hi(hi) {
              if (lo + 1 < hi) {
                     int mid = lo + (hi - lo)/2;
                     l = new Node(v, lo, mid); r = new Node(v,
                          mid, hi);
                     val = f(l->val, r->val);
              }
              else val = v[lo];
       }
       void getSegments(vector<Node*> &valid_segments, int L,
            int R) {
              if (R <= lo || hi <= L) return;
              if (L <= lo && hi <= R) {
           valid_segments.pb(this); return;
              }
              push();
              r->getSegments(valid_segments, L, R),
                   l->getSegments(valid_segments, L, R);
       }
    int innerQuery() {
       if (lo + 1 == hi) return lo;
       push();
       if (r->val <= l->val) return r->innerQuery();
       else return l->innerQuery();
    }
    pair<LL, int> query(int L, int R) {
       /* Complexity : logn + logn + logn (getting segments,
            iterating over segments & process one segment) */
       /* This function should only be called from root */
       vector<Node*> valid_segments;
       getSegments(valid_segments, L, R);
       /* Now we have at most logn segments */
       /* The segments do not overlap and they are stored from
            right to left */
       LL minn = MIN;
       Node *seg;
       for (auto node : valid_segments) {
           if (node->val < minn) {
              minn = node->val;
              seg = node;
           }
       }
       return {minn, seg->innerQuery()};
    }
       void set(int L, int R, LL x) {
              if (R <= lo || hi <= L) return;
              if (L <= lo && hi <= R) {
           mset = x, madd = 0;
           val = f2(x, hi - lo, OMIN); /// CHANGE IF NEEDED
              }
              else {
                     push(), l->set(L, R, x), r->set(L, R, x);
                     val = f(l->val, r->val);
              }
       }
       void add(int L, int R, LL x) {
              if (R <= lo || hi <= L) return;
              if (L <= lo && hi <= R) {
                     if (mset != inf) mset += x;
                     else madd += x;
                     val += f2(x, hi - lo, OMIN); /// CHANGE
                          IF NEEDED
              }
              else {
                     push(), l->add(L, R, x), r->add(L, R, x);
                     val = f(l->val, r->val);
              }
       }
       void push() {
              if (!l) {
                     int mid = lo + (hi - lo)/2;
                     l = new Node(lo, mid); r = new Node(mid,
                          hi);
              }
              if (mset != inf)
                     l->set(lo,hi,mset), r->set(lo,hi,mset),
                          mset = inf;
              else if (madd)
                     l->add(lo,hi,madd), r->add(lo,hi,madd),
                          madd = 0;
       }
};

/** All purpose segment tree - VARIATION */
```

1.5 Segment Tree Variation - LIS LDS all Types

```
/** All purpose segment tree */

/**
 * Source Inspiration: KACTL github
 * Description: Segment tree with ability to add or set values
      of large intervals, and compute max of intervals.
 * Can be changed to other things.
 * Use with a bump allocator for better performance, and
      SmallPtr or implicit indices to save memory.
 * Time: O(\log N).
 * Usage: Node* tr = new Node(v, 0, sz(v));
 * Range(L, R) means range (L to R-1) inclusive
 */

/*
   #pragma once
    #define size_0 (300 << 19) // 150 MEGABYTES
    #define size_1 (350 << 19) // 175 MEGABYTES -> Safe with
         moderate capacity
    #define size_2 (400 << 19) // 200 MEGABYTES -> Safe with
         huge capacity
    #define size_3 (450 << 19) // 225 MEGABYTES

    static char buf[size_1];
    void* operator new(size_t s) {
       static size_t i = sizeof buf;
       assert(s < i);
       return (void*)&buf[i -= s];
    }
    void operator delete(void*) {}
*/

LL f(LL a, LL b)
{
    return max(a, b); /// CHANGE IF NEEDED
}

const int OSUM = 1, OMIN = 2, OMAX = 3, OXOR = 4, OOR = 5, OAND
     = 6, OGCD = 7, OLCM = 8;

LL f2(LL x, LL len, int OPTION)
{
    /*
       f2 function should work wonderfully for any kind of
            tree's SET operation,
       and for SUM, MIN, MAX type tree's ADD operation, but
            for other types - doubtful (maybe improved later).
    */
    if (OPTION == OSUM) {
       return len * x;
    }
    else if (OPTION == OXOR) {
       if (len % 2 == 0) return 0;
       else return x;
    }
    else {
       return x;
    }
}

const LL SUM = 0, MIN = LLONG_MAX, MAX = LLONG_MIN, XOR = 0, OR
     = 0, GCD = 0, LCM = 1;
const LL AND = (1LL << 60) - 1; /// 60 on bits

struct Node {
       Node *l = 0, *r = 0;
       int lo, hi;
       LL mset = inf, madd = 0, val = MAX; /// CHANGE IF NEEDED
       Node(int lo, int hi) : lo(lo), hi(hi) {} // Large
            interval of MAX
       Node(vector<LL>& v, int lo, int hi) : lo(lo), hi(hi) {
              if (lo + 1 < hi) {
                     int mid = lo + (hi - lo)/2;
                     l = new Node(v, lo, mid); r = new Node(v,
                          mid, hi);
                     val = f(l->val, r->val);
```

```
            }
            else val = v[lo];
        }
        LL query(int L, int R) {
            if (R <= lo || hi <= L) return MAX; /// CHANGE
                IF NEEDED
            if (L <= lo && hi <= R) return val;
            push();
            return f(l->query(L, R), r->query(L, R));
        }
        void set(int L, int R, LL x) {
            if (R <= lo || hi <= L) return;
            if (L <= lo && hi <= R) {
               mset = x, madd = 0;
               val = f2(x, hi - lo, OMAX); /// CHANGE IF NEEDED
            }
            else {
                push(), l->set(L, R, x), r->set(L, R, x);
                val = f(l->val, r->val);
            }
        }
        void add(int L, int R, LL x) {
            if (R <= lo || hi <= L) return;
            if (L <= lo && hi <= R) {
                if (mset != inf) mset += x;
                else madd += x;
                val += f2(x, hi - lo, OMAX); /// CHANGE
                    IF NEEDED
            }
            else {
                push(), l->add(L, R, x), r->add(L, R, x);
                val = f(l->val, r->val);
            }
        }
        void push() {
            if (!l) {
                int mid = lo + (hi - lo)/2;
                l = new Node(lo, mid); r = new Node(mid,
                    hi);
            }
            if (mset != inf)
                l->set(lo,hi,mset), r->set(lo,hi,mset),
                    mset = inf;
            else if (madd)
                l->add(lo,hi,madd), r->add(lo,hi,madd),
                    madd = 0;
        }
};

/** All purpose segment tree */

template<typename T>
vector<int> lis(vector<T> v, bool strict)
{
    vector<int> ret(SZ(v)); /* longest
        increasing/non-decreasing subsequence ending at index
        i */
    T minn = numeric_limits<T>::max(), maxx =
        numeric_limits<T>::min();
    for (T x : v) minn = min(minn, x), maxx = max(maxx, x);
    Node *root = new Node(minn, maxx + 1);
    for (int i = 0; i < SZ(v); i++) {
        int val = root->query(minn, v[i] + !strict);
        ret[i] = max(1, val + 1);
        root->set(v[i], v[i] + 1, ret[i]);
    }
    return ret;
}

template<typename T>
vector<int> lds(vector<T> v, bool strict)
{
    vector<int> ret(SZ(v)); /* longest
        decreasing/non-increasing subsequence ending at index
        i */
    T minn = numeric_limits<T>::max(), maxx =
        numeric_limits<T>::min();
    for (T x : v) minn = min(minn, x), maxx = max(maxx, x);
    Node *root = new Node(minn, maxx + 1);
    for (int i = 0; i < SZ(v); i++) {
        int val = root->query(v[i] + strict, maxx + 1);
        ret[i] = max(1, val + 1);
        root->set(v[i], v[i] + 1, ret[i]);
    }
    return ret;
}

template<typename T>
vector<int> lis_reverse(vector<T> v, bool strict)
{
    /* longest increasing/non-decreasing subsequence starting
        at index i */
    reverse(all(v));
    auto ret = lds(v, strict);
    reverse(all(ret));
    return ret;
}

template<typename T>
vector<int> lds_reverse(vector<T> v, bool strict)
{
    /* longest decreasing/non-increasing subsequence starting
        at index i */
    reverse(all(v));
    auto ret = lis(v, strict);
    reverse(all(ret));
    return ret;
}
```

1.6 Segment Tree Variation - Maximum Sum Subarray

```
/** All purpose segment tree - VARIATION */

/**
 * Source Inspiration: KACTL github
 * Description: Segment tree with ability to add or set values
      of large intervals, and compute max of intervals.
 * Can be changed to other things.
 * Use with a bump allocator for better performance, and
      SmallPtr or implicit indices to save memory.
 * Time: O(\log N).
 * Usage: Node* tr = new Node(v, 0, sz(v));
 * Range(L, R) means range (L to R-1) inclusive
 */

/**
   IMPORANT!!!
   Segment Tree Variation Description:
      Tree type: MAX tree with four significant values for
           each node (not straightforward MAX tree)
      Updates: Range or point SET operation, point ADD
           operation
      Query: For any range, maximum sub-array sum (picking at
           least one element)

   Problems (easy to hard) - (role model submission / problem
        page):
      i. https://vjudge.net/solution/26173596 (direct
           template)
*/

/*
   #pragma once

   #define size_0 (300 << 19) // 150 MEGABYTES
   #define size_1 (350 << 19) // 175 MEGABYTES -> Safe with
        moderate capacity
   #define size_2 (400 << 19) // 200 MEGABYTES -> Safe with
        huge capacity
   #define size_3 (450 << 19) // 225 MEGABYTES

   static char buf[size_1];
   void* operator new(size_t s) {
      static size_t i = sizeof buf;
      assert(s < i);
      return (void*)&buf[i -= s];
   }
   void operator delete(void*) {}
*/

const LL SUM = 0, MIN = LLONG_MAX, MAX = LLONG_MIN, XOR = 0, OR
     = 0, GCD = 0, LCM = 1;
const LL AND = (1LL << 60) - 1; /// 60 on bits

struct NodeValue
{
   LL sum = MAX, right_aligned = MAX, left_aligned = MAX,
        max_sum = MAX;
};

inline NodeValue f(const NodeValue &a, const NodeValue &b)
{
   if (a.max_sum == MAX) return b;
   else if (b.max_sum == MAX) return a;
```

```
    else return {a.sum + b.sum, max(b.right_aligned, b.sum +
         a.right_aligned), max(a.left_aligned, a.sum +
         b.left_aligned), max(max(a.max_sum, b.max_sum),
         a.right_aligned + b.left_aligned)}; /// CHANGE IF
         NEEDED
}

const int OSUM = 1, OMIN = 2, OMAX = 3, OXOR = 4, OOR = 5, OAND
     = 6, OGCD = 7, OLCM = 8;

NodeValue f2(LL x, LL len, int OPTION)
{
    /*
       f2 function should work wonderfully for any kind of
            tree's SET operation,
       and for SUM, MIN, MAX type tree's ADD operation, but
            for other types - doubtful (maybe improved later).
    */
    LL aligned = (x < 0) ? x : x * len;
    return {x * len, aligned, aligned, aligned};
}

struct Node {
       Node *l = 0, *r = 0;
       int lo, hi;
       LL mset = inf, madd = 0;
       NodeValue val; /// CHANGE IF NEEDED
       Node(int lo, int hi) : lo(lo), hi(hi) {} /// Large
            interval of MAX
       Node(vector<LL>& v, int lo, int hi) : lo(lo), hi(hi) {
              if (lo + 1 < hi) {
                     int mid = lo + (hi - lo)/2;
                     l = new Node(v, lo, mid); r = new Node(v,
                          mid, hi);
                     val = f(l->val, r->val);
              }
              else val = {v[lo], v[lo], v[lo], v[lo]};
       }
       NodeValue query(int L, int R) {
              if (R <= lo || hi <= L) return NodeValue(); ///
                   CHANGE IF NEEDED
              if (L <= lo && hi <= R) return val;
              push();
              return f(l->query(L, R), r->query(L, R));
       }
       void set(int L, int R, LL x) {
              if (R <= lo || hi <= L) return;
              if (L <= lo && hi <= R) {
                  mset = x, madd = 0;
                  val = f2(x, hi - lo, OMAX); /// CHANGE IF NEEDED
              }
              else {
                     push(), l->set(L, R, x), r->set(L, R, x);
                     val = f(l->val, r->val);
              }
       }
       void add(int L, int R, LL x) { /** We call add() for
            point add only */
              if (R <= lo || hi <= L) return;
              if (L <= lo && hi <= R) {
                     if (mset != inf) mset += x;
                     else madd += x;
                     val.sum += x, val.right_aligned += x,
                          val.left_aligned += x, val.max_sum
                          += x; /// CHANGE IF NEEDED
              }
              else {
                     push(), l->add(L, R, x), r->add(L, R, x);
                     val = f(l->val, r->val);
              }
       }
       void push() {
              if (!l) {
                     int mid = lo + (hi - lo)/2;
                     l = new Node(lo, mid); r = new Node(mid,
                          hi);
              }
              if (mset != inf)
                     l->set(lo,hi,mset), r->set(lo,hi,mset),
                          mset = inf;
              else if (madd)
                     l->add(lo,hi,madd), r->add(lo,hi,madd),
                          madd = 0;
       }
};

/** All purpose segment tree - VARIATION */
```

1.7 Union Find Disjoint Set

```
/*
    #1. Elements are numbered from 0 to (n - 1) inclusive
    #2. Source: Competitive Programming 1 (Steven Halim)
*/
struct UnionFindDisjointSet
{
    vector<int> pset;
    UnionFindDisjointSet(int n) {
        pset.resize(n);
        for (int i = 0; i < n; i++) pset[i] = i;
    }
    int findSet(int i) {
        return (pset[i] == i) ? i : (pset[i] =
             findSet(pset[i]));
    }
    void unionSet(int i, int j) {
        pset[findSet(i)] = findSet(j);
    }
    bool isSameSet(int i, int j) {
        return findSet(i) == findSet(j);
    }
};
```

2 Dynamic Programming

2.1 Knuth Optimization for DP

```
/* Knuth Optimization for DP */
/*
    Source Inspiration:
         https://www.quora.com/What-is-Knuths-optimization-in-dynamic-pr
    Template directly solves:
         https://www.spoj.com/problems/BRKSTRNG/
    (My source code for the problem is on vjudge)
*/
/* For "Matrix Chain Multiplication(MCM)" type problems - not
     the actual MCM AFAIK */
/* Converts O(n^3) to O(n^2) */
/* In this implementation, every range is inclusive */
/* 0 indexed */
/* T -> data type required for storing result */
/* Usage: KnuthDP<LL>(n) -> n: Array length */


/// depends on problem - start (to help determine costs)
vector<LL> sum;
/// depends on problem - end

template<typename T>
inline T KnuthDP(int n)
{
    vector<vector<T>> res(n, vector<T> (n));
    vector<vector<int>> mid(n, vector<int> (n));
    for (int len = 1; len <= n; len++) {
        for (int r = len - 1; r < n; r++) {
            int l = r - len + 1;
            if (len == 1) {
                res[l][r] = 0; /// depends on problem
                mid[l][r] = l;
                continue;
            }
            int mLeft = mid[l][r-1];
            int mRight = mid[l+1][r];
            res[l][r] = numeric_limits<T>::max(); /// depends on
                 problem
            for (int m = mLeft; m <= mRight; m++) {
                if (m == r) continue; /* for len == 2 */
                /// depends on problem - start
                    T tres = res[l][m] + res[m+1][r];
                    T cost = sum[r];
                    if (l) cost -= sum[l-1];
                    tres += cost;
                    if (tres < res[l][r]) {
                        res[l][r] = tres;
                        mid[l][r] = m;
                    }
                /// depends on problem - end
            }
        }
    }
    return res[0][n-1];
}
```

```
/* Knuth Optimization for DP */
```

2.2 Matrix

```
/**  Matrix  */

/*
    #1. Problems (easy to hard) - (role model submission /
          problem page):
        i.  https://vjudge.net/solution/26176019
*/
template<typename T>
struct Matrix
{
    vector<vector<T>> matrix;

    Matrix() {}

    Matrix(int n, int m)
    {
        matrix = vector<vector<T>> (n, vector<T> (m));
    }

    Matrix operator * (const Matrix& B) const
    {
        int n = SZ(matrix), m = SZ(B[0]), c = SZ(matrix[0]);
        Matrix ret(n, m);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                /**
                 * Initializing ret[i][j] with correct value
                       (and dimension when T is Matrix).
                 * (%) operator should ignore (Matrix<T> %
                       (int/LL/ULL..)) type operation.
                 * Using (%) operator here is necessary if mod
                       value is given.
                 */
                ret[i][j] = matrix[i][0] * B[0][j];
                for (int k = 1; k < c; k++) {
                    ret[i][j] = ret[i][j] + matrix[i][k] *
                          B[k][j];
                }
            }
        }
        return ret;
    }

    Matrix operator + (const Matrix& B) const
    {
        int n = SZ(matrix), m = SZ(B[0]);
        Matrix ret(n, m);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                /**
                 * (%) operator should ignore (Matrix<T> %
                       (int/LL/ULL..)) type operation.
                 * Using (%) operator here also is necessary if
                       mod value is given.
                 */
                ret[i][j] = matrix[i][j] + B[i][j];
            }
        }
        return ret;
    }

    template<typename P>
    Matrix pow(P p) {
        if (p == 1) return *this;
        Matrix ret = pow(p / 2);
        ret = ret * ret;
        if (p % 2) ret = ret * (*this);
        return ret;
    }

    template<typename N>
    Matrix& operator % (const N ignore)
    {
        return *this;
    }

    vector<T>& operator [] (int u) { return matrix[u]; }
    const vector<T>& operator [] (int u) const { return
          matrix[u]; }
};

template<typename T>
struct Solver
{
    /**
     * T -> Data type required for storing result
     */
    template<typename N>
    T nthFib(N n, T x, T y) const
    {
        /**
         * x -> fib(0), y -> fib(1)
         */
        if (!n) return 0;
        Matrix<T> a(2, 2), b(2, 1);
        a[0][1] = a[1][0] = a[1][1] = 1;
        b[0][0] = x; b[1][0] = y;
        return (a.pow(n) * b)[0][0];
    }
};

/**  Matrix  */
```

2.3 Maximum sum sub-(parallelepiped or rectangular prism or cube)

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

// Source: modified version of maximum sum sub-rectangle from
     Competitive Programming 1 (by Steven Halim)
// Complexity: O(n^6) (total loops = n^6 + n^3)
// This code is for a parallelepiped / rectangular prism / cube
     / 3D rectangle where a, b & c are dimensions
// Make slight changes for minimum result

int main()
{
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    long long int arr[a][b][c];
    for (int i = 0; i < a; i++) {
        for (int j = 0; j < b; j++) {
            for (int k = 0; k < c; k++) {
                scanf("%lld", &arr[i][j][k]);
                if (i) arr[i][j][k] += arr[i-1][j][k];
                if (j) arr[i][j][k] += arr[i][j-1][k];
                if (k) arr[i][j][k] += arr[i][j][k-1];
                if (i && j) arr[i][j][k] -= arr[i-1][j-1][k];
                if (i && k) arr[i][j][k] -= arr[i-1][j][k-1];
                if (j && k) arr[i][j][k] -= arr[i][j-1][k-1];
                if (i && j && k) arr[i][j][k] +=
                      arr[i-1][j-1][k-1];
            }
        }
    }
    long long int maxSubRect = LONG_LONG_MIN;
    for (int i = 0; i < a; i++) {
        for (int j = 0; j < b; j++) {
            for (int k = 0; k < c; k++) {
                for (int l = i; l < a; l++) {
                    for (int m = j; m < b; m++) {
                        for (int n = k; n < c; n++) {
                            long long int subRect = arr[l][m][n];
                            if (i) subRect -= arr[i-1][m][n];
                            if (j) subRect -= arr[l][j-1][n];
                            if (k) subRect -= arr[l][m][k-1];
                            if (i && j) subRect +=
                                  arr[i-1][j-1][n];
                            if (i && k) subRect +=
                                  arr[i-1][m][k-1];
                            if (j && k) subRect +=
                                  arr[l][j-1][k-1];
                            if (i && j && k) subRect -=
                                  arr[i-1][j-1][k-1];
                            maxSubRect = max(maxSubRect, subRect);
                        }
                    }
                }
            }
        }
    }
    printf("%lld\n", maxSubRect);

    return 0;
}
```

2.4 Maximum sum sub-rectangle

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

// Source: Competitive Programming 1 (by Steven Halim)
// Complexity: O(n^4) (total loops = n^4 + n^2)
// This code is for a square rectangle where n is the length of
     each side
// Make slight changes for a rectangle of any shape and / or
     minimum sub-rectangle

int main()
{
    int n;
    scanf("%d", &n);
    int arr[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &arr[i][j]);
            if (i) arr[i][j] += arr[i-1][j];
            if (j) arr[i][j] += arr[i][j-1];
            if (i && j) arr[i][j] -= arr[i-1][j-1];
        }
    }
    int maxSubRect = -127 * 100 * 100;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = i; k < n; k++) {
                for (int l = j; l < n; l++) {
                    int subRect = arr[k][l];
                    if (i) subRect -= arr[i-1][l];
                    if (j) subRect -= arr[k][j-1];
                    if (i && j) subRect += arr[i-1][j-1];
                    maxSubRect = max(maxSubRect, subRect);
                }
            }
        }
    }
    printf("%d\n", maxSubRect);

    return 0;
}
```

3 Geometry

3.1 Geometry

```
FYI:
    acos(x), asin(x) etc. functions always return radian value.
    For cos(theta), sin(theta) etc. functions theta has to be
         in radians.
    2 radians = 360 degrees
    Degree to radian:
        1 degree = (1 / 180) radians
        x degrees = x * (1 / 180) radians
    Radian to Degree:
        1 radian = 180 degrees
        x radians = x * 180 degrees

Constants:
    PI = acos(-1)

Projectile:
    For a projectile launched at an angle,
        H = (u * u * sin(angle) * sin(angle)) / (2 * g)
        R = (u * u * sin(2 * angle)) / g
        T = (2 * u * sin(angle)) / g
    where H = maximum height, R = travelled distance, T = time
         of flight,
    g = free fall acceleration, angle = launch angle in radians
    It takes T / 2 time to reach the max height and T / 2 time
         to reach zero height
    from the maximum height.

Area:
    Area covered by three points (ax, ay), (bx, by) & (cx, cy):
        Area = abs((ax * (by - cy) + bx * (cy - ay) + cx * (ay
             - by)) / 2)
```

3.2 Geometry$_{t}emplate$

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

typedef long long int LL;
typedef long double LD;

int dx[] = {0, 0, -1, +1};
int dy[] = {+1, -1, 0, 0};


//********.........GEOMETRY.........********

//My edits: double -> LD
//2 lines in halfPlaneIntersection(...)
//all(vector<...>) to (v.begin(), v.end()) ->
     MinimumEnclosingCircle(3), PolygonStubbing(1)
//Point to PT in GetLineABC(...)
//UNVERIFIED CW and CCW sorting section at the end
//(Should work if we can see every corner from the inside from
     the center of the minimum enclosing circle)
//PolygonPolygonIntersection(...) function

#define M_PI acos(-1.0)
#define EPS 1e-12
#define NEX(x) ((x+1)%n)
#define PRV(x) ((x-1+n)%n)
#define RAD(x) ((x*M_PI)/180)
#define DEG(x) ((x*180)/M_PI)
#define mp make_pair
const LD PI=acos(-1.0);

inline int dcmp (LD x) { return x < -EPS ? -1 : (x > EPS); }
//inline int dcmp (int x) { return (x>0) - (x<0); }

class PT {
public:
    LD x, y;
    PT() {}
    PT(LD x, LD y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y) {}
    LD Magnitude() {return sqrt(x*x+y*y);}

    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); }
    bool operator > (const PT& u) const { return u < *this; }
    bool operator <= (const PT& u) const { return *this < u ||
         *this == u; }
    bool operator >= (const PT& u) const { return *this > u ||
         *this == u; }
    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 LD u) const { return PT(x * u, y * u);
         }
    PT operator / (const LD u) const { return PT(x / u, y / u);
         }
    LD operator * (const PT& u) const { return x*u.y - y*u.x; }
};

LD dot(PT p, PT q)   { return p.x*q.x+p.y*q.y; }
LD dist2(PT p, PT q) { return dot(p-q,p-q); }
LD dist(PT p, PT q) { return sqrt(dist2(p,q)); }
LD cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }

LD myAsin(LD val) {
    if(val>1) return PI*0.5;
    if(val<-1) return -PI*0.5;
    return asin(val);
}

LD myAcos(LD val) {
    if(val>1) return 0;
    if(val<-1) return PI;
    return acos(val);
}

ostream &operator<<(ostream &os, const PT &p) {
    os << "(" << p.x << "," << p.y << ")";
}

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

```
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p)  { return PT(p.y,-p.x); }

PT RotateCCW(PT p,LD t) {
    return PT(p.x*cos(t)-p.y*sin(t),p.x*sin(t)+p.y*cos(t));
}

PT RotateAroundPointCCW(PT p,PT pivot,LD t) {
    PT trans=p-pivot;
    PT ret=RotateCCW(trans,t);
    ret=ret+pivot;
    return ret;
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
    return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

LD DistancePointLine(PT a,PT b,PT c) {
    return dist(c,ProjectPointLine(a,b,c));
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
    LD r = dot(b-a,b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance from c to segment between a and b
LD DistancePointSegment(PT a, PT b, PT c) {
    return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
LD DistancePointPlane(LD x, LD y, LD z,
                      LD a, LD b, LD c, LD d)
{
    return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or
    collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
    return dcmp(cross(b-a, c-d)) == 0;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
    return LinesParallel(a, b, c, d)
      && dcmp(cross(a-b, a-c)) == 0
      && dcmp(cross(c-d, c-a)) == 0;
}

//UNTESTED CODE SEGMENT
```

```
// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
    if (LinesCollinear(a, b, c, d)) {
        if (dcmp(dist2(a, c)) == 0 || dcmp(dist2(a, d)) == 0 ||
            dcmp(dist2(b, c)) == 0 || dcmp(dist2(b, d)) == 0)
                return true;
        if (dcmp(dot(c-a, c-b)) > 0 && dcmp(dot(d-a, d-b)) > 0
              && dcmp(dot(c-b, d-b)) > 0)
            return false;
        return true;
    }
    if (dcmp(cross(d-a, b-a)) * dcmp(cross(c-a, b-a)) > 0)
        return false;
    if (dcmp(cross(a-c, d-c)) * dcmp(cross(b-c, d-c)) > 0)
        return false;
    return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
    b=b-a; d=c-d; c=c-a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
    b=(a+b)/2;
    c=(a+c)/2;
    return ComputeLineIntersection(b, b+RotateCW90(a-b), c,
          c+RotateCW90(a-c));
}

bool PointOnSegment(PT s, PT e, PT p) {
    if(p == s || p == e)  return 1;
    return dcmp(cross(s-p, s-e)) == 0
        && dcmp(dot(PT(s.x-p.x, s.y-p.y), PT(e.x-p.x,
             e.y-p.y))) < 0;
}

int PointInPolygon(vector < PT > p, PT q) {
    int i, j, cnt = 0;
    int n = p.size();
    for(i = 0, j = n-1; i < n; j = i++) {
        if(PointOnSegment(p[i], p[j], q))
            return 1;
        if(p[i].y > q.y != p[j].y > q.y &&
           q.x <
                (LD)(p[j].x-p[i].x)*(q.y-p[i].y)/(LD)(p[j].y-p[i].y)
                + p[i].x)
            cnt++;
    }
    return cnt&1;
}

// determine if point is on the boundary of a polygon
```

```
bool PointOnPolygon(const vector<PT> &p, PT q) {
    for (int i = 0; i < p.size(); i++)
        if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()],
             q), q) < EPS)
            return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
//THIS DOESN'T WORK FOR a == b
vector<PT> CircleLineIntersection(PT a, PT b, PT c, LD r) {
    vector<PT> ret;
    b = b-a;
    a = a-c;
    LD A = dot(b, b);
    LD B = dot(a, b);
    LD C = dot(a, a) - r*r;
    LD D = B*B - A*C;
    if (D < -EPS) return ret;
    ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
    if (D > EPS)
        ret.push_back(c+a+b*(-B-sqrt(D))/A);
    return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, LD r, LD R) {
    vector<PT> ret;
    LD d = sqrt(dist2(a, b));
    if (d > r+R || d+min(r, R) < max(r, R)) return ret;
    LD x = (d*d-R*R+r*r)/(2*d);
    LD y = sqrt(r*r-x*x);
    PT v = (b-a)/d;
    ret.push_back(a+v*x + RotateCCW90(v)*y);
    if (y > 0)
        ret.push_back(a+v*x - RotateCCW90(v)*y);
    return ret;
}

// This code computes the area or centroid of a (possibly
    nonconvex)
// polygon, assuming that the coordinates are listed in a
    clockwise or
// counterclockwise fashion. Note that the centroid is often
    known as
// the "center of gravity" or "center of mass".
LD ComputeSignedArea(const vector<PT> &p) {
    LD area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}

LD ComputeArea(const vector<PT> &p) {
    return fabs(ComputeSignedArea(p));
}
```

```
LD ShoeLace(vector<PT> &vec) {
    int i,n;
    LD ans=0;
    n=vec.size();
    for(i=0;i<n;i++)
         ans+=vec[i].x*vec[NEX(i)].y-vec[i].y*vec[NEX(i)].x;
    return fabs(ans)*0.5;
}

PT ComputeCentroid(const vector<PT> &p) {
    PT c(0,0);
    LD scale = 6.0 * ComputeSignedArea(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
}

LD PAngle(PT a,PT b,PT c) { //Returns positive angle abc
    PT temp1(a.x-b.x,a.y-b.y),temp2(c.x-b.x,c.y-b.y);
    LD
         ans=myAsin((temp1.x*temp2.y-temp1.y*temp2.x)/(temp1.Magnitude()*temp2.Magnitude()));
    if((ans<0&&(temp1.x*temp2.x+temp1.y*temp2.y)<0)||(ans>=0&&(temp1.x*temp2.x+temp1.y*temp2.y)<0))
        ans=PI-ans;
    ans=ans<0?2*PI+ans:ans;
    return ans;
}

LD SignedArea(PT a,PT b,PT c){ //The area is positive if abc is
     in counter-clockwise direction
    PT temp1(b.x-a.x,b.y-a.y),temp2(c.x-a.x,c.y-a.y);
    return 0.5*(temp1.x*temp2.y-temp1.y*temp2.x);
}

bool XYasscending(PT a,PT b) {
    if(abs(a.x-b.x)<EPS) return a.y<b.y;
    return a.x<b.x;
}

//Makes convex hull in counter-clockwise direction without
     repeating first point
//undefined if all points in given[] are collinear
//to allow 180' angle replace <= with <
void MakeConvexHull(vector<PT>given,vector<PT>&ans){
    int i,n=given.size(),j=0,k=0;
    vector<PT>U,L;
    ans.clear();
    sort(given.begin(),given.end(),XYasscending);
    for(i=0;i<n;i++){
        while(true){
            if(j<2) break;
            if(SignedArea(L[j-2],L[j-1],given[i])<=EPS) j--;
            else break;
        }
        if(j==L.size()){
            L.push_back(given[i]);
            j++;
        }
        else{
            L[j]=given[i];
            j++;
        }
    }
    for(i=n-1;i>=0;i--){
        while(1){
            if(k<2) break;
            if(SignedArea(U[k-2],U[k-1],given[i])<=EPS) k--;
            else break;
        }
        if(k==U.size()){
            U.push_back(given[i]);
            k++;
        }
        else{
            U[k]=given[i];
            k++;
        }
    }
    for(i=0;i<j-1;i++) ans.push_back(L[i]);
    for(i=0;i<k-1;i++) ans.push_back(U[i]);
}
typedef PT vector;
typedef vector<PT> Polygon;

struct DirLine {
    PT p;
    Vector v;
    LD ang;
    DirLine () {}
//    DirLine (PT p, Vector v): p(p), v(v) { ang = atan2(v.y,
     v.x); }
    //adds the left of line p-q
    DirLine (PT p, PT q) { this->p = p; this->v.x = q.x-p.x;
         this->v.y = q.y-p.y; ang = atan2(v.y, v.x); }
    bool operator < (const DirLine& u) const { return ang <
         u.ang; }
};

bool getIntersection (PT p, Vector v, PT q, Vector w, PT& o) {
    if (dcmp(cross(v, w)) == 0) return false;
    Vector u = p - q;
    LD k = cross(w, u) / cross(v, w);
    o = p + v * k;
    return true;
}

bool onLeft(DirLine l, PT p) { return dcmp(l.v * (p-l.p)) >= 0;
     }

int halfPlaneIntersection(DirLine* li, int n, vector<PT>& poly)
     { // my_edit
    sort(li, li + n);

    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(q[last].v * q[last-1].v) == 0) {
            last--;
            if (onLeft(q[last], li[i].p)) q[last] = li[i];
        }

        if (first < last)
            getIntersection(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; return
         0; }
    getIntersection(q[last].p, q[last].v, q[first].p,
         q[first].v, p[last]);
    poly.resize(last - first + 1); // my_edit
    int m = 0;
    for (int i = first; i <= last; i++) poly[m++] = p[i];
    delete [] p; delete [] q;
    return m;
}

// 2nd_part

LD CirclishArea(PT a, PT b, PT cen, LD r) {
    LD ang = fabs(atan2((a-cen).y, (a-cen).x) -
         atan2((b-cen).y, (b-cen).x));
    if (ang > PI) ang = 2*PI - ang;
    return (ang * r * r) / 2.0;
}
//intersection of circle and triangle
LD CicleTriangleIntersectionArea(PT a, PT b, PT c, LD radius) {
    vector<PT>g = CircleLineIntersection(a, b, c, radius);
    if (b < a) swap(a, b);
    if (g.size() < 2) return CirclishArea(a, b, c, radius);
    else {
        PT l = g[0], r = g[1];
        if (r < l) swap(l, r);
        if (b < l || r < a) return CirclishArea(a, b, c,
             radius);
        else if (a < l && b < r) return fabs(SignedArea(c, b,
             l)) + CirclishArea(a, l, c, radius);
        else if (r < b && l < a) return fabs(SignedArea(a, c,
             r)) + CirclishArea(r, b, c, radius);
        else if (a < l && r < b) return fabs(SignedArea(c, l,
             r)) + CirclishArea(a, l, c, radius) +
             CirclishArea(b, r, c, radius);
        else return fabs(SignedArea(a, b, c));
    }
    return 0;
}
```

```
//intersection of circle and simple polygon (vertexes in
     counterclockwise order)
LD CirclePolygonIntersectionArea(vector<PT> &p, PT c, LD r) {
    int i, j, k, n = p.size();
    LD sum = 0;
    for (i = 0; i < p.size(); i++) {
        LD temp = CicleTriangleIntersectionArea(p[i],
             p[(i+1)%n], c, r);
        LD sign = SignedArea(c, p[i], p[(i+1)%n]);
        if (dcmp(sign) == 1) sum += temp;
        else if (dcmp(sign) == -1) sum -= temp;
    }
    return sum;
}

//returns the left portion of convex polygon u cut by line a---b
vector<PT> CutPolygon(vector<PT> &u, PT a, PT b) {
    vector<PT> ret;
    int n = u.size();
    for (int i = 0; i < n; i++) {
        PT c = u[i], d = u[(i+1)%n];
        if (dcmp((b-a)*(c-a)) >= 0) ret.push_back(c);
        if (ProjectPointLine(a, b, c) == c ||
            ProjectPointLine(a, b, d) == d) continue;
        if (dcmp((b-a)*(d-c)) != 0) {
            PT t;
            getIntersection(a, b-a, c, d-c, t);
            if (PointOnSegment(c, d, t))
                ret.push_back(t);
        }
    }
    return ret;
}

typedef pair < PT, PT > seg_t;

vector<PT> tanCP(PT c, LD r, PT p) {
    LD x = dot(p - c, p - c);
    LD d = x - r * r;
    vector<PT> res;
    if (d < -EPS) return res;
    if (d < 0) d = 0;
    PT q1 = (p - c) * (r * r / x);
    PT q2 = RotateCCW90((p - c) * (-r * sqrt(d) / x));
    res.push_back(c + q1 - q2);
    res.push_back(c + q1 + q2);
    return res;
}
//Always check if the circles are same
vector<seg_t> tanCC(PT c1, LD r1, PT c2, LD r2) {
    vector<seg_t> res;
    if (fabs(r1 - r2) < EPS) {
    PT dir = c2 - c1;
    dir = RotateCCW90(dir * (r1 / dir.Magnitude()));
    res.push_back(seg_t(c1 + dir, c2 + dir));
    res.push_back(seg_t(c1 - dir, c2 - dir));
    } else {
    PT p = ((c1 * -r2) + (c2 * r1)) / (r1 - r2);
    vector<PT> ps = tanCP(c1, r1, p), qs = tanCP(c2, r2, p);
    for (int i = 0; i < ps.size() && i < qs.size(); ++i) {
      res.push_back(seg_t(ps[i], qs[i]));
    }
    }
    PT p = ((c1 * r2) + (c2 * r1)) / (r1 + r2);
    vector<PT> ps = tanCP(c1, r1, p), qs = tanCP(c2, r2, p);
    for (int i = 0; i < ps.size() && i < qs.size(); ++i) {
    res.push_back(seg_t(ps[i], qs[i]));
    }
    return res;
}
//move segment a---b perpendicularly by distance d
pair < PT, PT > MoveSegmentLeft(PT a, PT b, LD d) {
    LD l = dist(a, b);
    PT p = ((RotateCCW90(b - a) * d) / l) + a;
    return mp(p, p + b - a);
}

void GetLineABC(PT A, PT B, LD &a, LD &b, LD &c) {
    a=A.y-B.y; b=B.x-A.x; c=A.x*B.y-A.y*B.x;
}

LD Sector(LD r, LD alpha) {
    return r * r * 0.5 * (alpha - sin(alpha));
}
LD CircleCircleIntersectionArea(LD r1, LD r2, LD d) {
    if (dcmp(d - r1 - r2) != -1) return 0;
    if (dcmp(d + r1 - r2) != 1) return PI * r1 * r1;
    if (dcmp(d + r2 - r1) != 1) return PI * r2 * r2;
    // using law of cosines
    LD ans = Sector(r1, 2 * acos((r1 * r1 + d * d - r2 * r2) /
         (2.0 * r1 * d)));
    ans += Sector(r2, 2 * acos((r2 * r2 + d * d - r1 * r1) /
         (2.0 * r2 * d)));
    return ans;
}

//length of common part of polygon p and line s-t, O(nlogn)
LD PolygonStubbing(vector <PT> &p, PT s, PT t) {
    int n = p.size();
    LD sm = 0;
    for(int i=0;i<n;i++) sm+=p[i]*p[(i+1)%n];
    if(dcmp(sm) == -1)reverse(p.begin(), p.end()); // my_edit
    vector< pair<LD,int> >event;
    for(int i=0; i<n; i++) {
        int lef = dcmp(cross(p[i]-s, t-s));
        int rig = dcmp(cross(p[NEX(i)]-s, t-s));
        if(lef == rig) continue;
        LD r = cross(p[NEX(i)]-s, p[NEX(i)]-p[i])/cross(t-s,
             p[NEX(i)]-p[i]);
        if(lef>rig) event.push_back(make_pair(r,(!lef || !rig ?
             -1 : -2)));
        else event.push_back(make_pair(r,(!lef || !rig ? 1 :
             2)));
    }
    sort(event.begin(),event.end());
    int cnt = 0;
    LD sum = 0,la = 0;
    for(int i=0; i<(int)event.size(); i++) {
        if (cnt>0) sum += event[i].first-la;
        la = event[i].first;
        cnt += event[i].second;
    }
    return sum*(t-s).Magnitude();
}

// Minimum Enclosing Circle Randomized O(n)
// Removing Duplicates takes O(nlogn)

typedef pair < PT, LD > circle;
bool IsInCircle(circle C, PT p) {
    return dcmp(C.second - dist(C.first, p)) >= 0;
}
circle MinimumEnclosingCircle2(vector < PT > &p, int m, int n) {
    circle D = mp((p[m]+p[n])/2.0, dist(p[m], p[n])/2.0);
    for (int i = 0; i < m; i++)
        if (!IsInCircle(D, p[i])) {
            D.first = ComputeCircleCenter(p[i], p[m], p[n]);
            D.second = dist(D.first, p[i]);
        }
    return D;
}
circle MinimumEnclosingCircle1(vector < PT > &p, int n) {
    circle D = mp((p[0]+p[n])/2.0, dist(p[0], p[n])/2.0);
    for (int i = 1; i < n; i++)
        if (!IsInCircle(D, p[i])) {
            D = MinimumEnclosingCircle2(p, i, n);
        }
    return D;
}
//changes vector; sorts and removes duplicate points(complexity
     bottleneck, unneccessary)
circle MinimumEnclosingCircle(vector < PT > p) {
    srand(time(NULL));
    sort(p.begin(), p.end()); //comment if tle // my_edit
    p.resize(distance(p.begin(), unique(p.begin(), p.end())));
         //comment if tle // my_edit
    random_shuffle(p.begin(), p.end()); // my_edit
    if (p.size() == 1) return mp(p[0], 0);
    circle D = mp((p[0]+p[1])/2.0, dist(p[0], p[1])/2.0);
    for (int i = 2; i < p.size(); i++)
        if (!IsInCircle(D, p[i])) {
            D = MinimumEnclosingCircle1(p, i);
        }
    return D;
}

// UNVERIFIED CODE // my_edit
// Should work if we can see every corner from the inside from
     the center of the minimum enclosing circle
// Sort points clockwise & counterclockwise

PT polygonCenter;

bool less_comp(PT a, PT b)
{
    if (dcmp(a.x - polygonCenter.x) != -1 && dcmp(b.x -
         polygonCenter.x) == -1)
        return true;
    if (dcmp(a.x - polygonCenter.x) == -1 && dcmp(b.x -
         polygonCenter.x) != -1)
```

```
        return false;
    if (dcmp(a.x - polygonCenter.x) == 0 && dcmp(b.x -
          polygonCenter.x) == 0) {
        if (dcmp(a.y - polygonCenter.y) != -1 || dcmp(b.y -
             polygonCenter.y) != -1)
            return a.y > b.y;
        return b.y > a.y;
    }

    // compute the cross product of vectors (polygonCenter ->
          a) x (polygonCenter -> b)
    LD det = (a.x - polygonCenter.x) * (b.y - polygonCenter.y)
          - (b.x - polygonCenter.x) * (a.y - polygonCenter.y);
    if (dcmp(det) == -1)
        return true;
    if (dcmp(det) == 1)
        return false;

    // points a and b are on the same line from the
          polygonCenter
    // check which point is closer to the polygonCenter
    LD d1 = (a.x - polygonCenter.x) * (a.x - polygonCenter.x) +
          (a.y - polygonCenter.y) * (a.y - polygonCenter.y);
    LD d2 = (b.x - polygonCenter.x) * (b.x - polygonCenter.x) +
          (b.y - polygonCenter.y) * (b.y - polygonCenter.y);
    return (d1 - d2) == 1; // This line determines what to do
          if multiple points are on the same "hour.
    // 1 gives furthest first and -1 gives closest first.
}

//This will order the points clockwise starting from the 12
      o'clock.
//Points on the same "hour" will be ordered starting from the
      ones that are further from the center.
void sortCW(vector<PT>& given)
{
    if ((int)given.size() < 3) return;
    polygonCenter = MinimumEnclosingCircle(given).first;
    sort(given.begin(), given.end(), less_comp);
}

//Exactly opposite of sortCW()
void sortCCW(vector<PT>& given)
{
    sortCW(given);
    reverse(given.begin(), given.end());
}

// make sure p1 and p2 are anti-clockwise(because of DirLine)
int PolygonPolygonIntersection(vector<PT> p1, vector<PT> p2,
      vector<PT>& poly) // my_edit
{
    DirLine d[(int)p1.size()+(int)p2.size()];
    for (int i = 0; i < (int)p1.size(); i++) {
        d[i] = DirLine(p1[i], p1[(i+1)%(int)p1.size()]);
    }
    for (int i = 0; i < (int)p2.size(); i++) {
        d[i+(int)p1.size()] = DirLine(p2[i],
              p2[(i+1)%(int)p2.size()]);
    }
    int n = halfPlaneIntersection(d, (int)p1.size() +
          (int)p2.size(), poly);
    sortCCW(poly); // Just to be sure the points are CCW
    return n;
}

// polygon must be clockwise sorted beforehand (use sortCW(..)
      function)
// strictly inside polygon : -1, on polygon : 0, outside
      polygon : 1
// complexity : logn
int pointAndConvexPolygonInLogn(const vector<PT>& poly, PT p)
{
    int n = (int)poly.size();
    // if binary search is going to work
    if (poly[0] == p) return 0;
    else if (onLeft(DirLine(poly[0], poly[1]), p) &&
          onLeft(DirLine(poly[n-1], poly[0]), p)) return 1;
    // binary search
    int low = 1, high = n - 1;
    while (low < high) {
        int mid = (low + high + 1) / 2;
        DirLine d(poly[0], poly[mid]);
        if (onLeft(d, p) || dcmp(DistancePointLine(poly[0],
              poly[mid], p)) == 0) {
            high = mid - 1;
        }
        else {
            low = mid;
        }
    }
    // corner case
    if (low == 1) {
        if (PointOnSegment(poly[0], poly[1], p)) return 0;
        else if (onLeft(DirLine(poly[0], poly[1]), p)) return 1;
    }
    // all other conditions
    if (low == n - 1) {
        return 1;
    }
    else if (low == n - 2) {
        if (PointOnSegment(poly[n-2], poly[n-1], p) ||
              PointOnSegment(poly[0], poly[n-1], p))
            return 0;
        else if (onLeft(DirLine(poly[n-1], poly[n-2]), p))
            return -1;
        else
            return 1;
    }
    else {
        if (PointOnSegment(poly[low], poly[low+1], p))
            return 0;
        else if (onLeft(DirLine(poly[low+1], poly[low]), p))
            return -1;
        else
            return 1;
    }
}

//********.........GEOMETRY.........********

int main()
{
    int co = 0;
    while (1) {
        int n;
        cin >> n;
        if (!n) break;
        vector<PT> given;
        for (int i = 0; i < n; i++) {
            PT temp;
            cin >> temp;
            given.push_back(temp);
        }
        sortCW(given);
        for (int i = 0; i < n; i++) {
            cout << given[i] << endl;
        }
        sortCCW(given);
        for (int i = 0; i < n; i++) {
            cout << given[i] << endl;
        }
    }

    return 0;
}
```

4 Graph

4.1 Bipartite Matching Optimized Kuhn (weighted nodes allowed)

```
/*
    Optimized Kuhn (weighted nodes allowed) for Bipartite
          Matching
*/

/*
    Only one side is weighted. The vector 'order' stores all
    pair<weight[node], node> - such pairs. We assume that
    the left side (corresponding variables: n, L) is
    weighted.
    'order' is sorted as desired first.
    'mt' from this snippet and 'R' from the other version
    are same.
    Many other things are similar.
*/

/*
    Source:
        http://e-maxx.ru/algo/vertex_weighted_matching
        http://e-maxx.ru/algo/kuhn_matching
*/


vector<pair<int, int>> order;
```

```
struct BipartiteMatcher {
    int n, k;
    vector < vector<int> > g;
    vector<int> mt;
    vector<char> used;
    vector<int> L;

    BipartiteMatcher(int nn, int kk)
    {
        n = nn;
        k = kk;
        g.resize(n);
        used.resize(n);
        mt.resize(k);
        L.resize(n);
    }

    void AddEdge(int u, int v)
    {
        g[u].push_back(v);
    }

    bool try_kuhn (int v) {
        if (used[v]) return false;
        used[v] = true;
        for (size_t i=0; i<g[v].size(); ++i) {
            int to = g[v][i];
            if (mt[to] == -1 || try_kuhn (mt[to])) {
                mt[to] = v;
                return true;
            }
        }
        return false;
    }

    int Solve() {
        sort(all(order));
        int ret = 0;

        mt.assign (k, -1);
        for (int v=0; v<n; ++v) {
            int x = order[v].second;
            used.assign (n, false);
            try_kuhn (x);
            L[x] = -1;
        }

        for (int i=0; i<k; ++i)
            if (mt[i] != -1) {
                ret++;
                L[mt[i]] = i;
            }

        return ret;
    }
};

/*
    Optimized Kuhn (weighted nodes allowed) for Bipartite
        Matching
*/
```

4.2 Bipartite Matching Optimized Kuhn

```
/*
    #1. Optimized Kuhn for Maximum Matching on Bipartite Graph
    #2. Worst case: O(V*V) (much faster in use)
    #3. Works better than hopcroft-karp (if not always, almost
        always)
*/
struct BipartiteMatcher {
    vector<vector<int>> G;
    vector<int> L, R, Viz;

    BipartiteMatcher(int n, int m) :
        G(n), L(n, -1), R(m, -1), Viz(n) {
    }
    void AddEdge(int a, int b)
    {
        G[a].push_back(b);
    }
    bool Match(int node)
    {
        if (Viz[node]) return false;
        Viz[node] = true;
        for (auto vec : G[node]) {
            if (R[vec] == -1) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
        for (auto vec : G[node]) {
            if (Match(R[vec])) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
        return false;
    }
    int Solve()
    {
        bool ok = true;
        while (ok) {
            ok = 0;
            fill(Viz.begin(), Viz.end(), 0);
            for (int i = 0; i < (int)L.size(); ++i) {
                if (L[i] == -1) ok |= Match(i);
            }
        }
        int ret = 0;
        for (int i = 0; i < L.size(); ++i)
            ret += (L[i] != -1);
        return ret;
    }
};
```

4.3 Bipartite Matching Variation - Minimum Path Cover (Vertex Disjoint) of a DAG

```
/*
    #1. Optimized Kuhn for Maximum Matching on Bipartite Graph
    #2. Worst case: O(V*V) (much faster in use)
    #3. Works better than hopcroft-karp (if not always, almost
        always)
*/
struct BipartiteMatcher {
    vector<vector<int>> G;
    vector<int> L, R, Viz;

    BipartiteMatcher(int n, int m) :
        G(n), L(n, -1), R(m, -1), Viz(n) {
    }
    void AddEdge(int a, int b)
    {
        G[a].push_back(b);
    }
    bool Match(int node)
    {
        if (Viz[node]) return false;
        Viz[node] = true;
        for (auto vec : G[node]) {
            if (R[vec] == -1) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
        for (auto vec : G[node]) {
            if (Match(R[vec])) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
        return false;
    }
    int Solve()
    {
        bool ok = true;
        while (ok) {
            ok = 0;
            fill(Viz.begin(), Viz.end(), 0);
            for (int i = 0; i < (int)L.size(); ++i) {
                if (L[i] == -1) ok |= Match(i);
            }
        }
        int ret = 0;
        for (int i = 0; i < L.size(); ++i)
            ret += (L[i] != -1);
    }
};
```

```
        return ret;
    }
};

/*
    #1. Minimum Path Cover (Vertex Disjoint) of a DAG
    #2. We want to cover all the nodes using minimum number of
          paths that don't share any vertex
    #2. Intuition:
        i.
              https://en.wikipedia.org/wiki/Maximum_flow_problem#Minimum_path_cover_in_directed_acyclic_graph
        ii.
              https://towardsdatascience.com/solving-minimum-path-cover-on-a-dag-21b16ca11a0
        iii.
              https://codeforces.com/blog/entry/13320?#comment-181252
*/
struct MPC_VD_DAG
{
    int n;
    BipartiteMatcher G = BipartiteMatcher(0, 0);
    MPC_VD_DAG(int n) : n(n), G(n, n) {}
    void addEdge(int u, int v) {
        G.AddEdge(u, v);
    }
    int solve() {
        return n - G.Solve();
    }
};
```

4.4 Bipartite Matching Variation - Minimum Path Cover of a Cyclic Graph where a Path Can Intersect with Itself but Not with Other Paths

```
/*
    #1. Minimum Path Cover on a Directed Graph (can contain
          cycles) where the path can intersect with itself, but
          not with other paths
    #2. After the helper DFS runs, O(3^n) Submask DP is used
    #3. The nodes are 0 indexed
*/
struct MPC
{
    int n;
    vector<VI> g;
    vector<VB> vis;
    VB path_exists;
    MPC(int n) : n(n), g(n) {}
    void addEdge(int u, int v) {
        g[u].pb(v);
    }
    void dfs(int mask, int last) {
        vis[mask][last] = 1;
        path_exists[mask] = 1;
        for (int v : g[last]) {
            int next_mask = ON(mask, v);
            if (!vis[next_mask][v]) dfs(next_mask, v);
        }
    }
    int solve() {
        VI dp(1 << n);
        vis = vector<VB> (1 << n, VB(n));
        path_exists = VB(1 << n);
        for (int i = 0; i < n; i++) {
            dfs(ON(0, i), i);
        }
        for (int mask = 0; mask < (1 << n); mask++) {
            dp[mask] = !mask ? 0 : (path_exists[mask] ? 1 :
                  CNT(mask));
            if (mask && !path_exists[mask]) {
                for (int submask = (mask - 1) & mask; submask >
                      0; submask = (submask - 1) & mask) {
                    dp[mask] = min(dp[mask], dp[submask] +
                          dp[submask^mask]);
                }
            }
        }
        return dp[(1<<n)-1];
    }
};
```

4.5 Bipartite Matching Variation - Minimum Path Cover of a DAG

```
/*
    #1. Optimized Kuhn for Maximum Matching on Bipartite Graph
    #2. Worst case: O(V*V) (much faster in use)
    #3. Works better than hopcroft-karp (if not always, almost
          always)
*/
struct BipartiteMatcher {
    vector<vector<int>> G;
    vector<int> L, R, Viz;

    BipartiteMatcher(int n, int m) :
        G(n), L(n, -1), R(m, -1), Viz(n) {
    }
    void AddEdge(int a, int b)
    {
        G[a].push_back(b);
    }
    bool Match(int node)
    {
        if (Viz[node]) return false;
        Viz[node] = true;
        for (auto vec : G[node]) {
            if (R[vec] == -1) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
        for (auto vec : G[node]) {
            if (Match(R[vec])) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
        return false;
    }
    int Solve()
    {
        bool ok = true;
        while (ok) {
            ok = 0;
            fill(Viz.begin(), Viz.end(), 0);
            for (int i = 0; i < (int)L.size(); ++i) {
                if (L[i] == -1) ok |= Match(i);
            }
        }
        int ret = 0;
        for (int i = 0; i < L.size(); ++i)
            ret += (L[i] != -1);
        return ret;
    }
};

/*
    #1. Minimum Path Cover (Vertex Disjoint) of a DAG
    #2. We want to cover all the nodes using minimum number of
          paths that don't share any vertex
    #2. Intuition:
        i.
              https://en.wikipedia.org/wiki/Maximum_flow_problem#Minimum_p
        ii.
              https://towardsdatascience.com/solving-minimum-path-cover-o
        iii.
              https://codeforces.com/blog/entry/13320?#comment-181252
*/
struct MPC_VD_DAG
{
    int n;
    BipartiteMatcher G = BipartiteMatcher(0, 0);
    MPC_VD_DAG(int n) : n(n), G(n, n) {}
    void addEdge(int u, int v) {
        G.AddEdge(u, v);
    }
    int solve() {
        return n - G.Solve();
    }
};

/*
    #1. Minimum Path Cover of a DAG
    #2. We want to cover all the nodes using minimum number of
          paths that may share nodes
    #3. After finding the transitive closure, it becomes Vertex
          Disjoint Minimum Path Cover
    #4. Intuition:
        i.
              https://codeforces.com/blog/entry/14688?#comment-196673
```

```
*/
struct MPC_DAG
{
    int n;
    vector<vector<int>> g;
    MPC_DAG(int n) : n(n), g(n) {}
    void addEdge(int u, int v) {
        g[u].pb(v);
    }
    void dfs(int u, vector<bool> &vis, vector<int> &nodes) {
        vis[u] = 1;
        nodes.pb(u);
        for (int v : g[u]) {
            if (!vis[v]) dfs(v, vis, nodes);
        }
    }
    int solve() {
        MPC_VD_DAG G(n);
        for (int i = 0; i < n; i++) {
            vector<bool> vis(n, 0);
            vector<int> nodes;
            dfs(i, vis, nodes);
            for (int node : nodes) {
                if (node != i) G.addEdge(i, node);
            }
        }
        return G.solve();
    }
};
```

4.6 Block Cut Tree, Biconnected Component Articulation Point

```
/* Block Cut Tree, Biconnected Component & Articulation Point */

struct graph
{
        int n;
        vector<vector<int>> adj;

        graph(int n) : n(n), adj(n) {}

        void add_edge(int u, int v)
        {
                adj[u].push_back(v);
                adj[v].push_back(u);
        }

        int add_node()
        {
                adj.push_back({});
                return n++;
        }

        vector<int>& operator[](int u) { return adj[u]; }
};

pair<vector<vector<int>>, vector<int>>
      biconnected_components_articulation_points(graph &adj)
{
        int n = adj.n;

        vector<int> num(n), low(n), art(n), stk;
        vector<vector<int>> comps;

        function<void(int, int, int&)> dfs = [&](int u, int p,
              int &t)
        {
                num[u] = low[u] = ++t;
                stk.push_back(u);

                for (int v : adj[u]) if (v != p)
                {
                        if (!num[v])
                        {
                                dfs(v, u, t);
                                low[u] = min(low[u], low[v]);

                                if (low[v] >= num[u])
                                {
                                        art[u] = (num[u] > 1 ||
                                              num[v] > 2);

                                        comps.push_back({u});
                                        while (comps.back().back()
                                              != v)
                                          comps.back().push_back(stk.back()),
                                                stk.pop_back();

                                }
                        }
                        else low[u] = min(low[u], num[v]);
                }
        };

        for (int u = 0, t; u < n; ++u)
                if (!num[u]) dfs(u, -1, t = 0);

        return {comps, art};
}

graph build_block_cut_tree(graph &adj)
{
    int n = adj.n;

    auto pre = biconnected_components_articulation_points(adj);
    vector<int> art = pre.ss;
    vector<vector<int>> comps = pre.ff;

    graph tree(0);
    vector<int> id(n);

    for (int u = 0; u < n; ++u)
        if (art[u]) id[u] = tree.add_node();

    for (auto &comp : comps)
    {
        int node = tree.add_node();
        for (int u : comp)
            if (!art[u]) id[u] = node;
            else tree.add_edge(node, id[u]); /// each edge might
                  be added twice here. verify and/or fix later.
                  possible fix: add directed edges instead of
                  undirected ones
    }

    return tree;
};

/* Block Cut Tree, Biconnected Component, Articulation Point */
```

4.7 Connected Components of a Complete Graph

```
/*
    #1. There is a complete graph, but there are some blocked
          roads.
    #2. We need to find the components without using blocked
          roads.
    #3. Because the number of usable roads is too high, we
          should use this template.
    #4. Otherwise, basic dfs is enough.
    #5. Problems (easy to hard) - (role model submission /
          problem page):
        i.
            https://codeforces.com/contest/1243/submission/82715710
            (direct template)
        ii.
            https://codeforces.com/contest/190/submission/82717910
*/
vector<int> ust[M];
unordered_set<int, custom_hash> un_vis;

bool f(int u, int v)
{
    if (!SZ(ust[u])) return 0;
    auto it = lower_bound(all(ust[u]), v);
    if (it == ust[u].end()) return 0;
    return *it == v;
}

void dfs(int u)
{
    vector<int> a;
    for (int v : un_vis) {
        if (!f(u, v)) {
            a.pb(v);
        }
    }
    for (int v : a) {
        un_vis.erase(v);
    }
    for (int v : a) {
```

```
        dfs(v);
    }
}

int main()
{
    int n, m;
    I(n, m);
    for (int i = 0; i < n; i++) {
        un_vis.insert(i);
    }
    while (m--) {
        int a, b;
        I(a, b);
        a--; b--;
        ust[a].pb(b);
        ust[b].pb(a);
    }
    for (int i = 0; i < n; i++) {
        sort(all(ust[i]));
    }
    int ans = 0;
    for (int i = 0; i < n; i++) {
        if (un_vis.find(i) != un_vis.end()) {
            un_vis.erase(i);
            dfs(i);
            ans++;
        }
    }
    O(ans - 1);

    return 0;
}
```

4.8 Dijkstra BFS (class version) Upgraded

```
/*
    #1. T1 -> Data type required for maximum distance
    #2. T2 -> Data type required for maximum edge weight
    #3. Introduced two data types for memory efficiency
*/
template<typename T1, typename T2>
class DirectedGraph
{
    typedef pair<T1, int> ii;
    const T1 INF = numeric_limits<T1>::max();

public:
    int nodes;
    vector<vector<pair<int, T2>>> AdjList;
    vector<T1> dist;
    vector<int> par;
    DirectedGraph(int n)
    {
        nodes = n;
        AdjList.resize(n);
        dist.resize(n);
        par.resize(n);
    }
    void addEdge(int u, int v, T2 c)
    {
        AdjList[u].push_back(make_pair(v, c));
    }
    void dijkstra(int s)
    {
        for (int i = 0; i < nodes; i++) {
            dist[i] = INF;
            par[i] = -1;
        }
        priority_queue<ii, vector<ii>, greater<ii>> pq; /**
             BFS: queue<ii> pq; */
        dist[s] = 0;
        pq.push(ii(0, s));
        while (!pq.empty()) {
            ii top = pq.top(); /** BFS: pq.front(); */
            pq.pop();
            T1 d = top.first;
            int u = top.second;
            if (dcmp(d - dist[u]) == 0) {
                for (auto it = AdjList[u].begin(); it !=
                     AdjList[u].end(); it++) {
                    int v = it->first; T2 weight_u_v = it->second;
                    if (dcmp(dist[u] + weight_u_v - dist[v]) ==
                         -1) {
                        dist[v] = dist[u] + weight_u_v;
                        par[v] = u;
                        pq.push(ii(dist[v], v));
                    }
                }
            }
        }
    }
    vector<int> path(int dest)
    {
        vector<int> ret;
        if (par[dest] == -1)
            return ret;
        int curr = dest;
        while (curr != -1) {
            ret.pb(curr);
            curr = par[curr];
        }
        reverse(all(ret));
        return ret;
    }
};
```

4.9 Dijkstra BFS Variation - Minimum Weight Cycle in Directed Graph

```
/*
    #1. T1 -> Data type required for maximum distance
    #2. T2 -> Data type required for maximum edge weight
    #3. Introduced two data types for memory efficiency
*/
template<typename T1, typename T2>
class DirectedGraph
{
    typedef pair<T1, int> ii;
    const T1 INF = numeric_limits<T1>::max();

public:
    int nodes;
    vector<vector<pair<int, T2>>> AdjList;
    vector<T1> dist;
    vector<int> par;
    DirectedGraph(int n)
    {
        nodes = n;
        AdjList.resize(n);
        dist.resize(n);
        par.resize(n);
    }
    void addEdge(int u, int v, T2 c)
    {
        AdjList[u].push_back(make_pair(v, c));
    }
    T1 minimumWeightCycle(int s)
    {
        T1 ret = numeric_limits<T1>::max();
        for (int i = 0; i < nodes; i++) {
            dist[i] = INF;
            par[i] = -1;
        }
        priority_queue<ii, vector<ii>, greater<ii>> pq; /** For
             unweighted graph(BFS): queue<ii> pq; */
        dist[s] = 0;
        pq.push(ii(0, s));
        while (!pq.empty()) {
            ii top = pq.top(); /** For unweighted graph(BFS):
                 pq.front(); */
            pq.pop();
            T1 d = top.first;
            int u = top.second;
            if (dcmp(d - dist[u]) == 0) {
                for (auto it = AdjList[u].begin(); it !=
                     AdjList[u].end(); it++) {
                    int v = it->first; T2 weight_u_v = it->second;
                    if (dcmp(dist[u] + weight_u_v - dist[v]) ==
                         -1) {
                        dist[v] = dist[u] + weight_u_v;
                        par[v] = u;
                        pq.push(ii(dist[v], v));
                    }
                    else if (v == s) {
                        ret = min(ret, dist[u] + weight_u_v); /**
                             store best u and run path(u) later
                             to print cycle */
                    }
                }
            }
        }
        return ret;
```

```
    }
    vector<int> path(int dest)
    {
        vector<int> ret;
        if (par[dest] == -1)
            return ret;
        int curr = dest;
        while (curr != -1) {
            ret.pb(curr);
            curr = par[curr];
        }
        reverse(all(ret));
        return ret;
    }
};
```

4.10 Dijkstra BFS Variation - Minimum Weight Cycle in Undirected Graph

```
/*
    #1. T1 -> Data type required for maximum distance
    #2. T2 -> Data type required for maximum edge weight
    #3. Introduced two data types for memory efficiency
    #4. Problems (easy to hard) - (role model submission /
          problem page):
        i.
             https://codeforces.com/contest/1206/submission/83601330
             (direct template)
        ii.
             https://codeforces.com/contest/1364/submission/83696823
*/
template<typename T1, typename T2>
class DirectedGraph
{
    typedef pair<T1, int> ii;
    const T1 INF = numeric_limits<T1>::max();

public:
    int nodes;
    int furthest_member, second_furthest;
    vector<vector<pair<int, T2>>> AdjList;
    vector<T1> dist;
    vector<int> par;
    DirectedGraph(int n)
    {
        nodes = n;
        AdjList.resize(n);
        dist.resize(n);
        par.resize(n);
    }
    void addEdge(int u, int v, T2 c)
    {
        AdjList[u].push_back(make_pair(v, c));
    }
    T1 minimumWeightCycle(int s)
    {
        T1 ret = numeric_limits<T1>::max();
        for (int i = 0; i < nodes; i++) {
            dist[i] = INF;
            par[i] = -1;
        }
        priority_queue<ii, vector<ii>, greater<ii>> pq; /** For
             unweighted graph(BFS): queue<ii> pq; */
        dist[s] = 0;
        pq.push(ii(0, s));
        while (!pq.empty()) {
            ii top = pq.top(); /** For unweighted graph(BFS):
                 pq.front(); */
            pq.pop();
            T1 d = top.first;
            int u = top.second;
            if (dcmp(d - dist[u]) == 0) {
                for (auto it = AdjList[u].begin(); it !=
                     AdjList[u].end(); it++) {
                    int v = it->first; T2 weight_u_v = it->second;
                    if (dcmp(dist[u] + weight_u_v - dist[v]) ==
                         -1) {
                        dist[v] = dist[u] + weight_u_v;
                        par[v] = u;
                        pq.push(ii(dist[v], v));
                    }
                    else {
                        if (par[v] != u && par[u] != v) {
                            T1 curr = dist[u] + dist[v] +
                                 weight_u_v;
                            if (curr < ret) {
                                ret = curr;
                                furthest_member = u;
                                second_furthest = v;
                            }
                        }
                    }
                }
            }
        }
        return ret;
    }
    vector<int> path(int dest)
    {
        vector<int> ret;
        if (par[dest] == -1)
            return ret;
        int curr = dest;
        while (curr != -1) {
            ret.pb(curr);
            curr = par[curr];
        }
        reverse(all(ret));
        return ret;
    }
    vector<int> cycle()
    {
        vector<int> path1 = path(furthest_member);
        vector<int> path2 = path(second_furthest);
        reverse(all(path2)); path2.pop_back();
        vector<int> ret;
        for (int u : path1) ret.pb(u);
        for (int u : path2) ret.pb(u);
        return ret;
    }
    vector<int> flower()
    {
        /**
            Because this is an undirected graph, sometimes the
                 minimum weight cycle through 's' may look like
                 a flower,
            with repeating nodes at the beginning and at the end
                 with a simple cycle in between.
            This function returns the simple cycle in the middle
                 (top of the flower).
        */
        vector<int> path1 = path(furthest_member);
        vector<int> path2 = path(second_furthest);
        reverse(all(path1)); reverse(all(path2));
        int flower_start;
        while (path1.back() == path2.back()) {
            flower_start = path1.back();
            path1.pop_back();
            path2.pop_back();
        }
        reverse(all(path1));
        vector<int> ret;
        ret.push_back(flower_start);
        for (int u : path1) ret.pb(u);
        for (int u : path2) ret.pb(u);
        return ret;
    }
};
```

4.11 Dinic Better

```
/* Dinic */

struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u),
         cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
```

```
}

void add_edge(int v, int u, long long cap) {
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;
}

bool bfs() {
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int id : adj[v]) {
            if (edges[id].cap - edges[id].flow < 1)
                continue;
            if (level[edges[id].u] != -1)
                continue;
            level[edges[id].u] = level[v] + 1;
            q.push(edges[id].u);
        }
    }
    return level[t] != -1;
}

long long dfs(int v, long long pushed) {
    if (pushed == 0)
        return 0;
    if (v == t)
        return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size();
         cid++) {
        int id = adj[v][cid];
        int u = edges[id].u;
        if (level[v] + 1 != level[u] || edges[id].cap -
            edges[id].flow < 1)
            continue;
        long long tr = dfs(u, min(pushed, edges[id].cap -
            edges[id].flow));
        if (tr == 0)
            continue;
        edges[id].flow += tr;
        edges[id ^ 1].flow -= tr;
        return tr;
    }
    return 0;
}

long long flow() {
    long long f = 0;
    while (true) {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        q.push(s);
        if (!bfs())
            break;
        fill(ptr.begin(), ptr.end(), 0);
        while (long long pushed = dfs(s, flow_inf)) {
            f += pushed;
        }
    }
    return f;
  }
};

/* Dinic */
```

4.12 General graph maximum matching

```
/*... general matching(undirected) ...*/

struct GenMatch { // 1-base
    static const int MAXN = 514;
    int V;
    bool el[MAXN][MAXN];
    int pr[MAXN];
    bool inq[MAXN],inp[MAXN],inb[MAXN];
    queue<int> qe;
    int st,ed;
    int nb;
    int bk[MAXN],djs[MAXN];
    int ans;
    void init(int _V) {
        V = _V;
        for(int i = 0; i <= V; i++) {
            for(int j = 0; j <= V; j++) el[i][j] = 0;
            pr[i] = bk[i] = djs[i] = 0;
            inq[i] = inp[i] = inb[i] = 0;
        }
        ans = 0;
    }
    void add_edge(int u, int v) {
        el[u][v] = el[v][u] = 1;
    }
    int lca(int u,int v) {
        for(int i = 0; i <= V; i++) inp[i] = 0;
        while(1) {
            u = djs[u];
            inp[u] = true;
            if(u == st) break;
            u = bk[pr[u]];
        }
        while(1) {
            v = djs[v];
            if(inp[v]) return v;
            v = bk[pr[v]];
        }
        return v;
    }
    void upd(int u) {
        int v;
        while(djs[u] != nb) {
            v = pr[u];
            inb[djs[u]] = inb[djs[v]] = true;
            u = bk[v];
            if(djs[u] != nb) bk[u] = v;
        }
    }
    void blo(int u,int v) {
        nb = lca(u,v);
        for (int i=0; i<=V; i++) inb[i] = 0;
        upd(u); upd(v);
        if(djs[u] != nb) bk[u] = v;
        if(djs[v] != nb) bk[v] = u;
        for(int tu = 1; tu <= V; tu++)
            if(inb[djs[tu]]) {
                djs[tu] = nb;
                if(!inq[tu]){
                    qe.push(tu);
                    inq[tu] = 1;
                }
            }
    }
    void flow() {
        for(int i = 1; i <= V; i++) {
            inq[i] = 0;
            bk[i] = 0;
            djs[i] = i;
        }

        while(qe.size()) qe.pop();
        qe.push(st);
        inq[st] = 1;
        ed = 0;
        while(qe.size()) {
            int u = qe.front(); qe.pop();
            for(int v = 1; v <= V; v++)
                if(el[u][v] && (djs[u] != djs[v])
                    && (pr[u] != v)) {
                    if((v == st) || ((pr[v] >
                        0) && bk[pr[v]] > 0))
                        blo(u,v);
                    else if(bk[v] == 0) {
                        bk[v] = u;
                        if(pr[v] > 0) {
                            if(!inq[pr[v]])
                                qe.push(pr[v]);
                        } else {
                            ed = v;
                            return;
                        }
                    }
                }
        }
    }
    void aug() {
        int u,v,w;
        u = ed;
        while(u > 0) {
            v = bk[u];
            w = pr[v];
            pr[v] = u;
            pr[u] = v;
            u = w;
        }
    }
```

```
	int solve() {
		for(int i = 0; i <= V; i++) pr[i] = 0;
		for(int u = 1; u <= V; u++)
			if(pr[u] == 0) {
				st = u;
				flow();
				if(ed > 0) {
					aug();
					ans ++;
				}
			}
		return ans;
	}
}G;

/*... general matching(undirected) ...*/
```

4.13 Kosaraju's Algorithm for Strongly Connected Components

```
/*
    #1. Finds the strongly connected components and creates the
        DAG made by considering strongly connected components
        as nodes
    #2. The algorithm is Kosaraju's algorithm
*/
struct Kosaraju
{
    vector<bool> vis;
    stack<int> stck;
    vector<vector<int>> g, rg;
    vector<int> col;
    vector<vector<int>> res_dag;
    void dfs1(int u)
    {
        vis[u] = 1;
        for (int v : g[u]) {
            if (!vis[v]) dfs1(v);
        }
        stck.push(u);
    }
    void dfs2(int u, int curr)
    {
        col[u] = curr;
        vis[u] = 1;
        for (int v : rg[u]) {
            if (!vis[v]) dfs2(v, curr);
        }
    }
    void solve(const vector<vector<int>> &gg)
    {
        g = gg;
        int n = SZ(g);
        rg.clear();
        rg.resize(n);
        for (int u = 0; u < n; u++) {
            for (int v : g[u]) rg[v].pb(u);
        }
        stck = stack<int>();
        vis.resize(n);
        fill(all(vis), 0);
        col.resize(n);
        res_dag.clear();
        /* Algorithm Start */
        for (int i = 0; i < n; i++) {
            if (!vis[i]) dfs1(i);
        }
        fill(all(vis), 0);
        int scc_count = 0;
        while (!stck.empty()) {
            int top = stck.top();
            stck.pop();
            if (!vis[top]) dfs2(top, scc_count++);
        }
        res_dag.resize(scc_count);
        for (int u = 0; u < n; u++) {
            for (int v : g[u]) {
                if (col[u] != col[v]) {
                    res_dag[col[u]].pb(col[v]); /* Multiple edges
                        might be present, can be modified to
                        change this */
                }
            }
        }
    }
};
```

4.14 Maximum Independent Set of a Bipartite Graph

```
/*
    #1. Optimized Kuhn for Maximum Matching on Bipartite Graph
    #2. Worst case: O(V*V) (much faster in use)
    #3. Works better than hopcroft-karp (if not always, almost
        always)
*/
struct BipartiteMatcher {
    vector<vector<int>> G;
    vector<int> L, R, Viz;

    BipartiteMatcher(int n, int m) :
        G(n), L(n, -1), R(m, -1), Viz(n) {
    }
    void AddEdge(int a, int b)
    {
        G[a].push_back(b);
    }
    bool Match(int node)
    {
        if (Viz[node]) return false;
        Viz[node] = true;
        for (auto vec : G[node]) {
            if (R[vec] == -1) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
        for (auto vec : G[node]) {
            if (Match(R[vec])) {
                L[node] = vec;
                R[vec] = node;
                return true;
            }
        }
        return false;
    }
    int Solve()
    {
        bool ok = true;
        while (ok) {
            ok = 0;
            fill(Viz.begin(), Viz.end(), 0);
            for (int i = 0; i < (int)L.size(); ++i) {
                if (L[i] == -1) ok |= Match(i);
            }
        }
        int ret = 0;
        for (int i = 0; i < L.size(); ++i)
            ret += (L[i] != -1);
        return ret;
    }
};

/*
    #1. Maximum Independent Set of a Bipartite Graph
    #2. Intuition:
        i.
            https://en.wikipedia.org/wiki/Bipartite_graph#K%C5%91nig's_
*/
struct MIS
{
    int n, m;
    BipartiteMatcher G = BipartiteMatcher(0, 0);
    MIS(int n, int m) : n(n), m(m), G(n, m) {}
    void addEdge(int u, int v) {
        G.AddEdge(u, v);
    }
    int solve() {
        return n + m - G.Solve();
    }
};
```

4.15 Minimum or Maximum Spanning Tree for Undirected Graph

```
/*
    #1. Elements are numbered from 0 to (n - 1) inclusive
```

```
    #2. Source: Competitive Programming 1 (Steven Halim)
*/
struct UnionFindDisjointSet
{
    vector<int> pset;
    UnionFindDisjointSet(int n) {
        pset.resize(n);
        for (int i = 0; i < n; i++) pset[i] = i;
    }
    int findSet(int i) {
        return (pset[i] == i) ? i : (pset[i] =
             findSet(pset[i]));
    }
    void unionSet(int i, int j) {
        pset[findSet(i)] = findSet(j);
    }
    bool isSameSet(int i, int j) {
        return findSet(i) == findSet(j);
    }
};

/*
    #1. T1 -> Data type required for resultant tree's total
         edge weight (MST cost)
    #2. T2 -> Data type required for maximum edge weight
    #3. Nodes are numbered from 0 to (n - 1) inclusive
    #4. Can find minimum MST or maximum MST depending on value
         of 'bool minimum' during function call
    #5. Introduced two data types for memory efficiency
    #6. Source: Competitive Programming 1 (Steven Halim)
*/
template<typename T1, typename T2>
struct UndirectedGraphMST
{
    int nodes;
    vector<pair<T2, pii>> edges;

    UndirectedGraphMST(int n)
    {
        nodes = n;
    }
    void addEdge(int u, int v, T2 weight)
    {
        edges.pb({weight, {u, v}});
    }
    pair<T1, vector<pii>> MST(bool minimum) const /* Total cost
         & vector of resultant tree edges */
    {
        T1 mst_cost = 0; vector<pii> tree_edges;
        UnionFindDisjointSet S(nodes);
        vector<pair<T2, pii>> edgeList(all(edges));
        sort(all(edgeList)); if (!minimum)
             reverse(all(edgeList));

        for (auto top : edgeList) {
            if (!S.isSameSet(top.ss.ff, top.ss.ss)) {
                mst_cost += top.ff;
                S.unionSet(top.ss.ff, top.ss.ss);
                tree_edges.pb({top.ss.ff, top.ss.ss});
                /* One of the selected edges, do anything else
                     if needed */
            }
        }
        return {mst_cost, tree_edges};
    }
};
```

4.16 Tree Diameter Head with Minimum or Maximum Possible Node as One of Its Ends

```
/*
    #1. What we want is this: among all the possible tree
         diameters, the one that has the smallest/biggest node
        at one of its ends.
    #2. The idea is very simple, just use/look at the code
         snippet given below.
    #3. The code is for the smallest node and it's a bit
         incomplete, but it conveys the trick.
    #4. Problems (easy to hard) - (role model
         submission/problem page):
        i. https://vjudge.net/solution/26210889
*/
int maxx, max_node;
void dfs(int u, int p, int d)
{
    if (d > maxx) {
        maxx = d;
        max_node = u;
    }
    else if (d == maxx) {
        max_node = min(max_node, u);
    }
    for (int v : g[u]) {
        if (v != p) {
            dfs(v, u, d + 1);
        }
    }
}
int diameter_head()
{
    maxx = -1;
    dfs(0, -1, 0);
    int head = max_node;
    maxx = -1;
    dfs(head, -1, 0);
    int tail = max_node;
    return min(head, tail);
}
```

5 Miscellaneous

5.1 FYI

```
If the problem requires input/output via file(s):
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);

In c++:
    for precision:
        cout << setprecision(x) << fixed << y << setpecision(w)
             << fixed << z << endl;
        where x and w are how precise the output must be and y
             and z are the double
        variables.

For hashing:
    base = 347, mod = 1000000007, mod2 = 22801761391
    all of these are prime numbers

Bitwise fact:
    a + b = a ^ b + 2 * (a & b);

Position of rightmost set bit(0 indexed, from right):
    int pos = log2(mask&-mask);

Given n coins, if we have to distribute them between k
     people(giving someone zero is allowed, but we have to
     give away all of the coins)
How many ways?
    => nCr(n + k - 1, n);

nCr = n! / (r! * (n - r)!)
nPr = n! / (n - r)!
```

5.2 Histogram

```
// C++ program to find maximum rectangular area in
// linear time
#include<iostream>
#include<stack>
using namespace std;

// The main function to find the maximum rectangular
// area under given histogram with n bars
int getMaxArea(int hist[], int n)
{
    // Create an empty stack. The stack holds indexes
    // of hist[] array. The bars stored in stack are
    // always in increasing order of their heights.
    stack<int> s;

    int max_area = 0; // Initalize max area
    int tp; // To store top of stack
    int area_with_top; // To store area with top bar
```

```
                // as the smallest bar

    // Run through all bars of given histogram
    int i = 0;
    while (i < n)
    {
        // If this bar is higher than the bar on top
        // stack, push it to stack
        if (s.empty() || hist[s.top()] <= hist[i])
            s.push(i++);

        // If this bar is lower than top of stack,
        // then calculate area of rectangle with stack
        // top as the smallest (or minimum height) bar.
        // 'i' is 'right index' for the top and element
        // before top in stack is 'left index'
        else
        {
            tp = s.top(); // store the top index
            s.pop(); // pop the top

            // Calculate the area with hist[tp] stack
            // as smallest bar
            area_with_top = hist[tp] * (s.empty() ? i :
                                i - s.top() - 1);

            // update max area, if needed
            if (max_area < area_with_top)
                max_area = area_with_top;
        }
    }

    // Now pop the remaining bars from stack and calculate
    // area with every popped bar as the smallest bar
    while (s.empty() == false)
    {
        tp = s.top();
        s.pop();
        area_with_top = hist[tp] * (s.empty() ? i :
                            i - s.top() - 1);

        if (max_area < area_with_top)
            max_area = area_with_top;
    }

    return max_area;
}

// Driver program to test above function
int main()
{
    int hist[] = {6, 2, 5, 4, 5, 1, 6};
    int n = sizeof(hist)/sizeof(hist[0]);
    cout << "Maximum area is " << getMaxArea(hist, n);
    return 0;
}
```

5.3 Museum

```
/* error() function: */
int DEBUG_LINE = 0;
#define error(args...) { cout << "[DEBUG] Line " <<
     DEBUG_LINE++ << ": "; string _s = #args; err(_s, 0,
     args); }
void err(const string &name, int in) {}
template<typename T, typename... Args>
void err(const string &name, int in, T a, Args... args) {
    if (name[in] == ' ') in++; string curr_name;
    while (in < SZ(name) && name[in] != ',') {
        if (name[in] == 34 || name[in] == 39) {
            curr_name.pb(name[in]);
            char c = name[in++];
            while (name[in] != c) curr_name.pb(name[in++]);
        }
        curr_name.pb(name[in++]);
    }
    if (curr_name.back() == ' ') curr_name.pop_back();
        OUT(curr_name), OUT(" = "), OUT(a),
             OUT((sizeof...(args) ? ", " : "\n"));
        err(name, ++in, args...);
}


/* error() function */
template<typename T1, typename T2>
inline ostream& operator<<(ostream& os, pair<T1, T2> p) { os <<
     "{" << p.first << ", " << p.second << "}"; return os; }
template<typename T>
inline ostream& operator<<(ostream& os, vector<T>& a) { os <<
     "["; for (int i = 0; i < (int)a.size(); i++) { if (i) os
     << ", "; os << a[i]; } os << "]"; return os; }


#define error(args...) { string _s = #args; replace(_s.begin(),
     _s.end(), ',', ' '); stringstream _ss(_s);
     istream_iterator<string> _it(_ss); err(_it, args); }
void err(istream_iterator<string> it) {}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args) {
       cout << *it << " = " << a << endl;
       err(++it, args...);
}
```

5.4 Sorting

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

// When using sort(vii.begin(), vii.end()) for vector<pair<int,
     int> > vii,
// the pairs are compared lexicographically. As a result, the
     pairs with
// equal first elements are also sorted by their second element.
// But sometimes we might need to preserve the relative order
     of pairs with
// with the same first elements.
// This code sorts vii by their first element where the
// relative order of pairs with equal first elements is
     preserved.
// Complexity: nlogn
// Source: stackoverflow

struct compare_first_only {
    template<typename T1, typename T2>
    bool operator()(const std::pair<T1, T2>& p1, const
         std::pair<T1, T2>& p2) {
        return p1.first < p2.first;
    }
};

int main()
{
    vector<pair<int, int> > vii;
    stable_sort(vii.begin(), vii.end(), compare_first_only());

    return 0;
}
```

5.5 Template

```
#include<stdio.h>
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;


#define    UI                  unsigned int
#define    LL                  long long int
#define    LD                  long double
#define    ULL                 unsigned long long int
#define    VI                  vector<int>
#define    VB                  vector<bool>
#define    VLL                 vector<LL>
#define    VULL                vector<ULL>


#define    pii                 pair<int, int>
#define    mp                  make_pair
#define    pb                  push_back
#define    ff                  first
#define    ss                  second
#define    SZ(x)               (int)x.size()
#define    all(x)              x.begin(), x.end()


#define    sp(x)               setprecision(x) << fixed
#define    what_is(x)          cout << #x << " is " << x <<
     endl
```

```
#define    PI                    acos(-1.0)
#define    EPS                   1e-12
#define    S_SIZE                4000010    /* 3.81640625 mb */


const LL inf = 1000000000;
const LL mod = 1000000000 + 7;


inline void IO() { ios_base::sync_with_stdio(0); cin.tie(0);
     cout.tie(0); }
template<typename T> inline int dcmp (T x) { const T eps = EPS;
     return x < -eps ? -1 : (x > eps); }


template<class T> inline int CHECK(T MASK, int i) { return
     (MASK >> i) & 1; }
template<class T> inline T ON(T MASK, int i) { return MASK |
     (T(1) << i); }
template<class T> inline T OFF(T MASK, int i) { return MASK &
     (~(T(1) << i)); }
template<typename T> inline int CNT(T MASK) {
    if (numeric_limits<T>::digits <= numeric_limits<unsigned
         int>::digits) return __builtin_popcount(MASK);
    else return __builtin_popcountll(MASK);
}
template<class T> inline int RIGHT(T MASK) { return log2(MASK &
     -MASK); }


int dx4[] = { 0, 0, -1, +1 };
int dy4[] = { +1, -1, 0, 0 };
int dx8[] = { 1, 1, 0, -1, -1, -1, 0, 1, 0 };
int dy8[] = { 0, 1, 1, 1, 0, -1, -1, -1, 0 };
int dx8Knight[] = { +2, +2, +1, -1, -2, -2, -1, +1 };
int dy8Knight[] = { +1, -1, -2, -2, -1, +1, +2, +2 }; /*
     clockwise, starting from 3 o'clock */


inline void I(int &a) { scanf("%d", &a); }
inline void I(LL &a) { scanf("%lld", &a); }
inline void I(ULL &a) { scanf("%llu", &a); }
inline void I(char *a) { scanf("%s", a); }
char Iarr[S_SIZE]; inline void I(string &a) { scanf("%s",
     Iarr); a = Iarr; }
inline void I(LD &a) { cin >> a; }
inline void I(double &a) { cin >> a; }
inline void I(bool &a) { int aa; I(aa); a = aa; }
template<typename T1, typename T2> inline void I(pair<T1, T2>
     &a) { I(a.ff); I(a.ss); }
template<typename T> inline void I(vector<T> &a) { for (T &aa :
     a) I(aa); }
template<typename T, typename... Args> inline void I(T &a, Args
     &... args) { I(a); I(args...); }


inline void OUT(const int &a) { printf("%d", a); }
inline void OUT(const LL &a) { printf("%lld", a); }
inline void OUT(const ULL &a) { printf("%llu", a); }
inline void OUT(const char *a) { printf("%s", a); }
inline void OUT(const char &a) { printf("%c", a); }
inline void OUT(const string &a) { for (const char &aa : a)
     OUT(aa); }
inline void OUT(const bool &a) { printf("%d", a); }
template<typename T1, typename T2> inline void OUT(const
     pair<T1, T2> &a) { OUT("{"); OUT(a.ff); OUT(", ");
     OUT(a.ss); OUT("}"); }
template<typename T> inline void OUT(const T &a) { int i = 0;
     OUT("{"); for (const auto &aa : a) { if (i++) OUT(", ");
     OUT(aa); } OUT("}"); }
template<typename T, typename... Args> inline void OUT(const T
     &a, const Args &... args) { OUT(a); OUT(" ");
     OUT(args...); }
template<typename... Args> inline void O(const Args &... args)
     { OUT(args...); OUT("\n"); }


#define error(args...) { string _s = "[" + string(#args) + "] =
     ["; OUT(_s); err(args); }
void err() {}
template<typename T, typename... Args>
void err(T a, Args... args) {
       OUT(a), OUT((sizeof...(args) ? ", " : "]\n"));
       err(args...);
}


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 custom_hash_pair {
    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()(pair<uint64_t, uint64_t> x) const {
        static const uint64_t FIXED_RANDOM =
             chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x.ff + FIXED_RANDOM) * 3 +
             splitmix64(x.ss + FIXED_RANDOM);
    }
};


inline void faster(auto &unorderedMap, int n) { int num = 2;
     while (num < n) num *= 2; unorderedMap.reserve(num);
     unorderedMap.max_load_factor(0.25); }

// gp_hash_table<LL, int, custom_hash> table;
// unordered_map<LL, int, custom_hash> table;
// head


const int M = 200010; // maximum number of cubes


int main()
{


    return 0;
}
```

5.6 Trick - Matching Elements after Left Circular Shift

```
/*
    #1. The value of temp[i] indicates the number of matching
         elements (a[i] == b[i])
    #2. Here, hypothetical array a means array of (i, i + 1, i
         + 2, ..., (i + n - 1) % n) elements of curr
    #3. And, hypothetical array b means array of (0, 1, 2, ...,
         n - 1) elements of target
    #4. Assumption: Elements of target must be unique (not
         proven, just intuition)
    #5. Problems (easy to hard) - (role model submission /
         problem page):
        i.
              https://codeforces.com/contest/1365/submission/82814035
              (direct template)
        ii.
              https://codeforces.com/contest/1294/submission/82290705
              (direct template)
*/
int func(vector<int> curr, vector<int> target)
{
    gp_hash_table<int, int, custom_hash> pos;
    for (int i = 0; i < SZ(target); i++) {
        pos[target[i]] = i;
    }
    vector<int> temp(SZ(target), 0);
    for (int i = 0; i < SZ(curr); i++) {
        if (pos.find(curr[i]) != pos.end()) {
            int in = ((i-pos[curr[i]] % SZ(target)) +
                 SZ(target)) % SZ(target);
            temp[in]++;
        }
    }
    int ret = INT_MIN;
    for (int i = 0; i < SZ(temp); i++) {
        ret = max(ret, temp[i]);
    }
    return ret;
}
```

5.7 Trick - Maximum Subset Size such that for No Two Pairs, (x1 less$_e$qualx2, y1less$_e$qualy2)

```
/*
    #1. We have some pairs as elements {x, y} in a multiset
    #2. We want to know the maximum subset size such that for
          no two pairs of this subset, (x1 <= x2) and (y1 <= y2)
    #3. Solution:
        i. sort the multiset : sort(all(v))
        ii. answer is the longest strictly decreasing
              subsequence of v
        iii.O(nlogn)
    #4. Problems:
        i. https://codeforces.com/blog/entry/3781
        ii. https://vjudge.net/solution/26069618
*/
```

6 Number Theory

6.1 Fastest MulMods

```
/** The Fastest mulmod **/
long long int mulmod(long long int a,long long int b, long long
      MOD) {
   long double res = a;
   res *= b;
   long long int c = (long long)(res / MOD);
   a *= b;
   a -= c * MOD;
   a %= MOD;
   if (a < 0) a += MOD;
   return a;
}
/** The Fastest mulmod **/



/** The Second Fastest mulmod **/
long long modit(long long x,long long mod) {
       if(x>=mod) x-=mod;
       return x;
}
long long mult(long long x,long long y,long long mod) {
       long long s=0,m=x%mod;
       while(y) {
              if(y&1) s=modit(s+m,mod);
              y>>=1;
              m=modit(m+m,mod);
       }
       return s;
}
/** The Second Fastest mulmod **/
```

6.2 nCr Mod Anything

```
struct NCRMOD
{
    /* Tested with 1 <= mod <= 2 * 10e9 and 1 <= n <= 10e5 and
         0 <= m <= n. */
    /* MOD is fixed */
    typedef long long ll;

    #define pll pair<ll,ll>
    #define PB push_back
    #define MP make_pair
    #define N 100001

    int n, MOD, PHI, residue[N], fact[N], inv_fact[N];
    vector<int> primeDivisors;
    vector< vector<int> > C, P;

    int ModExp(int a, int n)
    {
        ll x = a % MOD, y = 1 % MOD;
        while(n) {
            if(n % 2)
                y = (x * y) % MOD;
            x = (x * x) % MOD;
            n /= 2;
        }
        return (int)y;
    }

    int ModInv(int a)
    {
        return ModExp(a, PHI - 1);
    }

    void PreProcess()
    {
        int m = MOD;
        for(int i = 2; i * i <= m; ++i) {
            if(m % i == 0) {
                while(m % i == 0)
                    m /= i;
                primeDivisors.PB(i);
            }
        }
        if(m > 1)
            primeDivisors.PB(m);
        m = primeDivisors.size();
        C.resize(m);
        P.resize(m);
        fact[0] = 1, inv_fact[0] = 1;
        for(int i = 1; i <= n; ++i)
            residue[i] = i;
        PHI = MOD;
        for(int i = 0; i < m; ++i) {
            int p = primeDivisors[i];
            PHI /= p;
            PHI *= (p - 1);
            C[i].resize(n + 1);
            for(int j = p; j <= n; j += p) {
                int x = residue[j], k = 0;
                while(x % p == 0) {
                    x /= p;
                    ++k;
                }
                residue[j] = x;
                C[i][j] = k;
            }
            for(int j = 1; j <= n; ++j)
                C[i][j] += C[i][j - 1];
            P[i].resize(C[i][n] + 1);
            P[i][0] = 1 % MOD;
            for(int j = 1; j < P[i].size(); ++j)
                P[i][j] = (1ll * p * P[i][j - 1]) % MOD;
        }
        for(int i = 1; i <= n; ++i)
            fact[i] = (1ll * residue[i] * fact[i - 1]) % MOD;
        inv_fact[n] = ModInv(fact[n]);
        for(int i = n - 1; i > 0; --i)
            inv_fact[i] = (1ll * residue[i + 1] * inv_fact[i +
                 1]) % MOD;
    }

    int NCR(int n, int r)
    {
        if(n < 0 || r < 0 || n < r)
            return 0;
        else {
            ll ans = fact[n];
            ans = (ans * inv_fact[r]) % MOD;
            ans = (ans * inv_fact[n - r]) % MOD;
            for(int i = 0; i < primeDivisors.size(); ++i) {
                int c = C[i][n] - C[i][r] - C[i][n - r];
                ans = (ans * P[i][c]) % MOD;
            }
            return (int)ans;
        }
    }
} obj;
```

6.3 nCr Mod Prime

```
class nCrModPrime
{
    ///    Time & space complexity: O(n)
    ///    mod has to be prime
private:
    int n;
    ULL* fact;
    ULL* inv;
    ULL* invFact;
    ULL primeMod;

public:
    nCrModPrime(int n, ULL m)
    {
```

```
        this->n = n;
        primeMod = m;
        fact = new ULL[n+1];
        inv = new ULL[n+1];
        invFact = new ULL[n+1];
        fact[0] = 1;
        for (int i = 1; i <= n; i++)
            fact[i] = (fact[i-1] * i) % m;
        num.genModInv(n, m, inv);
        num.genModInvFact(n, m, inv, invFact);
    }

    ULL nCr(int n, int r)
    {
        if (n < r) return 0;
        return (((fact[n] * invFact[r]) % primeMod) *
             invFact[n-r]) % primeMod;
    }
};
```

6.4 nCr without mod (n is fixed)

```
///*... nCr without mod ...*/

class nCrCalc
{
    ///     Time & space complexity: O(k), where k is
    ///     the maximum value of r in all nCr queries
private:
    ULL n;
    ULL* C;
    int last;

    gen(int st, int en)
    {
        for (int i = st; i <= en; i++) {
            C[i] = C[i-1] * (n - i + 1);
            C[i] /= i;
        }
        last = en;
    }

public:
    nCrCalc(ULL n, int MAX_K = -1)
    {
        this->n = n;
        if (MAX_K == -1) MAX_K = n;
        C = new ULL[MAX_K+1];
        C[0] = 1;
        last = 0;
    }

    ULL nCr(int k)
    {
        if (k > n - k) k = n - k;
        if (k > last) gen(last + 1, k);
        return C[k];
    }
};

///*... nCr without mod ...*/
```

6.5 Number Theory

```
struct NumberTheory {
    LL gcd(LL a, LL b) { while (b) { a %= b; swap(a, b); }
         return a; }
    LL lcm(LL a, LL b) { return (a / gcd(a, b)) * b; }
    LL bigMod(LL p, LL e, LL m) {
        LL ret = 1; p %= m;
        while (e > 0) { if (e & 1) ret = (ret * p) % m; p = (p
             * p) % m; e >>= 1; }
        return ret;
    }
    /// (1/a) % m when a & m are co-primes
    LL modInverse(LL a, LL m) {
        LL g = gcd(a, m);
        if (g != 1) {
            cout << "Inverse doesn't exist" << endl; return -1;
        }
        LL m0 = m;
        LL y = 0, x = 1;
        if (m == 1) return 0;
        while (a > 1) {
            LL q = a / m;
            LL t = m;
            m = a % m, a = t;
            t = y;
            y = x - q * y;
            x = t;
        }
        if (x < 0) x += m0;
        return x;
    }
    /// generate all (1/i) % m in [1..n] where n is less than m
    /// m must be prime
    void genModInv(int n, LL m, ULL inv[]) {
        inv[1] = 1;
        for (int i = 2; i <= n; ++i)
            inv[i] = (m - (m / i) * inv[m%i] % m) % m;
    }
    /// generate all (1/i!) % m in [1..n] where n is less than m
    /// m must be prime
    void genModInvFact(int n, LL m, ULL inv[], ULL invFact[]) {
        invFact[0] = invFact[1] = 1;
        for (int i = 2; i <= n; i++)
            invFact[i] = (invFact[i-1] * inv[i]) % m;
    }
} num;
```

6.6 Probability Expected Value - GamblersRuin

```
/*
    #1. Classical Definition: Two players begin with fixed
        stakes, transferring points until one or the other is
        "ruined" by getting to zero points.
    #2. A starts with a points and has probability p of taking
        1 point from B in each round.
    #3. B starts with b points and has probability q = (1 - p)
        of taking 1 point from A in each round.
    #4. Source Inspiration:
        i.
            http://www2.math.uu.se/~sea/kurser/stokprocmn1/slumpvandring
            (Theorem 4 & 5)
        ii. https://en.wikipedia.org/wiki/Gambler%27s_ruin
*/
struct GamblersRuin
{
    LD a, b;
    LD p, q;
    GamblersRuin(LL a, LL b, LD p = 0.5) : a(a), b(b), p(p),
        q(1 - p) {}
    LD probabilityOfAWinning() {
        if (dcmp(p - 0.5) == 0) return a / (a + b);
        else return (pow(q / p, a) - 1) / (pow(q / p, a + b) -
            1);
    }
    LD probabilityOfBWinning() {
        return 1 - probabilityOfAWinning();
    }
    LD expectedNumberOfRounds() {
        if (dcmp(p - 0.5) == 0) return a * b;
        else return (a / (q - p)) - (((a + b) / (q - p)) *
            probabilityOfAWinning());
    }
};
```

6.7 Rho, prime checking, factorization

```
/**
    Rho, prime checking, factorization
    https://codeforces.com/blog/entry/61901?#comment-458901
**/

namespace Big
{
    using u128 = __uint128_t;
    using i128 = __int128;
    using u64 = unsigned long long;
    using i64 = long long;

    const int C1 = 126;
    const int C2 = 64;
    static const int step = 1 << 9;
```

```
    const u128 prime[13] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
         31, 37, 41};
    const int psize = 13;
}

namespace Small
{
    using u128 = ULL;
    using i128 = LL;
    using u64 = unsigned int;
    using i64 = int;

    const int C1 = 126 / 2;
    const int C2 = 64 / 2;
    static const int step = 1 << 8;

    const u128 prime[7] = {2, 325, 9375, 28178, 450775,
         9780504, 1795265022};
    const int psize = 7;
}

using namespace Big; /// Small

namespace Factor
{
    int const num_tries = 10;
    u128 n, niv_n;

    void setn(u128 n_)
    {
        n = n_;
        niv_n = 1;
        u128 x = n;
        for (int i = 1; i <= C1; i++) {
            niv_n *= x;
            x *= x;
        }
    }

    u128 HI(u128 x) { return x >> C2; };
    u128 LO(u128 x) { return u64(x); };

    struct u256
    {
        u128 hi, lo;

        static u256 mul128(u128 x, u128 y)
        {
            u128 t1 = LO(x) * LO(y);
            u128 t2 = HI(x) * LO(y) + HI(y) * LO(x) + HI(t1);
            return {HI(x) * HI(y) + HI(t2), (t2 << C2) + LO(t1)};
        }
    };

    u128 mmul(u128 x, u128 y)
    {
        u256 m = u256::mul128(x, y);
        u128 ans = m.hi - u256::mul128(m.lo * niv_n, n).hi;
        if (i128(ans) < 0) ans += n;
        return ans;
    }

    inline u128 f(u128 x, u128 inc)
    {
        return mmul(x, x) + inc;
    }

    inline u128 gcd(u128 a, u128 b)
    {
        int shift = __builtin_ctzll(a | b);
        b >>= __builtin_ctzll(b);
        while (a) {
            a >>= __builtin_ctzll(a);
            if (a < b) swap(a, b);
            a -= b;
        }
        return b << shift;
    }

    inline u128 rho(u128 seed, u128 n, u64 inc)
    {
        setn(n);
        auto sub = [&] (u128 x, u128 y) { return x > y ? x - y
             : y - x; };
        u128 y = f(seed, inc), a = f(seed, inc);
        for (int l = 1; ; l <<= 1) {
            u128 x = y;
            for (int i = 1; i <= l; i++) y = f(y, inc);
            for (int k = 0; k < l; k += step) {
                int d = min(step, l - k);
                u128 g = seed, y0 = y;
                for (int i = 1; i <= d; i++) {
                    y = f(y, inc);
                    g = mmul(g, sub(x, y));
                }
                g = gcd(g,n);
                if (g == 1) continue;
                if (g != n) return g;
                u128 y = y0;
                while (gcd(sub(x, y), n) == 1) y = f(y, inc);
                return gcd(sub(x, y), n) % n;
            }
        }
        return 0;
    }

    mt19937_64 rd;

    u128 rho(u128 x)
    {
        if (x % 2 == 0) return 2;
        if (x % 3 == 0) return 3;
        uniform_int_distribution<u64> rng(2, u64(x) - 1);
        for (int i = 2; i < num_tries; i++) {
            u128 ans = rho(rng(rd), x, i);
            if (ans != 0 && ans != x) return ans;
        }
        return 0;
    }

    u128 factor(u128 x)
    {
        return rho(x);
    }
}

#define gc (c = getchar())
template <typename T>
void read(T &x)
{
    char c;
    while (gc < '0');
    x = c - '0';
    while (gc >= '0') x = x * 10 + c - '0';
}

template <typename T>
void write(T x, char c)
{
    static char st[40];
    int top = 0;
    do { st[++top] = '0' + x % 10; } while (x /= 10);
    do { putchar(st[top]); } while (--top);
    putchar(c);
}

u128 modit(u128 x, u128 mod)
{
        if (x >= mod) x -= mod;
        return x;
}

u128 mult(u128 x, u128 y, u128 mod) {
        u128 s = 0, m = x % mod;
        while (y) {
                if (y & 1) s = modit(s + m, mod);
                y >>= 1;
                m = modit(m + m, mod);
        }
        return s;
}

class MillerRabin
{
    private:
    u128 bigmod(u128 a, u128 p, u128 mod)
    {
        u128 x = a % mod, res = 1;
        while (p) {
            if (p & 1) res = mult(res, x, mod);
            x = mult(x, x, mod);
            p >>= 1;
        }
        return res;
    }

    bool witness(u128 a, u128 d, u128 s, u128 n)
    {
        u128 r = bigmod(a, d, n);
```

```
        if (r == 1 || r == n - 1) return false;
        int i;
        for (i = 0; i < s - 1; i++) {
            r = mult(r, r, n);
            if (r == 1) return true;
            if (r == n - 1) return false;
        }
        return true;
    }

    public:
    bool isPrime(u128 n)
    {
        if (n <= 1) return false;
        u128 p = n - 1, s = 0;
        while (!(p & 1)) {
            p /= 2;
            s++;
        }
        u128 d = p;
        p = n - 1;
        for (int i = 0; i < psize && prime[i] < n; i++) {
            if (witness(prime[i], d, s, n)) return false;
        }
        return true;
    }
} millerRabin;

const int factorizerConst = 350000;
struct Factorizer
{
    bitset<factorizerConst+1> flag;
    vector<u128> primes;
    vector<pair<u128, int> > factors;

    void init()
    {
        flag.set();
        for (u128 i = 2; i <= factorizerConst; i++) {
            if (flag[i]) {
                primes.pb(i);
                for (u128 j = i * i; j <= factorizerConst; j +=
                     i) {
                    flag[j] = 0;
                }
            }
        }
    }

    void clr()
    {
        factors.clear();
    }

    void rhoFactorize(u128 n)
    {
        while (n != 1) {
            if (millerRabin.isPrime(n)) {
                factors.pb(mp(n, 1));
                return;
            }
            u128 x = Factor::factor(n);
            if (!millerRabin.isPrime(x)) {
                u128 y = n / x;
                rhoFactorize(x);
                if (x != y) rhoFactorize(y);
                return;
            }
            int cnt = 0;
            while (n % x == 0) {
                n /= x;
                cnt++;
            }
            factors.pb(mp(x, cnt));
        }
    }

    void factorize(u128 n)
    {
        for (u128 p : primes) {
            if (p * p > n) break;
            if (n % p == 0) {
                int cnt = 0;
                while (n % p == 0) {
                    cnt++;
                    n /= p;
                }
                factors.pb(mp(p, cnt));
            }
        }
        rhoFactorize(n);
        sort(all(factors));
    }
};

/** Rho, prime checking, factorization **/
```

6.8 Vector Space Gaussian Elimination (Linear Algebra)

```
/*
    #1. Using Gaussian Elimination (basically concept of Vector
        Space and Vector Basis and how they can be used for
        XOR),
        we can solve many problems where subset XOR is the main
            theme.
    #2. Intuition (by order of importance):
        i. https://codeforces.com/blog/entry/68953
        ii.
             https://en.wikipedia.org/wiki/Basis_(linear_algebra)#:~:text
        iii.https://math.stackexchange.com/a/1054206
        iv.
             https://www.hackerearth.com/practice/notes/gaussian-elimina
        v. https://codeforces.com/blog/entry/60003
    #3. Problems (easy to hard) - (role model
          submission/problem page):
        i. https://vjudge.net/solution/26195313 (direct
             template)
*/
class VectorSpace
{
    /* It's a (Z^d)2 vector space */
    /* (d - 1) is the highest MSB possible (0 indexed) */
    int d;
    VULL basis_vectors;
    int basis_size = 0;
public:
    VectorSpace(int d = 64) : d(d), basis_vectors(d) {}
    void insertVector(ULL mask) {
        for (int i = d - 1; i >= 0; i--) {
            if (CHECK(mask, i)) {
                if (basis_vectors[i]) {
                    mask ^= basis_vectors[i];
                }
                else {
                    basis_vectors[i] = mask;
                    basis_size++;
                    return;
                }
            }
        }
    }
    /* Add one member function for every type of problem solved
         in the future */
    ULL maximumSubsetXOR() {
        ULL ret = 0;
        for (int i = d - 1; i >= 0; i--) {
            if (basis_vectors[i]) {
                if (!CHECK(ret, i)) ret ^= basis_vectors[i];
            }
        }
        return ret;
    }
    ULL possibleDistinctValuesUsingSubsetXOR() {
        return 1ULL << basis_size;
    }
};
```

7 String