Contents

1 Contest

1.1 Header

```cpp
#pragma once

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

template <class TH>
void _dbg(const char *sdbg, TH h) { cerr << sdbg << '=' << h << endl; }

template <class TH, class... TA>
void _dbg(const char *sdbg, TH h, TA... a)
{
  while (*sdbg != ',')
    cerr << *sdbg++;
  cerr << '=' << h << ',';
  _dbg(sdbg + 1, a...);
}

template <class L, class R>
ostream &operator<<(ostream &os, pair<L, R> p)
{
  return os << "(" << p.first << ", " << p.second << ")";
}

template <class Iterable, class = typename enable_if<!is_same<string,
   Iterable>::value>::type>
auto operator<<(ostream &os, Iterable v) -> decltype(os << *begin(v))
{
  os << "[";
  for (auto vv : v)
    os << vv << ", ";
  return os << "]";
}

#define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__)

typedef pair<int, int> pii;
typedef long long ll;
typedef long double ld;

const int inf = 0x3f3f3f3f;
const long long infll = 0x3f3f3f3f3f3f3f3fll;

#define sz(x) ((int)(x).size())

// Return 1 if x > 0, 0 if x == 0 and -1 if x < 0.
template <class T>
```

```
int sign(T x) { return (x > 0) - (x < 0); }

template <class T>
T abs(const T &x) { return (x < T(0)) ? -x : x; }

// Pretty good compilation command:
// g++ -g a.cpp --std=c++14 -Wall -Wextra -Wno-unused-result -Wconversion -
   Wfatal-errors -fsanitize=undefined,address -o a.out

// int main()
// {
//  cin.sync_with_stdio(0);
//  cin.tie(0);
//  cin.exceptions(cin.failbit);
// }
```

1.2 Sample Debug

```
32cfcc  #include "header.hpp"
d41d8c
13a4b1  int main(void)
f95b70  {
3e8410    int a = 11, b = 12, c = 13;
b9ee34    vector<vector<int>> v = {{a, b}, {c}, {0, 1}};
2a803c    set<int> s = {a, b};
6b3b13    map<double, int> m;
af2bd1    m[2.5] = 2;
37a428    m[-3.1] = 3;
d41d8c
632de9    map<string, int> tab;
88ec8d    tab["abc"] = (int) 'a' + 'b' + 'c';
69908e    tab["abz"] = (int) 'a' + 'b' + 'z';
bd6def    int array[3] = {1, 2, 5};
d41d8c
5939a6    debug(a, b, c);
fb9ee1    debug(v);
b45aab    debug(s, m);
3cf91d    debug(tab);
d95ee2    debug(array); // This one does not work.
cbb184  }
50cc4b
```

1.3 Hash Code

```
#!/bin/bash
# Hashes each line of a file, ignoring all whitespace and comments (multi-
   line comments will be bugged).

while IFS= read -r line; do # Loops lines of stdin.
  echo "$line" | cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum | cut
    -c-6 | tr -d '[:space:]';
```

```
  echo "  $line"; # Before $line is a tab.
done
```

2 Data Structures

2.1 BIT

```cpp
#include "../../contest/header.hpp"

/*
  BIT: element update, range sum query and sum lower_bound in O(log(N)).
  Represents an array of elements in range [1, N].
*/

template <class T>
struct bit
{
  int n, LOGN;
  vector<T> val;
  bit(int _n) : n(_n), LOGN(log2(n + 1)), val(_n + 1, 0) {}

  // val[pos] += x
  void update(int pos, T x)
  {
    for (int i = pos; i <= n; i += -i & i)
      val[i] += x;
  }

  // sum of range [1, pos]
  T query(int pos)
  {
    T retv = 0;
    for (int i = pos; i > 0; i -= -i & i)
      retv += val[i];
    return retv;
  }

  // min pos such that sum of [1, pos] >= sum, or n + 1 if none exists.
  int lower_bound(T x)
  {
    T sum = 0;
    int pos = 0;

    for (int i = LOGN; i >= 0; i--)
      if (pos + (1 << i) <= n && sum + val[pos + (1 << i)] < x)
        sum += val[pos += (1 << i)];

    return pos + 1; // pos will have position of largest value less than x.
  }
};
```

6acfcc

2.2 BIT2D

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
caf843    BIT: element update, range sum query in O(log(n) * log(m)). This
   can also be generalized for 3d.
a6cfe6    Represents a matrix of elements in range [1 ... n][1 ... m].
c4c9bd  */
d41d8c
4fce64  template <class T>
f6f3a7  struct bit2d
f95b70  {
14e0a7    int n, m;
f7ea55    vector<vector<T>> val;
9c8214    bit2d(int _n, int _m) : n(_n), m(_m), val(_n + 1, vector<T>(_m + 1,
    0)) {}
d41d8c
d41d8c    // val[i][j] += x
4460cb    void update(int r, int c, T x)
f95b70    {
9e45d9      for (int i = r; i <= n; i += -i & i)
13d333        for (int j = c; j <= m; j += -j & j)
ff237f          val[i][j] += x;
cbb184    }
d41d8c
d41d8c    // sum of positions (1 ... r, 1 ... c)
450f85    T query(int r, int c)
f95b70    {
56622d      T retv = 0;
bc7409      for (int i = r; i > 0; i -= -i & i)
d53722        for (int j = c; j > 0; j -= -j & j)
86df71          retv += val[i][j];
6272cf      return retv;
cbb184    }
d41d8c
d41d8c    // sum of positions (ri ... rf, ci ... cf). (1 <= ri <= rf <= n)
   and (1 <= ci <= cf <= m). TODO: test me.
bdc664    T query_rect(int ri, int ci, int rf, int cf)
f95b70    {
6072bc      return query(rf, cf) - query(rf, ci - 1) - query(ri - 1, cf) +
   query(ri - 1, ci - 1);
cbb184    }
2145c1  };
a4a33c
```

2.3 Dynamic Seg

```
5d1131  #include "../../contest/header.hpp"
```

```
d41d8c
d41d8c  /*
811629    Segment tree with dynamic memory allocation and arbitrary interval.
91eb69      Every operation is O(log(r-l))
b5a53f      Uses O(min(r-l, n*log(r-l)) memory, where n is the number of
   insertions.
d41d8c
ca2095    Constraints:
3dcfba      Segment tree range [l, r] must be such that 0 <= l <= r.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
4fce64  template<class T>
e4accb  struct node
f95b70  {
f48ea0    T val;
af32d9    node *left, *right;
d41d8c
995125    T get(int l, int r, int a, int b)
f95b70    {
47234b      if (l == a && r == b)
d943f4        return val;
814ad2      int mid = (l + 0ll + r) / 2;
f890f2      if (b <= mid)
ac57ce        return left ? left->get(l, mid, a, b) : 0;
a54f0c      else if (a > mid)
1c7837        return right ? right->get(mid + 1, r, a, b) : 0;
2954e9      else
9b1cb1        return (left ? left->get(l, mid, a, mid) : 0) + (right ? right
   ->get(mid + 1, r, mid + 1, b) : 0);
cbb184    }
d41d8c
14d5ea    void update(int l, int r, int a, T x)
f95b70    {
bd3398      if (l == r)
c43fe0        val = x;
2954e9      else
f95b70      {
814ad2        int mid = (l + 0ll + r) / 2;
a49729        if (a <= mid)
1ec55a          (left ? left : (left = new node()))->update(l, mid, a, x);
2954e9        else
92fe63          (right ? right : (right = new node()))->update(mid + 1, r, a,
    x);
d41d8c
dd51dd        val = (left ? left->val : 0) + (right ? right->val : 0);
cbb184      }
cbb184    }
2145c1  };
```

d41d8c
eef0d8

2.4 Linear Container

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
1ee0c3    Line Container (most common for convex hull trick). Amortized O(log
    N) per operation.
48cf95      Container where you can add lines of the form kx+m, and query
   maximum values at points x.
dc45cd      Useful for dynamic programming.
d41d8c
1d1558    Source: https://github.com/kth-competitive-programming/kactl/blob/
   master/content/contest/template.cpp
c4c9bd  */
d41d8c
3fe318  struct line
f95b70  {
3e2604    mutable ll k, m, p;
889941    bool operator<(const line &o) const { return k < o.k; }
abfd1f    bool operator<(ll x) const { return p < x; }
2145c1  };
d41d8c
0c8ce5  struct line_container : multiset<line, less<>>
f95b70  {
d41d8c    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
f5e3e7    const ll inf = LLONG_MAX;
d41d8c
9608c5    ll div(ll a, ll b)
f95b70    { // floored division
353cf0      return a / b - ((a ^ b) < 0 && a % b);
cbb184    }
d41d8c
9c092f    bool isect(iterator x, iterator y)
f95b70    {
f959d1      if (y == end())
f95b70      {
09a75e        x->p = inf;
d1fe4d        return false;
cbb184      }
3cca77      if (x->k == y->k)
83e301        x->p = x->m > y->m ? inf : -inf;
2954e9      else
b4284e        x->p = div(y->m - x->m, x->k - y->k);
870ec6      return x->p >= y->p;
cbb184    }
d41d8c
928f4b    void add(ll k, ll m)
```

```
f95b70    {
116e6c      auto z = insert({k, m, 0}), y = z++, x = y;
2d9d80      while (isect(y, z))
96cee5        z = erase(z);
d94b4e      if (x != begin() && isect(--x, y))
c07d21        isect(x, y = erase(y));
57dd20      while ((y = x) != begin() && (--x)->p >= y->p)
77462a        isect(x, erase(y));
cbb184    }
d41d8c
e8b5c2    ll query(ll x)
f95b70    {
229883      assert(!empty());
7d13b8      auto l = *lower_bound(x);
96a2bc      return l.k * x + l.m;
cbb184    }
2145c1  };
d41d8c
66b35a
```

2.5 Min Queue

```
d41d8c  /*
958401    max(min) queue with O(1) get_max(min).
d41d8c
f67dcb    Tips:
c53808      - Useful for sliding window 1D and 2D.
af9dc1      - For 2D problems, you will need to pre-compute another matrix,
55e3e9      by making a row-wise traversal, and calculating the min/max value
79c288      beginning in each cell. Then you just make a column-wise traverse
b21db2      as they were each an independent array.
c4c9bd  */
d41d8c
8f0a66  struct max_queue
f95b70  {
84841a    queue<ll> q;
889d23    deque<ll> s;
d41d8c
dbb27b    int size()
f95b70    {
593f12      return (int)q.size();
cbb184    }
d41d8c
a1fe24    void push(ll val)
f95b70    {
d41d8c      // while (!s.empty() && s.back() > val) -> for a min_queue
1cb658      while (!s.empty() && s.back() < val) // for a max_queue
342ca4        s.pop_back();
fcc849      s.push_back(val);
d41d8c
```

```
380c99      q.push(val);
cbb184    }
d41d8c
d99fc4    void pop()
f95b70    {
7a8432      ll u = q.front();
833270      q.pop();
d41d8c
de7036      if (!s.empty() && s.front() == u)
784c93        s.pop_front();
cbb184    }
d41d8c
ba28bf    ll get_max()
f95b70    {
eccd4b      return s.front(); // same for min and max queue
cbb184    }
d41d8c
2145c1  };
82549d
```

2.6 Persistent Seg

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
a032aa    Persistent Segment Tree:
115cd2      Segment tree that stores all previous versions of itself.
91eb69      Every operation is O(log(r-l))
ad671a      Uses O(n*log(r-l)) memory, where n is the number of updates.
d41d8c
b95cae    Usage:
aca1a7      A new root is created for every persistent update (p_update) and
   returned.
0d5abd      Queries can be performed on any root as if it were a usual
   segment tree.
61a20d      You should keep a list of roots. Something like:
072987        vector<node *> roots = {new node()};
bf8bc0        roots.push_back(p_update(roots.back(), 0, 2*MAXV, a[i] + MAXV,
   v + 1));
d41d8c
ca2095    Constraints:
3dcfba      Segment tree range [l, r] must be such that 0 <= l <= r.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
e4accb  struct node
f95b70  {
97f03f    int val;
af32d9    node *left, *right;
```

```
d41d8c
1f6b0f    node(int x=0) : val(x), left(NULL), right(NULL) {}
2f77b9    node(node *l, node *r) : left(l), right(r) { val = (left ? left->
   val : 0) + (right ? right->val : 0); }
d41d8c
f219f1    int get(int l, int r, int a, int b)
f95b70    {
47234b      if (l == a && r == b)
d943f4        return val;
814ad2      int mid = (l + 0ll + r) / 2;
f890f2      if (b <= mid)
ac57ce        return left ? left->get(l, mid, a, b) : 0;
a54f0c      else if (a > mid)
1c7837        return right ? right->get(mid + 1, r, a, b) : 0;
2954e9      else
9b1cb1        return (left ? left->get(l, mid, a, mid) : 0) + (right ? right
   ->get(mid + 1, r, mid + 1, b) : 0);
cbb184    }
2145c1  };
d41d8c
63f202  node *p_update(node *prev, int l, int r, int a, int x)
f95b70  {
bd3398    if (l == r)
13478f      return new node(x);
d41d8c
814ad2    int mid = (l + 0ll + r) / 2;
a49729    if (a <= mid)
b73799      return new node(p_update(prev ? prev->left : NULL, l, mid, a, x),
    prev ? prev->right : NULL);
2954e9    else
460332      return new node(prev ? prev->left : NULL, p_update(prev ? prev->
   right : NULL, mid + 1, r, a, x));
cbb184  }
d41d8c
707f69
```

2.7 Lazy Seg

```
2b74fa  #include <bits/stdc++.h>
ca417d  using namespace std;
d41d8c
d41d8c  /*
6f561b    Segment Tree with Lazy updates:
d8b1dc      Range update and range query in O(log(MAX_RANGE))
c329b0      Binary search on tree in O(log(MAX_RANGE))
05382c      Given as an example since it is not worth it to copy a generic
   tree during a contest.
d41d8c
e3c955    Solves: https://codeforces.com/contest/1179/problem/C
c4c9bd  */
```

```
d41d8c
ab0dbf  #define MAX_RANGE 1123456
d41d8c
fd87fe  int val[4 * MAX_RANGE];
802d92  int delta[4 * MAX_RANGE];
d41d8c
4ee394  #define left(i) ((i) << 1)
56e5cf  #define right(i) (((i) << 1) + 1)
d41d8c
0379af  void prop(int id, int l, int r)
f95b70  {
cfd4b4    if (l != r)
f95b70    {
d41d8c      // Updates need to be numerically stackable (e.g. not valid to
   have a list of updates).
df541b      delta[left(id)] += delta[id];
966351      delta[right(id)] += delta[id];
cbb184    }
d41d8c
21c2c8    val[id] += delta[id]; // Node value needs to be obtainable without
   propagating all the way to root.
0a8860    delta[id] = 0;
cbb184  }
d41d8c
d41d8c  // Sum x in all elements in range [a, b].
f2b4f2  void update(int id, int l, int r, int a, int b, int x)
f95b70  {
addc1f    if (a == l && b == r)
f95b70    {
d50197      delta[id] += x;
b62cfe      prop(id, l, r);
cbb184    }
2954e9    else
f95b70    {
b62cfe      prop(id, l, r);
ae007b      int mid = (l + r) / 2;
f890f2      if (b <= mid)
f95b70      {
6dbd37        update(left(id), l, mid, a, b, x);
384ec5        prop(right(id), mid + 1, r);
cbb184      }
a54f0c      else if (a > mid)
f95b70      {
859d13        update(right(id), mid + 1, r, a, b, x);
221ad0        prop(left(id), l, mid);
cbb184      }
2954e9      else
f95b70      {
fc79c7        update(left(id), l, mid, a, mid, x);
04c83e        update(right(id), mid + 1, r, mid + 1, b, x);
```

```
cbb184      }
d41d8c
caf644      val[id] = min(val[left(id)], val[right(id)]);
cbb184    }
cbb184  }
d41d8c
d41d8c  // Get the minimum value in range [a, b].
9fed20  int get(int id, int l, int r, int a, int b)
f95b70  {
b62cfe    prop(id, l, r);
addc1f    if (a == l && b == r)
a0328b      return val[id];
2954e9    else
f95b70    {
ae007b      int mid = (l + r) / 2;
f890f2      if (b <= mid)
c55f80        return get(left(id), l, mid, a, b);
a54f0c      else if (a > mid)
26dd34        return get(right(id), mid + 1, r, a, b);
2954e9      else
5e3fad        return min(get(left(id), l, mid, a, mid), get(right(id), mid +
   1, r, mid + 1, b));
cbb184    }
cbb184  }
d41d8c
d41d8c  // Find index of rightmost element which is less than x. (works
   because this is a seg of min)
0529b3  int bsearch(int id, int l, int r, int x)
f95b70  {
b62cfe    prop(id, l, r);
d41d8c
bd3398    if (l == r)
f7d2ed      return (val[id] < x) ? l : -1;
2954e9    else
f95b70    {
ae007b      int mid = (l + r) / 2;
221ad0      prop(left(id), l, mid);
384ec5      prop(right(id), mid + 1, r);
f01b35      if (val[right(id)] < x)
018a94        return bsearch(right(id), mid + 1, r, x);
2954e9      else
bad725        return bsearch(left(id), l, mid, x);
cbb184    }
cbb184  }
d41d8c
1037bf  #define MAXN 312345
d41d8c
a58cd5  int a[MAXN];
c4b25f  int b[MAXN];
d41d8c
```

```
13a4b1  int main(void)
f95b70  {
b067b3    int n, m, q, tp, x, y;
d69917    scanf("%d %d", &n, &m);
5359f3    for (int i = 1; i <= n; i++)
f95b70    {
9376f3      scanf("%d", &a[i]);
49e934      update(1, 1, 1000000, 1, a[i], -1);
cbb184    }
d41d8c
8eae24    for (int i = 1; i <= m; i++)
f95b70    {
264aeb      scanf("%d", &b[i]);
472fcc      update(1, 1, 1000000, 1, b[i], 1);
cbb184    }
d41d8c
4aaeab    scanf("%d", &q);
a953ae    while (q--)
f95b70    {
960099      scanf("%d %d %d", &tp, &x, &y);
abc772      if (tp == 1)
f95b70      {
996a9b        update(1, 1, 1000000, 1, a[x], 1);
e603e6        a[x] = y;
28cfa0        update(1, 1, 1000000, 1, a[x], -1);
cbb184      }
2954e9      else
f95b70      {
8dbabe        update(1, 1, 1000000, 1, b[x], -1);
0464a9        b[x] = y;
bc18aa        update(1, 1, 1000000, 1, b[x], 1);
cbb184      }
d41d8c
584906      int tmp = bsearch(1, 1, 1000000, 0);
d41d8c
d41d8c      // Test of get and bsearch. Make sure all to the right are non-
   negative.
5a5bec      if (tmp != 1000000)
5df0f6        assert(get(1, 1, 1000000, tmp == -1 ? 1 : (tmp + 1), 1000000)
   >= 0);
c3e568      if (tmp != -1)
1d95f2        assert(get(1, 1, 1000000, tmp, tmp) < 0);
d41d8c
b03a7a      printf("%d\n", tmp);
cbb184    }
cbb184  }
90a905
```

2.8 Key Treap

```
#include "../../contest/header.hpp"

/*
  Treap:
    This treap implements something like a c++ set with additional operations: find the k-th element and count elements less than a given value.

  Time: O(log N) per operation.

  Author: Arthur Pratti Dadalto
*/

namespace treap
{
struct node
{
  int val; // node key.
  int p;   // node heap priority.
  int num; // node subtree size.
  node *left, *right;

  node(int _val) : val(_val), p(rand()), num(1), left(NULL), right(NULL) {}
};

int get_num(node *root)
{
  return (root == NULL) ? 0 : root->num;
}

void update_num(node *root)
{
  root->num = get_num(root->left) + get_num(root->right) + 1;
}

node *rotate_left(node *root)
{
  node *a = root;
  node *b = root->right;

  a->right = b->left;
  b->left = a;
  update_num(a);
  update_num(b);
  return b;
}

node *rotate_right(node *root)
{
```

```
d25f1b    node *a = root;
eb0328    node *b = root->left;
d41d8c
a09684    a->left = b->right;
7352c4    b->right = a;
a5e0c3    update_num(a);
2b11db    update_num(b);
73f89f    return b;
cbb184  }
d41d8c
d41d8c  // Insert new node with key x in treap rooted at root if not already
   there.
960bce  node *insert(node *root, int x)
f95b70  {
0edbc9    if (root == NULL)
13478f      return new node(x);
6b2a0b    if (x > root->val)
34c9df      root->right = insert(root->right, x);
ba0dc8    else if (x < root->val)
12f5b5      root->left = insert(root->left, x);
d41d8c
622638    update_num(root);
d41d8c
4f4bcf    if (root->right && root->right->p > root->p)
04107a      root = rotate_left(root);
c93ea7    if (root->left && root->left->p > root->p)
3f3108      root = rotate_right(root);
e2fc54    return root;
cbb184  }
d41d8c
d41d8c  // Remove node with key x in treap rooted at root if present.
d0ba77  node *remove(node *root, int x)
f95b70  {
0edbc9    if (root == NULL)
ea9b0a      return NULL;
6b2a0b    if (x > root->val)
fed39a      root->right = remove(root->right, x);
ba0dc8    else if (x < root->val)
6cf773      root->left = remove(root->left, x);
fb8e77    else if (root->left == NULL)
4de2d2      root = root->right;
a15580    else if (root->right == NULL)
2d4ff4      root = root->left;
386129    else if (root->left->p > root->right->p)
f95b70    {
3f3108      root = rotate_right(root);
fed39a      root->right = remove(root->right, x);
cbb184    }
2954e9    else
f95b70    {
```

```
04107a      root = rotate_left(root);
6cf773      root->left = remove(root->left, x);
cbb184    }
e6a2b0    if (root)
622638      update_num(root);
e2fc54    return root;
cbb184  }
d41d8c
d41d8c  // Return the k-th smallest element in tree rooted at root.
3576ec  int kth(node *root, int k)
f95b70  {
f9e30a    if (get_num(root->left) >= k)
7473ee      return kth(root->left, k);
f3e79f    else if (get_num(root->left) + 1 == k)
ae0ddc      return root->val;
2954e9    else
235aa0      return kth(root->right, k - get_num(root->left) - 1);
cbb184  }
d41d8c
d41d8c  // Return the number of elements smaller than x in tree rooted at
   root.
194e12  int count(node *root, int x)
f95b70  {
0edbc9    if (root == NULL)
bb30ba      return 0;
83010a    if (x < root->val)
da7c4c      return count(root->left, x);
08e5c0    else if (x == root->val)
140f45      return get_num(root->left);
2954e9    else
b73a02      return get_num(root->left) + 1 + count(root->right, x);
cbb184  }
cbb184  } // namespace treap
85f362
```

2.9 Sequential Treap

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
1977a5    Treap:
5c39c7      A short self-balancing tree. It acts as a sequential container
   with log-time splits/joins, and
df7261      is easy to augment with additional data.
d41d8c
4c88cf    Time: O(log N) per operation.
d41d8c
ca2095    Constraints:
c1b810      Acts as a vector of size N, with positions in range [0, N-1].
d41d8c
```

```
1d1558    Source: https://github.com/kth-competitive-programming/kactl/blob/
   master/content/data-structures/Treap.h
d41d8c
b95cae    Usage:
24eb84      To insert elements, create one node treaps. (e.g. treap::ins(root
   , new treap::node(x), i))
acfc60      To augment with extra data you should mostly add stuff to the
   recalc function. (e.g. to make it work like a seg tree)
03bb33      See applications for more usage examples.
c4c9bd  */
d41d8c
41c55a  namespace treap
f95b70  {
e4accb  struct node
f95b70  {
8f5901    node *l = 0, *r = 0;
97f03f    int val;   // Any value associated with node.
ee1179    int p;   // Node heap priority.
c6aff2    int c = 1; // Node subtree size.
674490    node(int val) : val(val), p(rand()) {}
86d631    void recalc();
2145c1  };
d41d8c
853943  int cnt(node *n) { return n ? n->c : 0; }
9af082  void node::recalc() { c = cnt(l) + cnt(r) + 1; }
d41d8c
d41d8c  // Apply function f on each tree node in order.
044d82  template <class F>
d5442c  void each(node *n, F f)
f95b70  {
f63660    if (n)
f95b70    {
cbc351      each(n->l, f);
ed31a5      f(n->val);
f5ab50      each(n->r, f);
cbb184    }
cbb184  }
d41d8c
d41d8c  // Split treap rooted at n in two treaps containing positions [0, k)
   and [k, ...)
de9c69  pair<node *, node *> split(node *n, int k)
f95b70  {
a020ba    if (!n)
e70a07      return {NULL, NULL};
9416bd    if (cnt(n->l) >= k) // "n->val >= k" for lower_bound(k)
f95b70    {
215a80      auto pa = split(n->l, k);
f3cfa7      n->l = pa.second;
2f09c0      n->recalc();
c05937      return {pa.first, n};
```

```
cbb184    }
2954e9    else
f95b70    {
7c23f0      auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
d37e77      n->r = pa.first;
2f09c0      n->recalc();
7af31a      return {n, pa.second};
cbb184    }
cbb184  }
d41d8c
d41d8c  // Merge treaps l and r keeping order (l first).
7f5419  node *merge(node *l, node *r)
f95b70  {
0c92a8    if (!l)
4c1f3c      return r;
6bf95d    if (!r)
792fd4      return l;
a0ade2    if (l->p > r->p)
f95b70    {
ed7b68      l->r = merge(l->r, r);
bf6a1f      l->recalc();
792fd4      return l;
cbb184    }
2954e9    else
f95b70    {
654f23      r->l = merge(l, r->l);
cda92d      r->recalc();
4c1f3c      return r;
cbb184    }
cbb184  }
d41d8c
d41d8c  // Insert treap rooted at n into position pos of treap rooted at t.
3fc637  node *ins(node *t, node *n, int pos)
f95b70  {
ca9a9f    auto pa = split(t, pos);
cc8215    return merge(merge(pa.first, n), pa.second);
cbb184  }
d41d8c
d41d8c  // Remove node at position pos from treap rooted at t.
1e0b32  node *rem(node *t, int pos)
f95b70  {
abdf75    node *a, *b, *c;
cf9546    tie(a, b) = split(t, pos);
0052e9    tie(b, c) = split(b, 1);
d41d8c
625cf2    delete b;
a300e4    return merge(a, c);
cbb184  }
d41d8c
d41d8c  // Example application: do a query in range [l, r].
```

```
0475c8  node *query(node *t, int l, int r)
f95b70  {
abdf75    node *a, *b, *c;
a8341d    tie(a, b) = split(t, l);
89f194    tie(b, c) = split(b, r - l + 1);
d41d8c
d41d8c    // printf("%lld\n", b->tab);
d41d8c
53aa0f    return merge(merge(a, b), c);
cbb184  }
d41d8c
d41d8c  // Example application: move the range [l, r) to index k.
b51124  void move(node *&t, int l, int r, int k)
f95b70  {
abdf75    node *a, *b, *c;
a8341d    tie(a, b) = split(t, l);
e81a2b    tie(b, c) = split(b, r - l);
1527bb    if (k <= l)
eeb6c2      t = merge(ins(a, b, k), c);
2954e9    else
646d6a      t = merge(a, ins(c, b, k - r));
cbb184  }
cbb184  } // namespace treap
02c35c
```

3 Geometry

3.1 2D

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  // 2D geometry operations. This file should not have algorithms.
d41d8c  // Author: some of it by Arthur Pratti Dadalto.
d41d8c  // Source: some of it from https://github.com/kth-competitive-
   programming/kactl/blob/master/content/geometry/.
d41d8c  // Usage: avoid int unless necessary.
d41d8c
d41d8c  // When increasing EPS, keep in mind that sqrt(1e9^2 + 1) = 1e9 + 5e
   -10.
22c921  const double EPS = 1e-12;
d41d8c
d41d8c  // Point struct implementation. Some methods are useful only when
   using this to represent vectors.
4fce64  template <class T>
4befb0  struct point
f95b70  {
5dcf91    typedef point<T> P;
645c5d    T x, y;
d41d8c
571f13    explicit point(T x = 0, T y = 0) : x(x), y(y) {}
0d0d56    bool operator<(P p) const { return tie(x, y) < tie(p.x, p.y); }
ec7475    bool operator==(P p) const { return tie(x, y) == tie(p.x, p.y); }
2798c7    P operator+(P p) const { return P(x + p.x, y + p.y); }
40d57e    P operator-(P p) const { return P(x - p.x, y - p.y); }
e03fa4    P operator*(T d) const { return P(x * d, y * d); }
0b99e8    P operator/(T d) const { return P(x / d, y / d); }
57bee4    T dot(P p) const { return x * p.x + y * p.y; }
460881    T cross(P p) const { return x * p.y - y * p.x; }
b3fab9    T cross(P a, P b) const { return (a - *this).cross(b - *this); } //
    product sign: right hand rule from a to b.
f681d2    T dist2() const { return x * x + y * y; }            // Distance
   squared to origin.
18b7a8    double dist() const { return sqrt((double)dist2()); }      //
   Vector norm (distance to origin).
9073ff    double angle() const { return atan2(y, x); }           // angle to
   x-axis in interval [-pi, pi]
6f5d42    point<double> unit() const { return *this / dist(); }      // makes
    dist()=1 (unit vector).
200c8f    P perp() const { return P(-y, x); }                // rotates +90
   degrees around origin.
567be8    point<double> normal() const { return perp().unit(); }       //
   perpendicular unit vector.
82fcdd    point<double> rotate(double a) const               // returns point
    rotated 'a' radians ccw around the origin.
f95b70    {
```

```
80d6a0      return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
cbb184    }
8ad6e5    double angle(P p) const { return p.rotate(-angle()).angle(); } //
   Angle between the vectors in interval [-pi, pi]. Positive if p is ccw from
    this.
2145c1  };
d41d8c
d41d8c  // Solves the linear system {a * x + b * y = e
d41d8c  //                          {c * x + d * y = f
d41d8c  // Returns {1, {x, y}} if solution is unique, {0, {0,0}} if no
   solution and {-1, {0,0}} if infinite solutions.
d41d8c  // If using integer function type, this will give wrong answer if
   answer is not integer.
d41d8c  // TODO: test me with integer and non-integer.
4fce64  template <class T>
562c39  pair<int, point<T>> linear_solve2(T a, T b, T c, T d, T e, T f)
f95b70  {
468cb9    point<T> retv;
256940    T det = a * d - b * c;
d41d8c
57f40d    if (det == 0) // Maybe do EPS compare if using floating point.
f95b70    {
cdd981      if (b * f == d * e && a * f == c * e)
3d7337        return {-1, point<T>()};
37dde3      return {0, point<T>()};
cbb184    }
d41d8c
d41d8c    // In case solution needs to be integer, use something like the
   line below.
d41d8c    // assert((e * d - f * b) % det == 0 && (a * f - c * e) % det == 0)
   ;
d41d8c
848480    return {1, point<T>((e * d - f * b) / det, (a * f - c * e) / det)};
cbb184  }
d41d8c
d41d8c  // Represents line segments defined by two points.
4fce64  template <class T>
4b2ec6  struct segment
f95b70  {
5dcf91    typedef point<T> P;
efb78f    P pi, pf; // Initial and final points.
d41d8c
a76c62    explicit segment(P a = P(), P b = P()) : pi(a), pf(b) {}
d41d8c
d41d8c    // Distance from this segment to a given point. TODO: test me.
325177    double dist(P p)
f95b70    {
58fd41      if (pi == pf)
adefd2        return (p - pi).dist();
96a4f0      auto d = (pf - pi).dist2();
```

```
486c32      auto t = min(d, max(.0, (p - pi).dot(pf - pi)));
5dab06      return ((p - pi) * d - (pf - pi) * t).dist() / d;
cbb184    }
d41d8c
d41d8c    // Checks if given point belongs to segment. Use dist(p) <= EPS
   instead when using point<double>.
0e3dba    bool on_segment(P p)
f95b70    {
50f719      return p.cross(pi, pf) == 0 && (pi - p).dot(pf - p) <= 0;
cbb184    }
d41d8c
d41d8c    // If a unique intersection point between the line segments exists
   then it is returned.
d41d8c    // If no intersection point exists an empty vector is returned.
d41d8c    // If infinitely many exist a vector with 2 elements is returned,
   containing the endpoints of the common line segment.
d41d8c    // The wrong position will be returned if P is point<ll> and the
   intersection point does not have integer coordinates.
d41d8c    // However, no problem in using it to check if intersects or not in
    this case (size of vector will be correct).
d41d8c    // Products of **three** coordinates are used in intermediate steps
    so watch out for overflow if using int or long long.
f3f800    vector<P> intersect(segment rhs)
f95b70    {
9b1730      auto oa = rhs.pi.cross(rhs.pf, pi), ob = rhs.pi.cross(rhs.pf, pf)
   ,
1d46ec         oc = pi.cross(pf, rhs.pi), od = pi.cross(pf, rhs.pf);
d41d8c
d41d8c      // Checks if intersection is single non-endpoint point.
288e4c      if (sign(oa) * sign(ob) < 0 && sign(oc) * sign(od) < 0)
655339        return {(pi * ob - pf * oa) / (ob - oa)};
d41d8c
4c122f      set<P> s;
0373dd      if (rhs.on_segment(pi))
f07e25        s.insert(pi);
6725fe      if (rhs.on_segment(pf))
3c93ab        s.insert(pf);
3ad8fc      if (on_segment(rhs.pi))
522b2f        s.insert(rhs.pi);
f425cd      if (on_segment(rhs.pf))
d1c5a5        s.insert(rhs.pf);
d2dd66      return vector<P>(s.begin(), s.end());
cbb184    }
2145c1  };
d41d8c
d41d8c  // Represents a line by its equation in the form a * x + b * y = c.
d41d8c  // Can be created from two points or directly from constants.
4fce64  template <class T>
3fe318  struct line
f95b70  {
```

```
5dcf91    typedef point<T> P;
52d831    T a, b, c; // line a * x + b * y = c
d41d8c
f4f0fd    explicit line(P p1, P p2) // TODO: test me.
f95b70    {
4c2f1e      assert(!(p1 == p2));
6a88e5      a = p2.y - p1.y;
82330e      b = p1.x - p2.x;
cfae8e      c = a * p1.x + b * p1.y;
d41d8c
d41d8c      // In case of int, it is useful to scale down by gcd (e.g to use
   in a set).
d41d8c      // Might be useful to normalize here.
cbb184    }
d41d8c
510551    explicit line(T _a, T _b, T _c) : a(_a), b(_b), c(_c) {}
d41d8c
d41d8c    // Distance from this line to a given point. TODO: test me.
325177    double dist(P p)
f95b70    {
d37216      return abs(a * p.x + b * p.y - c) / sqrt((double)(a * a + b * b))
   ;
cbb184    }
d41d8c
d41d8c    // Intersects this line with another given line. See linear_solve2
   for usage. TODO: test me.
4a5d8e    pair<int, P> intersect(line rhs)
f95b70    {
6c76dc      return linear_solve2(a, b, rhs.a, rhs.b, c, rhs.c);
cbb184    }
d41d8c
d41d8c    // Normalize line to c >= 0, a*a + b*b == 1. Only use with double.
050345    line normalize()
f95b70    {
22b5e2      double d = P(a, b).dist() * (c < 0 ? -1 : 1);
7c9abe      return line(a / d, b / d, c / d);
cbb184    }
2145c1  };
d41d8c
d41d8c  // Represents a circle by its center and radius. Mostly only works
   with double.
4fce64  template <class T>
0b1113  struct circle
f95b70  {
5dcf91    typedef point<T> P;
1ab228    P center;
c3df30    T r;
d41d8c
d41d8c    // Intersects circle with a given line. This does not work with
   integer types.
```

```
d41d8c    // If there is no intersection, returns 0 and retv is whatever.
d41d8c    // If intersection is a single point, returns 1 and retv is a pair
   of equal points.
d41d8c    // If intersection is two points, return 2 and retv is the two
   intersection points.
d41d8c    // Assume points are given in no particular order. If you really
   need it, should be leftmost first when looking from center of the circle.
ec2c6b    int intersect(line<T> l, pair<P, P> &retv)
f95b70    {
800175      l = l.normalize();
f543ca      l.c -= l.a * center.x + l.b * center.y; // Recenter so that we
   can consider circle center in origin.
18b956      P v(l.a, l.b);
cf8231      P p0 = v * l.c; // p0 is the point in the line closest to origin.
d41d8c
2d9566      if (p0.dist() > r + EPS) // No intersection.
bb30ba        return 0;
40b0e2      else if (p0.dist() > r - EPS) // dist in [r - EPS, r + EPS] ->
   single point intersection at p0.
f95b70      {
de0c90        retv = {p0, p0};
6a5530        return 1;
cbb184      }
d41d8c
85b09c      double d = sqrt(r * r - l.c * l.c); // d is distance from p0 to
   the intersection points.
c4bf3f      retv = {center + p0 + v.normal() * d, center + p0 - v.normal() *
   d};
18b932      return 2;
cbb184    }
d41d8c
d41d8c    // Intersects circle with another circle. This does not work with
   integer types.
d41d8c    // This assumes the circles do not have the same center. Check this
    case if needed, can have 0 or infinite intersection points.
d41d8c    // If there is no intersection, returns 0 and retv is whatever.
d41d8c    // If intersection is a single point, returns 1 and retv is a pair
   of equal points.
d41d8c    // If intersection is two points, return 2 and retv is the two
   intersection points.
d41d8c    // Assume points are given in no particular order. If you really
   need it, should be leftmost first when looking from center of the rhs
   circle.
f2bab0    int intersect(circle rhs, pair<P, P> &retv)
f95b70    {
db42cd      rhs.center = rhs.center - center;
2adf3a      int num = rhs.intersect(line<T>(2 * rhs.center.x, 2 * rhs.center.
   y, rhs.center.x * rhs.center.x + rhs.center.y * rhs.center.y + r * r - rhs
   .r * rhs.r), retv);
2a6a69      retv.first = retv.first + center;
```

```
e34010      retv.second = retv.second + center;
fcc01b      return num;
cbb184    }
d41d8c
d41d8c    // Returns a pair of the two points on the circle whose tangent
   lines intersect p.
d41d8c    // If p lies within the circle NaN-points are returned. P is
   intended to be Point<double>.
d41d8c    // The first point is the one to the right as seen from the point p
    towards the circle.
163627    pair<P, P> tangents(P p)
f95b70    {
75ad6b      p = p - center;
28b73b      double k1 = r * r / p.dist2();
f84c08      double k2 = sqrt(k1 - k1 * k1);
a64b03      return {center + p * k1 + p.perp() * k2, center + p * k1 - p.perp
   () * k2};
cbb184    }
d41d8c
d41d8c    // TODO: find pair of tangent lines passing two circles.
2145c1  };
d41d8c
d41d8c  // The circumcircle of a triangle is the circle intersecting all
   three vertices.
d41d8c  // Returns the unique circle going through points A, B and C (given
   in no particular order).
d41d8c  // This assumes that the triangle has non-zero area.
d41d8c  // TODO: test specifically.
11308f  circle<double> circumcircle(const point<double> &A, const point<
   double> &B, const point<double> &C)
f95b70  {
b10dc9    circle<double> retv;
6d2418    point<double> a = C - B, b = C - A, c = B - A;
1d9440    retv.r = a.dist() * b.dist() * c.dist() / abs(c.cross(b)) / 2;
0d1695    retv.center = A + (b * c.dist2() - c * b.dist2()).perp() / b.cross(
   c) / 2;
6272cf    return retv;
cbb184  }
af0eba
```

3.2 Graham Scan (convex hull)

```
d41d8c  /*
6ccc59    Solution for convex hull problem (minimum polygon covering a set of
    points) based on ordering ponts by angle.
3248ac    * Finds the subset of points in the convex hull in O(Nlog(N)).
687c39    * This version works if you either want intermediary points in
   segments or not (see comments delimited by //)
01b744    * This version works when all points are collinear
246d86    * This version works for repeated points if you add a label to
```

```
   struct, and use this label in overloaded +, - and =.
d41d8c
1d1558    Source: https://github.com/kth-competitive-programming/kactl/blob/
   master/content/contest/template.cpp
c4c9bd  */
d41d8c
2b74fa  #include<bits/stdc++.h>
d41d8c
ad1153  typedef long long ll;
d41d8c
ca417d  using namespace std;
d41d8c
67a100  template<typename T>
4befb0  struct point
f95b70  {
5dcf91    typedef point<T> P;
645c5d    T x, y;
d41d8c
571f13    explicit point(T x = 0, T y = 0) : x(x), y(y) {}
d41d8c    //Double version: bool operator<(P p) const { return  fabs(x - p.x)
    < EPS ? y < p.y : x < p.x; }
0d0d56    bool operator<(P p) const { return tie(x, y) < tie(p.x, p.y); }
d41d8c    //Double version: bool operator==(P p) const { return fabs(x - p.x)
    < EPS && fabs(y - p.y) < EPS; }
ec7475    bool operator==(P p) const { return tie(x, y) == tie(p.x, p.y); }
2798c7    P operator+(P p) const { return P(x + p.x, y + p.y); }
f681d2    T dist2() const { return x*x + y*y; }
40d57e    P operator-(P p) const { return P(x - p.x, y - p.y); }
57bee4    T dot(P p) const { return x * p.x + y * p.y; }
460881    T cross(P p) const { return x * p.y - y * p.x; }
b3fab9    T cross(P a, P b) const { return (a - *this).cross(b - *this); }
5b4ebf    long double dist() const { return sqrt((long double)dist2()); }
2145c1  };
d41d8c
d41d8c  /*Compara primeiro por angulo em relacao a origem e depois por
   distancia para a origem*/
67a100  template<typename T>
d74eff  bool cmp(point<T> a, point<T> b){
a9b570    if(a.cross(b) != 0)
c33606      return a.cross(b) > 0;
ba7b3a    return a.dist2() < b.dist2();
cbb184  }
d41d8c
67a100  template<typename T>
3c7876  vector<point<T> > CH(vector<point<T> > points){
d41d8c    /*Encontra pivo (ponto extremos que com ctz faz parte do CH)*/
95b799    point<T> pivot = points[0];
e409fb    for(auto p : points)
e01c07      pivot = min(pivot, p);
d41d8c
```

```
d41d8c    /*Desloca conjunto para pivo ficar na origem e ordena potos pelo
   angulo e distancia do pivo*/
9ac126    for(int i = 0; i < (int) points.size(); i++)
3010bd      points[i] = points[i] - pivot;
d41d8c
e2c4e0    sort(points.begin(), points.end(), cmp<ll>);
d41d8c
9ac126    for(int i = 0; i < (int) points.size(); i++)
eda5a9      points[i] = points[i] + pivot;
d41d8c
d41d8c    /*Ponto extra para fechar o poligono*/
36b3da    points.push_back(points[0]);
d41d8c
620533    vector<point<T> > ch;
d41d8c
b7f960    for(auto p : points){
d41d8c      /*Enquanto o proximo ponto gera uma curva para a direita, retira
   ultimo ponto atual*/
d41d8c      /*Segunda comparaÃğÃčo serve para caso especial de pontos
   colineares quando se quer eliminar os intermediarios*/
d41d8c      //Trocar terceira comparacao pra <= para discartar pontos do meio
    de arestas no ch
d41d8c      //Double: trocar terceira comparaÃğÃčo por < EPS (descarta pontos
    em arestas) ou < -EPS (mantem pinto em aresta
29fcb4      while(ch.size() > 1 && !(p == ch[ch.size() - 2]) && ch[ch.size()
   - 2].cross(ch[ch.size() - 1] , p) < 0)
9d9654        ch.pop_back();
d2ebaf      ch.push_back(p);
cbb184    }
d41d8c
d41d8c    /*Elimina ponto extra*/
9d9654    ch.pop_back();
d41d8c
66cc3c    return ch;
cbb184  }
d41d8c
e8d76f  int main(){
1a88fd    int n;
f4c120    scanf("%d", &n);
76374e    vector<point<ll> > p(n);
d41d8c
d41d8c    /*Le poligono*/
83008c    for(int i = 0; i < n; i++)
3daa4c      scanf("%lld %lld", &p[i].x, &p[i].y);
d41d8c
d41d8c    /*Encontra CH*/
680587    vector<point<ll> > ch = CH(p);
d41d8c
d41d8c    /*Imorime resultado*/
3b846e    printf("%d\n", (int)ch.size());
```

```
c857e7    for(int i = 0; i < (int)ch.size(); i++)
fc3047      printf("%d% %d\n", ch[i].x, ch[i].y);
d41d8c
cbb184  }
b6794f
```

3.3 Min Enclosing Circle (randomized)

```
ad578e  #include "../2d/2d.cpp"
d41d8c
d41d8c  /*
744027    Minimum Enclosing Circle:
2d38cc      Given a list of points, returns a circle of minimum radius such
   that all given
2602de      points are within the circle.
0c4a3b      Runs in O(n) expected time (in practice 200 ms for 10^5 points).
d41d8c
ca2095    Constraints:
99b71a      Non-empty list of points.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
e89126  #define point point<double>
0f3aa0  #define circle circle<double>
d41d8c
41ee07  circle min_enclosing_circle(vector<point> p)
f95b70  {
b4da45    shuffle(p.begin(), p.end(), mt19937(time(0)));
2e09de    point o = p[0];
76160f    double r = 0, eps = 1 + 1e-8;
fe16ed    for (int i = 0; i < sz(p); i++)
197ee7      if ((o - p[i]).dist() > r * eps)
f95b70      {
ba37a5        o = p[i], r = 0;
c791cd        for (int j = 0; j < i; j++)
f5972f          if ((o - p[j]).dist() > r * eps)
f95b70          {
d2b545            o = (p[i] + p[j]) / 2;
0657ce            r = (o - p[i]).dist();
674051            for (int k = 0; k < j; k++)
355d4d              if ((o - p[k]).dist() > r * eps)
f95b70              {
7fb807                o = circumcircle(p[i], p[j], p[k]).center;
0657ce                r = (o - p[i]).dist();
cbb184              }
cbb184          }
cbb184      }
d41d8c
645c1d    return {o, r};
```

```
cbb184  }
d41d8c
5d3836
```

3.4 Min Enclosing Circle (ternary search)

```
ad578e  #include "../2d/2d.cpp"
2729c3  #include "../../misc/ternary_search/ternary_search_continuous.cpp"
d41d8c
d41d8c  /*
744027    Minimum Enclosing Circle:
2d38cc      Given a list of points, returns a circle of minimum radius such
   that all given
2602de      points are within the circle.
a1c921      Runs in O(n * log^2((top - bot) / eps)) (in practice 2.5s at best
    for 10^5 points).
d41d8c
ca2095    Constraints:
99b71a      Non-empty list of points.
d41d8c
b95cae    Usage:
ca9f24      The coordinates of the circle's center must be in the range [bot,
    top].
a09e29      eps specifies the precision of the result, but set it to a higher
    value
53006e      than necessary since the error in x affects the y value.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
e89126  #define point point<double>
0f3aa0  #define circle circle<double>
d41d8c
e1710f  circle min_enclosing_circle(const vector<point> &p, double bot = -1e9
   , double top = 1e9, double eps = 1e-9)
f95b70  {
1841a9    circle retv;
d41d8c
0a37af    auto f1 = [&](double x) {
d9991d      auto f2 = [&](double y)
f95b70      {
996834        double r = 0;
fe16ed        for (int i = 0; i < sz(p); i++)
62adf4          r = max(r, (p[i].x - x)*(p[i].x - x) + (p[i].y - y)*(p[i].y -
    y));
4c1f3c        return r;
2145c1      };
410f57      retv.center.y = ternary_search(f2, bot, top, eps);
50ac50      return f2(retv.center.y);
2145c1    };
```

```
d41d8c
596ad7    retv.center.x = ternary_search(f1, bot, top, eps);
3b2a60    retv.r = sqrt(f1(retv.center.x));
d41d8c
6272cf    return retv;
cbb184  }
2acede
```

4 Graph

4.1 Biconnected Components

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
399f8b    Finding bridges, articulation points and biconnected components in
   O(V + E):
5d1e77      A bridge is an edge whose removal splits the graph in two
   connected components.
8dd98b      An articulation point is a vertex whose removal splits the graph
   in two connected components.
d41d8c
8254c5      A biconnected component (or 2VCC) is a maximal subgraph where the
    removal of any vertex doesn't
44c916          make the subgraph disconnected. In other words, it is a
   maximal 2-vertex-connected (2VC) subgraph.
d41d8c
deb2a1          A 2-connected graph is a 2VC one, except that a---b is
   considered 2VC but not 2-connected.
d41d8c
3a585b      Useful theorems:
d41d8c
45e89c          A 2-edge connected (2EC) graph is a graph without bridges.
   Any 2-connected graph is also 2EC.
d41d8c
20dbb2          Let G be a graph on at least 2 vertices. The following
   propositions are equivalent:
81e567              * (i) G is 2-connected;
959a93              * (ii) any two vertices are in a cycle; (a cycle can't
   repeat vertices)
3f2e8a              * (iii) any two edges are in a cycle and Îť(G) âĽě 2;
57a10e              * (iv) for any three vertices x,y et z, there is a (x,z)-
   path containing y.
561b74          Let G be a graph on at least 3 vertices. The following
   propositions are equivalent:
346e5e              * (i) G is 2-edge-connected;
4883af              * (ii) any edge is in a cycle;
368e3a              * (iii) any two edges are in a tour and Îť âĽě 1;
d6bb5f              * (iv) any two vertices are in a tour (a tour can repeat
   vertices)
d41d8c
8c2af3      If G is 2-connected and not bipartite, all vertices belong to
   some odd cycle. And any two vertices are in a odd cycle (not really proven
   ).
d41d8c
bcc5c9          If G is 2-edge-connected (proof by AC):
33bf43              For any two vertices x, y and one edge e, there is a (x,
   y)-walk containing e without repeating edges.
```

```
d41d8c
ab50d8          A graph admits a strongly connected orientation if and only
   if it is 2EC.
d3ea79          A strong orientation of a given bridgeless undirected graph
   may be found in linear time by performing
0ec987          a depth first search of the graph, orienting all edges in the
    depth first search tree away from the
d494fc          tree root, and orienting all the remaining edges (which must
   necessarily connect an ancestor and a
030955          descendant in the depth first search tree) from the
   descendant to the ancestor.
d41d8c
ca2095    Constraints:
b9aa54      ***undirected*** graph.
80b2d0      Vertices are labeled from 0 to n (inclusive).
568d46      Graph is connected (but for unconnected just replace single dfs
   call with a loop).
d41d8c
b95cae    Usage:
4436dc      Create the struct setting the starting vertex (a), the maximum
   vertex label (n),
e1993f        the graph adjacency list (graph) and a callback f to apply on
   the biconnected components.
f8f25e      Afterwards, art[i] == true if i is an articulation point.
e9a79e      If the pair {a, i} is on the bridges list, then the edge {a,
   graph[a][i]} is a bridge.
ccfd29          The callback must receive a vector of edges {a, b} that are
   in the same biconnected component.
a32c3f      Remember that for a single vertex, the biconnected callback will
   not be called.
d41d8c
e152b4      Sample Usage:
0ec6ee          auto rdm = apb(1, n, graph, [&](vector<pii> v){
f4ecd5      set<int> s;
9ad08e      for (int i = 0; i < sz(v); i++)
f95b70      {
f19ef4        s.insert(v[i].first);
0858fa        s.insert(v[i].second);
cbb184      }
d41d8c
0fe299      ans = max(ans, sz(s));
c0c97e        });
c4c9bd  */
d41d8c
f117a6  struct apb
f95b70  {
9cf2b9  vector<int> *graph;
9cf143  vector<bool> art;
c9001f  vector<int> num /* dfs order of vertices starting at 1 */, low;
c83796  vector<pii> bridges;
```

```
91936b  vector<pii> st;
53e65f  int id;
d41d8c
044d82  template<class F>
09caad  apb(int a, int n, vector<int> graph[], const F &f) : graph(graph),
   art(n + 1, false), num(n + 1), low(n + 1)
f95b70  {
0f6720    id = 1;
ccac4e    dfs(a, a, f);
cbb184  }
d41d8c
044d82  template<class F>
dc584b  void dfs(int a, int p, const F &f)
f95b70  {
7be506    low[a] = num[a] = id++;
34863b    int comp = 0;
d41d8c
1429ef    for (int i = 0; i < sz(graph[a]); i++)
f95b70    {
b7a810      if (num[graph[a][i]] == 0)
f95b70      {
d40410              int si = sz(st);
f309f5        comp++;
8ece2e              st.push_back({a, graph[a][i]}); // Tree edge.
d41d8c
fc5941        dfs(graph[a][i], a, f);
085d64        low[a] = min(low[a], low[graph[a][i]]);
d41d8c
bb63a0        if (low[graph[a][i]] >= num[a])
f95b70              {
558f81                  if (a != 1)
016392              art[a] = true;
d41d8c
b91456                  f(vector<pii>(st.begin() + si, st.end()));
901921                  st.resize(si);
cbb184              }
d41d8c
0e9ddb        if (low[graph[a][i]] > num[a])
b3cacb          bridges.push_back({a, i});
cbb184      }
624580      else if (graph[a][i] != p && num[graph[a][i]] < num[a]) // Back
   edge.
f95b70          {
066898              low[a] = min(low[a], num[graph[a][i]]);
8ece2e              st.push_back({a, graph[a][i]});
cbb184          }
cbb184    }
d41d8c
85e3a2    if (a == p && comp > 1)
016392      art[a] = true;
```

```
cbb184  }
2145c1  };
d41d8c
5cb0b8
```

4.2 Bipartite Matching (Hopcroft Karp)

```
2b74fa  #include <bits/stdc++.h>
ca417d  using namespace std;
d41d8c
d41d8c  /*
ec23c9    Hopcroft-Karp:
eaeddf      Bipartite Matching O(sqrt(V)E)
d41d8c
ca2095    Constraints:
998cc9      Vertices are labeled from 1 to l + r (inclusive).
682ff0      DO NOT use vertex 0.
968b86      Vertices 1 to l belong to left partition.
a6a4c4      Vertices l + 1 to l + r belong to right partition.
d41d8c
b95cae    Usage:
d86132      Set MAXV if necessary.
70636b      Call init passing l and r.
0f3b71      Add edges to the graph from left side to right side.
5263f1      Call hopcroft to get the matching size.
a0da8e      Then, each vertex v has its pair indicated in p[v] (or 0 for not
   paired).
c4c9bd  */
d41d8c
dde07b  namespace hopcroft
f95b70  {
998014  const int inf = 0x3f3f3f3f;
ed5ed2  const int MAXV = 112345;
d41d8c
3098d4  vector<vector<int>> graph;
0a3d29  int d[MAXV], q[MAXV], p[MAXV], l, r;
d41d8c
4025e1  void init(int _l, int _r)
f95b70  {
0ebd66    l = _l, r = _r;
2213c3    graph = vector<vector<int>>(l + r + 1);
cbb184  }
d41d8c
6a1cf9  bool bfs()
f95b70  {
18753f    int qb = 0, qe = 0;
4f2bde    memset(d, 0x3f, sizeof(int) * (l + 1));
a89ba9    for (int i = 1; i <= l; i++)
8b3877      if (p[i] == 0)
248d2f        d[i] = 0, q[qe++] = i;
```

```
d41d8c
2caa87    while (qb < qe)
f95b70    {
e8e8a0      int a = q[qb++];
0087d7      if (a == 0)
8a6c14        return true;
c4fff3      for (int i = 0; i < graph[a].size(); i++)
68367c        if (d[p[graph[a][i]]] == inf)
a8cd28          d[q[qe++] = p[graph[a][i]]] = d[a] + 1;
cbb184    }
d41d8c
d1fe4d    return false;
cbb184  }
d41d8c
0752c9  bool dfs(int a)
f95b70  {
0087d7    if (a == 0)
8a6c14      return true;
c4fff3    for (int i = 0; i < graph[a].size(); i++)
7d85df      if (d[a] + 1 == d[p[graph[a][i]]])
a2f815        if (dfs(p[graph[a][i]]))
f95b70        {
460f0a          p[a] = graph[a][i];
51e040          p[graph[a][i]] = a;
8a6c14          return true;
cbb184        }
d41d8c
343737    d[a] = inf;
d1fe4d    return false;
cbb184  }
d41d8c
68fd9d  int hopcroft()
f95b70  {
9e3790    memset(p, 0, sizeof(int) * (l + r + 1));
fc833c    int matching = 0;
d594a7    while (bfs())
f95b70    {
a89ba9      for (int i = 1; i <= l; i++)
8b3877        if (p[i] == 0)
57e7a2          if (dfs(i))
730cbb            matching++;
cbb184    }
d41d8c
2afcbe    return matching;
cbb184  }
cbb184  } // namespace hopcroft
976bec
```

4.3 Bridges/Articulation Points

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
62784d    Finding bridges and articulation points in O(V + E):
5d1e77      A bridge is an edge whose removal splits the graph in two
   connected components.
8dd98b      An articulation point is a vertex whose removal splits the graph
   in two connected components.
8b8ace      This can also be adapted to generate the biconnected components
   of a graph, since the
14a784      articulation points split components.
d41d8c
d41d8c
ca2095    Constraints:
b9aa54      ***undirected*** graph.
80b2d0      Vertices are labeled from 0 to n (inclusive).
1e6120      Graph is connected (otherwise it doesn't make sense).
d41d8c
b95cae    Usage:
668bef      Create the struct setting the starting vertex (a), the maximum
   vertex label (n)
8acebe      and the graph adjacency list (graph).
6ffc91      Aftewards, art[i] == true if i is an articulation point.
e9a79e      If the pair {a, i} is on the bridges list, then the edge {a,
   graph[a][i]} is a bridge.
c4c9bd  */
d41d8c
f117a6  struct apb
f95b70  {
9cf2b9  vector<int> *graph;
9cf143  vector<bool> art;
c9001f  vector<int> num /* dfs order of vertices starting at 1 */, low;
c83796  vector<pii> bridges;
53e65f  int id;
d41d8c
4dc736  apb(int a, int n, vector<int> graph[]) : graph(graph), art(n + 1,
   false), num(n + 1), low(n + 1)
f95b70  {
0f6720    id = 1;
bb407e    dfs(a, a);
cbb184  }
d41d8c
69c421  void dfs(int a, int p)
f95b70  {
7be506    low[a] = num[a] = id++;
34863b    int comp = 0;
d41d8c
c4fff3    for (int i = 0; i < graph[a].size(); i++)
f95b70    {
b7a810      if (num[graph[a][i]] == 0)
```

```
f95b70      {
f309f5        comp++;
783129        dfs(graph[a][i], a);
085d64        low[a] = min(low[a], low[graph[a][i]]);
d41d8c
b28b5f        if (a != 1 && low[graph[a][i]] >= num[a])
016392          art[a] = true;
d41d8c
0e9ddb        if (low[graph[a][i]] > num[a])
b3cacb          bridges.push_back({a, i});
cbb184      }
2cae7a      else if (graph[a][i] != p && num[graph[a][i]] < low[a])
ed0d8a        low[a] = num[graph[a][i]];
cbb184    }
d41d8c
85e3a2    if (a == p && comp > 1)
016392      art[a] = true;
cbb184  }
2145c1  };
d41d8c
780b6d
```

4.4 Max Flow (Dinic)

```
2b74fa  #include <bits/stdc++.h>
ca417d  using namespace std;
d41d8c
d41d8c  /*
908d2f    Dinic:
67cbe4      Max-flow O(V^2E)
eaeddf      Bipartite Matching O(sqrt(V)E)
d41d8c
ca2095    Constraints:
80b2d0      Vertices are labeled from 0 to n (inclusive).
8f4ce8      Edge capacities must fit int (flow returned is long long).
d41d8c
b95cae    Usage:
d86132      Set MAXV if necessary.
148d9c      Call init passing n, the source and the sink.
2d6398      Add edges to the graph by calling put_edge(_undirected).
bb3825      Call max_flow to get the total flow. Then, individual edge flows
   can be retrieved in the graph.
22c3c2      Note that flow will be negative in return edges.
c4c9bd  */
d41d8c
82657b  namespace dinic
f95b70  {
729806  struct edge
f95b70  {
bf6256    int dest, cap, re, flow;
```

```
2145c1  };
d41d8c
998014  const int inf = 0x3f3f3f3f;
8550b5  const int MAXV = 312345;
d41d8c
8a367b  int n, s, t, d[MAXV], q[MAXV], next[MAXV];
d8f9f2  vector<vector<edge>> graph;
d41d8c
bc6f23  void init(int _n, int _s, int _t)
f95b70  {
c992e9    n = _n, s = _s, t = _t;
b72d19    graph = vector<vector<edge>>(n + 1);
cbb184  }
d41d8c
7c85eb  void put_edge(int u, int v, int cap)
f95b70  {
506964    graph[u].push_back({v, cap, (int)graph[v].size(), 0});
68ec95    graph[v].push_back({u, 0, (int)graph[u].size() - 1, 0});
cbb184  }
d41d8c
d6a592  void put_edge_undirected(int u, int v, int cap)
f95b70  {
506964    graph[u].push_back({v, cap, (int)graph[v].size(), 0});
fce495    graph[v].push_back({u, cap, (int)graph[u].size() - 1, 0});
cbb184  }
d41d8c
6a1cf9  bool bfs()
f95b70  {
18753f    int qb = 0, qe = 0;
3c6658    q[qe++] = s;
98fde3    memset(d, 0x3f, sizeof(int) * (n + 1));
d66185    d[s] = 0;
2caa87    while (qb < qe)
f95b70    {
e8e8a0      int a = q[qb++];
c9a55a      if (a == t)
8a6c14        return true;
3352c6      for (int i = 0; i < (int)graph[a].size(); i++)
f95b70      {
10e42b        edge &e = graph[a][i];
d948dd        if (e.cap - e.flow > 0 && d[e.dest] == inf)
f4063b          d[q[qe++] = e.dest] = d[a] + 1;
cbb184      }
cbb184    }
d41d8c
d1fe4d    return false;
cbb184  }
d41d8c
1a19d4  int dfs(int a, int flow)
f95b70  {
```

```
c9a55a    if (a == t)
99d2e8      return flow;
10647a    for (int &i = next[a]; i < (int)graph[a].size(); i++)
f95b70    {
10e42b      edge &e = graph[a][i];
c6fb85      if (d[a] + 1 == d[e.dest] && e.cap - e.flow > 0)
f95b70      {
5f308a        int x = dfs(e.dest, min(flow, e.cap - e.flow));
5f75db        if (x == 0)
5e2bd7          continue;
7f9751        e.flow += x;
4c55a5        graph[e.dest][e.re].flow -= x;
ea5659        return x;
cbb184      }
cbb184    }
d41d8c
343737    d[a] = inf;
bb30ba    return 0;
cbb184  }
d41d8c
afa2f7  long long max_flow()
f95b70  {
f013d3    long long total_flow = 0;
d594a7    while (bfs())
f95b70    {
ba90c2      memset(next, 0, sizeof(int) * (n + 1));
60616b      while (int path_flow = dfs(s, inf))
a0d8d9        total_flow += path_flow;
cbb184    }
d41d8c
793f63    return total_flow;
cbb184  }
cbb184  } // namespace dinic
73c17c
```

4.5 Max Flow (Dinic w/ Scaling)

```
2b74fa  #include <bits/stdc++.h>
ca417d  using namespace std;
d41d8c
d41d8c  /*
3678e9    Dinic with Scaling:
476fd1      Max-flow O(VE * log(MAX_CAP)), but usually slower than regular
   Dinic.
d41d8c
ca2095    Constraints:
80b2d0      Vertices are labeled from 0 to n (inclusive).
8f4ce8      Edge capacities must fit int (flow returned is long long).
d41d8c
b95cae    Usage:
```

```
d86132      Set MAXV if necessary.
148d9c      Call init passing n, the source and the sink.
2d6398      Add edges to the graph by calling put_edge(_undirected).
bb3825      Call max_flow to get the total flow. Then, individual edge flows
   can be retrieved in the graph.
22c3c2      Note that flow will be negative in return edges.
c4c9bd  */
d41d8c
82657b  namespace dinic
f95b70  {
729806  struct edge
f95b70  {
bf6256    int dest, cap, re, flow;
2145c1  };
d41d8c
998014  const int inf = 0x3f3f3f3f;
8550b5  const int MAXV = 312345;
d41d8c
19c361  int n, s, t, lim, d[MAXV], q[MAXV], next[MAXV];
d8f9f2  vector<vector<edge>> graph;
d41d8c
bc6f23  void init(int _n, int _s, int _t)
f95b70  {
c992e9    n = _n, s = _s, t = _t;
b72d19    graph = vector<vector<edge>>(n + 1);
cbb184  }
d41d8c
7c85eb  void put_edge(int u, int v, int cap)
f95b70  {
506964    graph[u].push_back({v, cap, (int)graph[v].size(), 0});
68ec95    graph[v].push_back({u, 0, (int)graph[u].size() - 1, 0});
cbb184  }
d41d8c
d6a592  void put_edge_undirected(int u, int v, int cap)
f95b70  {
506964    graph[u].push_back({v, cap, (int)graph[v].size(), 0});
fce495    graph[v].push_back({u, cap, (int)graph[u].size() - 1, 0});
cbb184  }
d41d8c
6a1cf9  bool bfs()
f95b70  {
18753f    int qb = 0, qe = 0;
3c6658    q[qe++] = s;
98fde3    memset(d, 0x3f, sizeof(int) * (n + 1));
d66185    d[s] = 0;
2caa87    while (qb < qe)
f95b70    {
e8e8a0      int a = q[qb++];
c9a55a      if (a == t)
8a6c14        return true;
```

```
3352c6      for (int i = 0; i < (int)graph[a].size(); i++)
f95b70      {
10e42b        edge &e = graph[a][i];
21aca9        if (e.cap - e.flow >= lim && d[e.dest] == inf)
f4063b          d[q[qe++] = e.dest] = d[a] + 1;
cbb184      }
cbb184    }
d41d8c
d1fe4d    return false;
cbb184  }
d41d8c
1a19d4  int dfs(int a, int flow)
f95b70  {
c9a55a    if (a == t)
99d2e8      return flow;
10647a    for (int &i = next[a]; i < (int)graph[a].size(); i++)
f95b70    {
10e42b      edge &e = graph[a][i];
cbf046      if (d[a] + 1 == d[e.dest] && e.cap - e.flow >= lim /* >= 1 ? */)
f95b70      {
5f308a        int x = dfs(e.dest, min(flow, e.cap - e.flow));
5f75db        if (x == 0)
5e2bd7          continue;
7f9751        e.flow += x;
4c55a5        graph[e.dest][e.re].flow -= x;
ea5659        return x;
cbb184      }
cbb184    }
d41d8c
343737    d[a] = inf;
bb30ba    return 0;
cbb184  }
d41d8c
afa2f7  long long max_flow()
f95b70  {
f013d3    long long total_flow = 0;
aab413      for (lim = (1 << 30); lim >= 1; lim >>= 1)
d594a7      while (bfs())
f95b70      {
ba90c2        memset(next, 0, sizeof(int) * (n + 1));
60616b        while (int path_flow = dfs(s, inf))
a0d8d9          total_flow += path_flow;
cbb184      }
d41d8c
793f63    return total_flow;
cbb184  }
cbb184  } // namespace dinic
c74595
```

4.6 Min Cost Max Flow

```
2b74fa  #include <bits/stdc++.h>
ca417d  using namespace std;
d41d8c
d41d8c  /*
dfc480    Min-Cost Max-Flow: O(V^2E^2)
078f0d      Finds the maximum flow of minimum cost.
d41d8c
ca2095    Constraints:
80b2d0      Vertices are labeled from 0 to n (inclusive).
247575      Edge cost and capacities must fit int (flow and cost returned are
    long long).
75ef18      Edge Cost must be non-negative.
d41d8c
b95cae    Usage:
d86132      Set MAXV if necessary.
148d9c      Call init passing n, the source and the sink.
909583      Add edges to the graph by calling put_edge.
553d3e      Call mincost_maxflow to get the total flow and its cost (in this
   order).
0e374d      Individual edge flows can be retrieved in the graph. Note that
   flow will be negative in return edges.
c4c9bd  */
d41d8c
ad1153  typedef long long ll;
d29b14  typedef pair<long long, long long> pll;
d41d8c
e3de19  namespace mcmf
f95b70  {
729806  struct edge
f95b70  {
60f183    int dest, cap, re, cost, flow;
2145c1  };
d41d8c
ed5ed2  const int MAXV = 112345;
6a4c6c  const ll infll = 0x3f3f3f3f3f3f3f3fLL;
998014  const int inf = 0x3f3f3f3f;
d41d8c
128c92  int n, s, t, p[MAXV], e_used[MAXV];
97e5bb  bool in_queue[MAXV];
c97378  ll d[MAXV];
d41d8c
d8f9f2  vector<vector<edge>> graph;
d41d8c
bc6f23  void init(int _n, int _s, int _t)
f95b70  {
c992e9    n = _n, s = _s, t = _t;
b72d19    graph = vector<vector<edge>>(n + 1);
cbb184  }
d41d8c
a4abfa  void put_edge(int u, int v, int cap, int cost)
```

```
f95b70  {
bd3e8b    graph[u].push_back({v, cap, (int)graph[v].size(), cost, 0});
2b8b8d    graph[v].push_back({u, 0, (int)graph[u].size() - 1, -cost, 0});
cbb184  }
d41d8c
b34984  bool spfa()
f95b70  {
664c61    memset(in_queue, 0, sizeof(bool) * (n + 1));
9eef50    memset(d, 0x3f, sizeof(ll) * (n + 1));
26a528    queue<int> q;
d66185    d[s] = 0;
e2828b    p[s] = s;
08bec3    q.push(s);
ee6bdd    while (!q.empty())
f95b70    {
0930a5      int a = q.front();
833270      q.pop();
e7249b      in_queue[a] = false;
d41d8c
c4fff3      for (int i = 0; i < graph[a].size(); i++)
f95b70      {
10e42b        edge &e = graph[a][i];
6fa321        if (e.cap - e.flow > 0 && d[e.dest] > d[a] + e.cost)
f95b70        {
3bf598          d[e.dest] = d[a] + e.cost;
6d6530          p[e.dest] = a;
183d83          e_used[e.dest] = i;
27788c          if (!in_queue[e.dest])
b34293            q.push(e.dest);
04f0f7          in_queue[e.dest] = true;
cbb184        }
cbb184      }
cbb184    }
d41d8c
d1cd45    return d[t] < infll;
cbb184  }
d41d8c
99658d  pll mincost_maxflow()
f95b70  {
f04b2a    pll retv = pll(0, 0);
d9383f    while (spfa())
f95b70    {
e98031      int x = inf;
c9b315      for (int i = t; p[i] != i; i = p[i])
d4a316        x = min(x, graph[p[i]][e_used[i]].cap - graph[p[i]][e_used[i]].
   flow);
c9b315      for (int i = t; p[i] != i; i = p[i])
dc731d        graph[p[i]][e_used[i]].flow += x, graph[i][graph[p[i]][e_used[i
   ]].re].flow -= x;
d41d8c
```

```
75907a      retv.first += x;
1be465      retv.second += x * d[t];
cbb184    }
d41d8c
6272cf    return retv;
cbb184  }
cbb184  } // namespace mcmf
5bd8df
```

4.7 Heavy-Light Decomposition

```
2b74fa  #include<bits/stdc++.h>
d41d8c
ca417d  using namespace std;
d41d8c
eed838  #define ll long long
efe13e  #define pb push_back
d41d8c
3a6c63  typedef vector<ll> vll;
990dd5  typedef vector<int> vi;
d41d8c
e06cc0  #define MAXN 100010
d41d8c
d41d8c  //Vetor que guarda a arvore
698e25  vector<vi> adj;
d41d8c
9e6e6d  int subsize[MAXN], parent[MAXN];
d41d8c  //Inciar chainHead com -1; e chainSize e chainNo com 0.
080553  int chainNo = 0, chainHead[MAXN], chainPos[MAXN], chainInd[MAXN],
   chainSize[MAXN];
42a605  void hld(int cur){
cb42fb    if(chainHead[chainNo] == -1)
6591fe      chainHead[chainNo] = cur;
d41d8c
3a4605    chainInd[cur] = chainNo;
220e91    chainPos[cur] = chainSize[chainNo];
6f00fb    chainSize[chainNo]++;
d41d8c
89108d    int ind = -1, mai = -1;
9d9afd    for(int i = 0; i < (int)adj[cur].size(); i++){
9ff2fa      if(adj[cur][i] != parent[cur] && subsize[adj[cur][i]] > mai){
31fcc6        mai = subsize[adj[cur][i]];
b9b7e9        ind  = i;
cbb184      }
cbb184    }
d41d8c
27d206    if(ind >= 0)
f23581      hld(adj[cur][ind]);
d41d8c
e506c6    for(int i = 0; i < (int)adj[cur].size(); i++)
```

```
6f7286      if(adj[cur][i] != parent[cur] && i != ind){
959ef6        chainNo++;
270563        hld(adj[cur][i]);
cbb184      }
cbb184    }
d41d8c
d41d8c    //usar LCA para garantir que v eh pai de u!!
f179f7    ll query_up(int u, int v){
c20c7b      int uchain = chainInd[u], vchain = chainInd[v];
bdd5ea      ll ans = 0LL;
d41d8c
31e3cd      while(1){
f523c5        if(uchain == vchain){
d41d8c          //Query deve ir de chainPos[i] ate chainPos[v]
7d2150          ll cur = /*sum(chainPos[u], uchain) - (chainPos[u] == 0? 0LL :
     sum(chainPos[v] - 1, vchain))*/;
d133d8          ans += cur;
c2bef1          break;
cbb184        }
d41d8c
d41d8c        //Query deve ir de chainPos[i] ate o fim da estrutura
d41d8c        //ll cur = sum(chainPos[u], uchain);
d133d8        ans += cur;
a258cd        u = chainHead[uchain];
8039e1        u = parent[u];
cabc24        uchain = chainInd[u];
cbb184      }
ba75d2      return ans;
cbb184    }
d41d8c
b7aa64    int dfs0(int pos, int prev = -1){
c92501      int res  = 1;
97817b      for(int i = 0; i < (int)adj[pos].size(); i++){
ec49a3        int nx  = adj[pos][i];
773904        if(nx != prev){
3f20e3          res += dfs0(nx, pos);
522845          parent[nx] = pos;
cbb184        }
cbb184      }
a1881c      return subsize[pos] = res;
cbb184    }
d41d8c
0b8977    int main()
f95b70    {
d41d8c      //Salvar arvore em adj
d41d8c
d41d8c      //Inicializa estrutura de dados
b75143      memset(chainHead, -1, sizeof(chainHead));
d41d8c
d41d8c      //Ou 0, se for o no raiz
```

```
bf6cde    dfs0(1);
bac429    hld(1);
d41d8c
d41d8c    //Inicializar estruturas usadas
cbb184  }
90a698
```

4.8 Strongly Connected Components

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
eaba86    Strongly connected components in O(V + E):
970b0b      Finds all strongly connected components of a graph.
3f48a9          A strongly connected component is a maximal set of vertices such that
de0185          every vertex can reach every other vertex in the component.
0a2f52          The graph where the SCCs are considered vertices is a DAG.
d41d8c
ca2095    Constraints:
000269      Vertices are labeled from 1 to n (inclusive).
d41d8c
b95cae    Usage:
ee0d06      Create the struct setting the maximum vertex label (n) and the graph adjacency list (graph).
862124      Aftewards, ncomp has the number of SCCs in the graph and scc[i] indicates the SCC i
67d138          belongs to (1 <= scc[i] <= ncomp).
d41d8c
38224f          sorted is a topological ordering of the graph, byproduct of the algorithm.
484a00          if edge a -> b exists, a appears before b in the sorted list.
c4c9bd  */
d41d8c
d41d8c
73e60a  struct scc_decomp
f95b70  {
9cf2b9      vector<int> *graph;
00b6a0      vector<vector<int>> tgraph;
1b013f      vector<int> scc;
1ee615      vector<bool> been;
8d35d1      int ncomp;
2035f3      list<int> sorted;
d41d8c
4875fb      scc_decomp(int n, vector<int> graph[]) : graph(graph), tgraph(n + 1), scc(n + 1, 0), been(n + 1, false), ncomp(0)
f95b70      {
5359f3          for (int i = 1; i <= n; i++)
6376e8              for (int j = 0; j < graph[i].size(); j++)
14234d                  tgraph[graph[i][j]].push_back(i);
```

```
d41d8c
5359f3          for (int i = 1; i <= n; i++)
018df6        if(!been[i])
1e5da3          dfs(i);
d41d8c
16ef86      for(int a : sorted)
f49735        if(scc[a] == 0)
f95b70        {
a8f1f2                  ncomp++;
4dd966          dfst(a);
cbb184        }
cbb184      }
d41d8c
0cbab3      void dfs(int a)
f95b70      {
1689c6          been[a] = true;
c4fff3          for(int i = 0; i < graph[a].size(); i++)
b0c443              if(!been[graph[a][i]])
fded7f                  dfs(graph[a][i]);
ddbf66          sorted.push_front(a);
cbb184      }
d41d8c
9b760a      void dfst(int a)
f95b70      {
1689c6          been[a] = true;
d28fcc          scc[a] = ncomp;
9c28b7          for(int i = 0; i < tgraph[a].size(); i++)
c480c5              if(scc[tgraph[a][i]] == 0)
caa482                  dfst(tgraph[a][i]);
cbb184      }
2145c1  };
20fe5c
```

5 Misc

5.1 DP Optimization - Binary Search

```cpp
// https://codeforces.com/contest/321/problem/E

#include "../../contest/header.hpp"

/*
  Binary Search Optimization for DP:
    Optimizes dp of the form (or similar) dp[i][j] = min_{k < i}(dp[k][j-1] + c(k + 1, i)).
    The classical case is a partitioning dp, where k determines the break point for the next partition.
    In this case, i is the number of elements to partition and j is the number of partitions allowed.

    Let opt[i][j] be the values of k which minimize the function. (in case of tie, choose the smallest)
    To apply this optimization, you need opt[i][j] <= opt[i+1][j].
    That means the when you add an extra element (i + 1), your partitioning choice will not be to include more elements
    than before (e.g. will no go from choosing [k, i] to [k-1, i+1]).
    This is usually intuitive by the problem details.

    Time goes from O(n^2m) to O(nm log n).

    To apply try to write the dp in the format above and verify if the property holds.

  Author: Arthur Pratti Dadalto
*/

#define MAXN 4123
#define MAXM 812

int n, m;
int u[MAXN][MAXN];
int tab[MAXN][MAXM];

inline int c(int i, int j)
{
  return (u[j][j] - u[j][i - 1] - u[i - 1][j] + u[i - 1][i - 1]) / 2;
}

// This is responsible for computing tab[l...r][j], knowing that opt[l...r][j] is in range [low_opt...high_opt]
void compute(int j, int l, int r, int low_opt, int high_opt)
{
  int mid = (l + r) / 2, opt = -1; // mid is equivalent to i in the
```

```
   original dp.
d41d8c
7222d3    tab[mid][j] = inf;
0e2f2c    for (int k = low_opt; k <= high_opt && k < mid; k++)
6f6e42      if (tab[k][j - 1] + c(k + 1, mid) < tab[mid][j])
f95b70      {
451068        tab[mid][j] = tab[k][j - 1] + c(k + 1, mid);
613f3c        opt = k;
cbb184      }
d41d8c
d41d8c    // New bounds on opt for other pending computation.
42c8a1    if (l <= mid - 1)
c7dd31      compute(j, l, mid - 1, low_opt, opt);
8b4e40    if (mid + 1 <= r)
8aa379      compute(j, mid + 1, r, opt, high_opt);
cbb184  }
d41d8c
13a4b1  int main(void)
f95b70  {
d69917    scanf("%d %d", &n, &m);
5359f3    for (int i = 1; i <= n; i++)
947790      for (int j = 1; j <= n; j++)
f95b70      {
433ab9        getchar();
512e3d        u[i][j] = getchar() - '0';
cbb184      }
d41d8c
5359f3    for (int i = 1; i <= n; i++)
947790      for (int j = 1; j <= n; j++)
a10370        u[i][j] += u[i - 1][j] + u[i][j - 1] - u[i - 1][j - 1];
d41d8c
5359f3    for (int i = 1; i <= n; i++)
5c5410      tab[i][0] = inf;
d41d8c
d41d8c    // Original dp
d41d8c    // for (int i = 1; i <= n; i++)
d41d8c    //   for (int j = 1; j <= m; j++)
d41d8c    //   {
d41d8c    //     tab[i][j] = inf;
d41d8c    //     for (int k = 0; k < i; k++)
d41d8c    //       tab[i][j] = min(tab[i][j], tab[k][j-1] + c(k + 1,i);
d41d8c    //   }
d41d8c
2e2a5d    for (int j = 1; j <= m; j++)
fdaa69      compute(j, 1, n, 0, n - 1);
d41d8c
721eeb    cout << tab[n][m] << endl;
cbb184  }
f2bb43
```

5.2 DP Optimization - CHT

```
d41d8c  // https://codeforces.com/contest/319/problem/C
d41d8c
ad67d1  #include "../../data_structures/line_container/line_container.cpp"
d41d8c
d41d8c  /*
082e10    Convex Hull Trick for DP:
366d02      Transforms dp of the form (or similar) dp[i] = min_{j < i}(dp[j]
   + b[j] * a[i]).
ab5453      Time goes from O(n^2) to O(n log n), if using online line
   container, or O(n) if
56c021      lines are inserted in order of slope and queried in order of x.
d41d8c
3ffb56      To apply try to find a way to write the factor inside
   minimization as a linear function
ac2c47      of a value related to i. Everything else related to j will become
    constant.
c4c9bd  */
d41d8c
69abfb  #define MAXN 112345
d41d8c
a58cd5  int a[MAXN];
c4b25f  int b[MAXN];
d41d8c
f80900  ll tab[MAXN];
d41d8c
13a4b1  int main(void)
f95b70  {
1a88fd    int n;
f4c120    scanf("%d", &n);
83008c    for (int i = 0; i < n; i++)
9376f3      scanf("%d", &a[i]);
83008c    for (int i = 0; i < n; i++)
264aeb      scanf("%d", &b[i]);
d41d8c
a447b8    tab[0] = 0;
79ab5f    line_container l;
c01116    l.add(-b[0], -tab[0]);
d41d8c
aa4866    for (int i = 1; i < n; i++)
f95b70    {
23bd61      tab[i] = -l.query(a[i]);
8fd447      l.add(-b[i], -tab[i]);
cbb184    }
d41d8c
d41d8c    // Original DP O(n^2).
d41d8c    // for (int i = 1; i < n; i++)
d41d8c    // {
d41d8c    //  tab[i] = inf;
```

```
d41d8c    //  for (int j = 0; j < i; j++)
d41d8c    //    tab[i] = min(tab[i], tab[j] + a[i] * b[j]);
d41d8c    // }
d41d8c
cf6e15    cout << tab[n - 1] << endl;
cbb184  }
722d84
```

5.3 DP Optimization - Knuth

```
d41d8c  // https://www.spoj.com/problems/BRKSTRNG/
d41d8c
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
c97188    Knuth Optimization for DP:
a41b1d      Optimizes dp of the form (or similar) dp[i][j] = min_{i <= k <= j
   }(dp[i][k-1] + dp[k+1][j] + c(i, j)).
f833b2      The classical case is building a optimal binary tree, where k
   determines the root.
d41d8c
c8aa2c      Let opt[i][j] be the value of k which minimizes the function. (in
    case of tie, choose the smallest)
c472ed      To apply this optimization, you need opt[i][j - 1] <= opt[i][j]
   <= opt[i+1][j].
0eeb73      That means the when you remove an element form the left (i + 1),
   you won't choose a breaking point more to the left than before.
b38eb7      Also, when you remove an element from the right (j - 1), you won'
   t choose a breking point more to the right than before.
242554      This is usually intuitive by the problem details.
d41d8c
cbb42a      Time goes from O(n^3) to O(n^2).
d41d8c
895144      To apply try to write the dp in the format above and verify if
   the property holds.
f76c09      Be careful with edge cases for opt.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
dbf7e4  #define MAXN 1123
d41d8c
c4b25f  int b[MAXN];
1ee552  ll tab[MAXN][MAXN];
38ab0d  int opt[MAXN][MAXN];
ef864b  int l, n;
d41d8c
5a7750  int c(int i, int j)
f95b70  {
33e24b    return b[j + 1] - b[i - 1];
```

```
cbb184  }
d41d8c
13a4b1  int main(void)
f95b70  {
57a598    while (scanf("%d %d", &l, &n) != EOF)
f95b70    {
5359f3      for (int i = 1; i <= n; i++)
264aeb        scanf("%d", &b[i]);
665bd2      b[n + 1] = l;
00d08b      b[0] = 0;
d41d8c
da41df      for (int i = 1; i <= n + 1; i++)
d6bc61        tab[i][i - 1] = 0, opt[i][i - 1] = i;
d41d8c
586d50      for (int i = n; i > 0; i--)
5d4199        for (int j = i; j <= n; j++)
f95b70        {
639af9          tab[i][j] = infll;
823124          for (int k = max(i, opt[i][j - 1]); k <= j && k <= opt[i +
   1][j]; k++)
9e9168            if (tab[i][k - 1] + tab[k + 1][j] + c(i, j) < tab[i][j])
f95b70            {
680c31              tab[i][j] = tab[i][k - 1] + tab[k + 1][j] + c(i, j);
14da03              opt[i][j] = k;
cbb184            }
cbb184        }
d41d8c
ea7bd9      printf("%lld\n", tab[1][n]);
cbb184    }
cbb184  }
17b7c8
```

5.4 Ternary Search (continuous)

```
d41d8c  /*
0a2f9f    Ternary Search:
28ea2d      Finds x such that f(x) is minimum in range [bot, top] in O(lg((
   top - bot) / eps)).
6f01ba      Value is correct within the specified precision eps.
d41d8c
ca2095    Constraints:
7474cd      f(x) is strictly decreasing for some interval [bot, x1], constant
    in an interval [x1, x2]
60dab3      and strictly increasing in a interval [x2, top]. x1 <= x2 are
   arbitrary values where [x1, x2]
7523fb      is a plateau of optimal solutions.
d41d8c
b95cae    Usage:
5b60e3      Call the function passing a lambda expression or function f.
8bc9f4      If there are multiple possible solutions, assume that an
```

```
   arbitrary one in the plateau is returned.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
398727  template <typename F>
ca64a1  double ternary_search(const F &f, double bot = -1e9, double top = 1e9
   , double eps = 1e-9)
f95b70  {
14d91b    while (top - bot > eps)
f95b70    {
8e37d7      double x1 = (0.55*bot + 0.45*top); // (2*bot + top) / 3 is more
   stable, but slower.
3f8318      double x2 = (0.45*bot + 0.55*top);
9482ba      if (f(x1) > f(x2))
443914        bot = x1;
2954e9      else
16b1b8        top = x2;
cbb184    }
d41d8c
05eb81    return (bot + top) / 2;
cbb184  }
082811
```

6 Number Theory

6.1 Euclid

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
4b5b70    Extended Euclidean Algorithm:
71fa74      Returns the gcd of a and b.
2c19ff      Also finds numbers x and y for which a * x + b * y = gcd(a, b) (
   not unique).
a0c250      All pairs can be represented in the form (x + k * b / gcd, y - k
   * a / gcd) for k an arbitrary integer.
57ad55      If there are several such x and y, the function returns the pair
   for which |x| + |y| is minimal.
2d446b      If there are several x and y satisfying the minimal criteria, it
   outputs the pair for which X <= Y.
d41d8c
3997db    Source: modified from https://cp-algorithms.com/algebra/extended-
   euclid-algorithm.html
d41d8c
b95cae    Usage:
6475da      For non-extendend version, c++ has __gcd and __lcm.
d41d8c
ca2095    Constraints:
30a9e9      Produces correct results for negative integers as well.
c4c9bd  */
d41d8c
4fce64  template<class T>
94606e  T gcd(T a, T b, T &x, T &y)
f95b70  {
fcbb63    if (b == 0)
f95b70    {
483406      x = 1;
01dbf4      y = 0;
3f5343      return a;
cbb184    }
d41d8c
32895f    T x1, y1;
254183    T d = gcd(b, a % b, x1, y1);
711e33    x = y1;
a2a46d    y = x1 - y1 * (a / b);
be245b    return d;
cbb184  }
0c35ae
```

6.2 Pollard rho

```
d41d8c   /*
baee35    Description: Pollard-rho randomized factorization algorithm.
   Returns prime
```

```
9849b1    factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11})
   .
c21e7b    Time: O(n^1/4) gcd calls, less for numbers with small factors.
d41d8c
1d1558    Source: https://github.com/kth-competitive-programming/kactl/blob/
   master/content/number-theory/Factor.h
c4c9bd    */
f4cf5b  typedef unsigned long long ull;
088cf4  typedef long double ld;
d41d8c
ae330e  ull mod_mul(ull a, ull b, ull M) {
053258    ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
964402    return ret + M * (ret < 0) - M * (ret >= (ll)M);
cbb184  }
97f234  ull mod_pow(ull b, ull e, ull mod) {
c1a4a1    ull ans = 1;
4d1884    for (; e; b = mod_mul(b, b, mod), e /= 2)
0c1ecc      if (e & 1) ans = mod_mul(ans, b, mod);
ba75d2    return ans;
cbb184  }
d41d8c
da49ed  bool isPrime(ull n) {
32f8ec    if (n < 2 || n % 6 % 4 != 1) return n - 2 < 2;
43a246    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
c17dd6        s = __builtin_ctzll(n-1), d = n >> s;
d236d6    for(auto &a : A) {   // ^ count trailing zeroes
8a86e5      ull p = mod_pow(a, d, n), i = s;
274cbc      while (p != 1 && p != n - 1 && a % n && i--)
2cbb80        p = mod_mul(p, p, n);
e2871b      if (p != n-1 && i != s) return 0;
cbb184    }
6a5530    return 1;
cbb184  }
d41d8c
7eb30f  ull pollard(ull n) {
4de5da    auto f = [n](ull x) { return (mod_mul(x, x, n) + 1) % n; };
68eade    if (!(n & 1)) return 2;
7b6fa7    for (ull i = 2;; i++) {
e17462      ull x = i, y = f(x), p;
332fe8      while ((p = __gcd(n + y - x, n)) == 1)
b789c2        x = f(x), y = f(f(y));
c493bb      if (p != n) return p;
cbb184    }
cbb184  }
d41d8c
bc43a4  vector<ull> factorize(ull n) {
1b90e8    if (n == 1) return {};
6b5b32    if (isPrime(n)) return {n};
bc6125    ull x = pollard(n);
b3b29a    auto l = factorize(x), r = factorize(n / x);
```

```
7af87c    l.insert(l.end(), all(r));
792fd4    return l;
cbb184  }
93549d
```

6.3 Modular Inverse

```
771bdd  #include "../euclid/euclid.cpp"
d41d8c
d41d8c  /*
18a91e    Modular Inverse:
76e032      Returns an integer x such that (a * x) % m == 1.
4e4745      The modular inverse exists if and only if a and m are relatively
   prime.
ff1c03      Modular inverse is also equal to a^(phi(m) - 1) % m.
261f2c      In particular, if m is prime a^(-1) == a^(m-2), which might be
   faster to code.
d41d8c
3997db    Source: modified from https://cp-algorithms.com/algebra/module-
   inverse.html
c4c9bd  */
d41d8c
4fce64  template<class T>
b267c1  T mod_inverse(T a, T m)
f95b70  {
645c5d    T x, y;
6553c1    assert(gcd(a, m, x, y) == 1); // Or return something, if gcd is not
    1 the inverse doesn't exist.
08ffd4    return (x % m + m) % m;
cbb184  }
7efa11
```

6.4 Phi

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
bf26c1    Euler's totient function (PHI):
fc5093      Euler's totient function, also known as phi-function PHI(n),
   counts the number of integers
d7ef61      between 1 and n inclusive, which are coprime to n. Two numbers
   are coprime if their greatest
bf3431      common divisor equals 1 (1 is considered to be coprime to any
   number).
d41d8c
d41d8c
3997db    Source: modified from https://cp-algorithms.com/algebra/phi-
   function.html
e1fde7      and https://github.com/kth-competitive-programming/kactl/blob/
   master/content/number-theory/phiFunction.h
d41d8c
```

```
b95cae    Usage:
a1b248      Some useful properties:
9f5d84      - If p is a prime number, PHI(p)=p-1.
d4d311      - If a and b are relatively prime, PHI(ab)=PHI(a)*PHI(b).
65a02c      - In general, for not coprime a and b, PHI(ab)=PHI(a)*PHI(b)*d/
   PHI(d), with d=gcd(a,b) holds.
417c3d      - PHI(PHI(m)) <= m / 2
037fb5      - Euler's theorem: a^PHI(m) === 1 (mod m), for a and m coprime.
bffb1c      - For a and m coprime: a^n === a^(n % PHI(m)) (mod m)
986ce1      - For arbitrary x,m and n >= log_2(m): x^n === x^(PHI(m)+[n % PHI
   (m)]) (mod m)
8d568b      The one above allows computing modular exponentiation for really
   large exponents.
ec5d4e      - If d is a divisor of n, then there are phi(n/d) numbers i <= n
   for which gcd(i,n)=d
137411      - sum_{d|n} phi(d) = n
c228b3      - sum_{1 <= k <= n, gcd(k,n)=1} k = n * phi(n) / 2, for n > 1
c4c9bd  */
d41d8c
d41d8c  // Use this one for few values of phi.
b5f6f9  int phi(int n)
f95b70  {
efa47a    int result = n;
83f497    for (int i = 2; i * i <= n; i++)
f95b70    {
775f6d      if (n % i == 0)
f95b70      {
49edb8        while (n % i == 0)
1358bf          n /= i;
21cd49        result -= result / i;
cbb184      }
cbb184    }
f3d362    if (n > 1)
e48781      result -= result / n;
dc8384    return result;
cbb184  }
d41d8c
4fee4d  namespace totient
f95b70  {
2d637a  const int MAXV = 1000001; // Takes ~0.03 s for 10^6.
6e559b  int phi[MAXV];
d41d8c
b2a56e  void init()
f95b70  {
9484bb    for (int i = 0; i < MAXV; i++)
ed1f90      phi[i] = i & 1 ? i : i / 2;
9be7d6    for (int i = 3; i < MAXV; i += 2)
a2252f      if (phi[i] == i)
8a4437        for (int j = i; j < MAXV; j += i)
a9ba36          phi[j] -= phi[j] / i;
```

```
cbb184  }
cbb184  } // namespace totient
e79764
```

6.5 Sieve

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
d5cfbe    Sieve of Eratosthenes:
0a5343      Finds all primes in interval [2, MAXP] in O(MAXP) time.
7e51df      Also finds lp[i] for every i in [2, MAXP], such that lp[i] is the
    minimum prime factor of i.
6dbf8e      Particularly useful for factorization.
d41d8c
3997db    Source: modified from https://cp-algorithms.com/algebra/prime-sieve
   -linear.html
d41d8c
b95cae    Usage:
9290fb      Set MAXP and call init.
85265b      Sieve for 10^7 should run in about 0.2 s.
c4c9bd  */
d41d8c
2ca8b0  namespace sieve
f95b70  {
bace98  const int MAXP = 10000000; // Will find primes in interval [2, MAXP].
39b809  int lp[MAXP + 1]; // lp[i] is the minimum prime factor of i.
632f82  vector<int> p; // Ordered list of primes up to MAXP.
d41d8c
b2a56e  void init()
f95b70  {
008cd3    for (int i = 2; i <= MAXP; i++)
f95b70    {
d4a1cc      if (lp[i] == 0)
b6fab7        p.push_back(lp[i] = i);
d41d8c
9d854a      for (int j = 0; j < (int)p.size() && p[j] <= lp[i] && i * p[j] <=
    MAXP; j++)
fb7d48        lp[i * p[j]] = p[j];
cbb184    }
cbb184  }
cbb184  } // namespace sieve
e9076d
```

7 Numerical

7.1 Big Int

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  // This code is not meant to be written in icpc contests. This is
   just here to fill a void for now.
d41d8c  // Source: someone on CF
d41d8c
d41d8c  // NOTE:
d41d8c  // This code contains various bug fixes compared to the original
   version from
d41d8c  // indy256 (github.com/indy256/codelibrary/blob/master/cpp/
   numbertheory/bigint-full.cpp),
d41d8c  // including:
d41d8c  // - Fix overflow bug in mul_karatsuba.
d41d8c  // - Fix overflow bug in fft.
d41d8c  // - Fix bug in initialization from long long.
d41d8c  // - Optimized operators + - *.
d41d8c  //
d41d8c  // Tested:
d41d8c  // - https://www.e-olymp.com/en/problems/266: Comparison
d41d8c  // - https://www.e-olymp.com/en/problems/267: Subtraction
d41d8c  // - https://www.e-olymp.com/en/problems/271: Multiplication
d41d8c  // - https://www.e-olymp.com/en/problems/272: Multiplication
d41d8c  // - https://www.e-olymp.com/en/problems/313: Addition
d41d8c  // - https://www.e-olymp.com/en/problems/314: Addition/Subtraction
d41d8c  // - https://www.e-olymp.com/en/problems/317: Multiplication (simple
   / karatsuba / fft)
d41d8c  // - https://www.e-olymp.com/en/problems/1327: Multiplication
d41d8c  // - https://www.e-olymp.com/en/problems/1328
d41d8c  // - VOJ BIGNUM: Addition, Subtraction, Multiplication.
d41d8c  // - SGU 111: sqrt
d41d8c  // - SGU 193
d41d8c  // - SPOJ MUL, VFMUL: Multiplication.
d41d8c  // - SPOJ FDIV, VFDIV: Division.
d41d8c
d73a77  const int BASE_DIGITS = 9;
82e97b  const int BASE = 1000000000;
d41d8c
6acb6c  struct BigInt {
d65d12      int sign;
a9d078      vector<int> a;
d41d8c
d41d8c      // -------------------- Constructors --------------------
d41d8c      // Default constructor.
1acfca      BigInt() : sign(1) {}
d41d8c
d41d8c      // Constructor from long long.
```

```
ccf902      BigInt(long long v) {
324222          *this = v;
cbb184      }
235125      BigInt& operator = (long long v) {
ce6fc2          sign = 1;
ea2149          if (v < 0) {
6a74a9              sign = -1;
6fab41              v = -v;
cbb184          }
22838a          a.clear();
fefe2d          for (; v > 0; v = v / BASE)
c237f1              a.push_back(v % BASE);
357a55          return *this;
cbb184      }
d41d8c
d41d8c      // Initialize from string.
c710ec      BigInt(const string& s) {
e65d4a          read(s);
cbb184      }
d41d8c
d41d8c      // -------------------- Input / Output -------------------
6c30c4      void read(const string& s) {
ce6fc2          sign = 1;
22838a          a.clear();
bec7a6          int pos = 0;
a68fdf          while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+
    ')) {
dbe226              if (s[pos] == '-')
2b8bd1                  sign = -sign;
17dad0              ++pos;
cbb184          }
7959ef          for (int i = s.size() - 1; i >= pos; i -= BASE_DIGITS) {
c67d6f              int x = 0;
d343c4              for (int j = max(pos, i - BASE_DIGITS + 1); j <= i; j++)
cfc7e4                  x = x * 10 + s[j] - '0';
7c6978              a.push_back(x);
cbb184          }
0ebb65          trim();
cbb184      }
bd2995      friend istream& operator>>(istream &stream, BigInt &v) {
ac0066          string s;
e0c759          stream >> s;
c4002a          v.read(s);
a87cf7          return stream;
cbb184      }
d41d8c
44647f      friend ostream& operator<<(ostream &stream, const BigInt &v) {
b5c525          if (v.sign == -1 && !v.isZero())
27bc2a              stream << '-';
4fda68          stream << (v.a.empty() ? 0 : v.a.back());
```

```
fce618          for (int i = (int) v.a.size() - 2; i >= 0; --i)
018b85              stream << setw(BASE_DIGITS) << setfill('0') << v.a[i];
a87cf7          return stream;
cbb184      }
d41d8c
d41d8c      // -------------------- Comparison --------------------
7014c0      bool operator<(const BigInt &v) const {
eb909f          if (sign != v.sign)
603965              return sign < v.sign;
a2765e          if (a.size() != v.a.size())
f7d303              return a.size() * sign < v.a.size() * v.sign;
305fef          for (int i = ((int) a.size()) - 1; i >= 0; i--)
00d0de              if (a[i] != v.a[i])
2441c5                  return a[i] * sign < v.a[i] * sign;
d1fe4d          return false;
cbb184      }
d41d8c
426053      bool operator>(const BigInt &v) const {
54bd3a          return v < *this;
cbb184      }
65677c      bool operator<=(const BigInt &v) const {
0fe7a0          return !(v < *this);
cbb184      }
605209      bool operator>=(const BigInt &v) const {
d9c542          return !(*this < v);
cbb184      }
880606      bool operator==(const BigInt &v) const {
7f44a6          return !(*this < v) && !(v < *this);
cbb184      }
062171      bool operator!=(const BigInt &v) const {
6c55aa          return *this < v || v < *this;
cbb184      }
d41d8c
d41d8c      // Returns:
d41d8c      // 0 if |x| == |y|
d41d8c      // -1 if |x| < |y|
d41d8c      // 1 if |x| > |y|
ce6386      friend int __compare_abs(const BigInt& x, const BigInt& y) {
e78df5          if (x.a.size() != y.a.size()) {
c86c62              return x.a.size() < y.a.size() ? -1 : 1;
cbb184          }
d41d8c
a552ab          for (int i = ((int) x.a.size()) - 1; i >= 0; --i) {
a5b2df              if (x.a[i] != y.a[i]) {
b1ec3d                  return x.a[i] < y.a[i] ? -1 : 1;
cbb184              }
cbb184          }
bb30ba          return 0;
cbb184      }
d41d8c
```

```
d41d8c      // -------------------- Unary operator - and operators +-
   --------------------
1e3c00      BigInt operator-() const {
18bf1f          BigInt res = *this;
b9607c          if (isZero()) return res;
d41d8c
290faa          res.sign = -sign;
b5053e          return res;
cbb184      }
d41d8c
d41d8c      // Note: sign ignored.
d60e6f      void __internal_add(const BigInt& v) {
f7247c          if (a.size() < v.a.size()) {
2ce41c              a.resize(v.a.size(), 0);
cbb184          }
1addcf          for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size()
   ) || carry; ++i) {
df4512              if (i == (int) a.size()) a.push_back(0);
d41d8c
85e77e              a[i] += carry + (i < (int) v.a.size() ? v.a[i] : 0);
49bff0              carry = a[i] >= BASE;
1791a8              if (carry) a[i] -= BASE;
cbb184          }
cbb184      }
d41d8c
d41d8c      // Note: sign ignored.
8b47dc      void __internal_sub(const BigInt& v) {
65cb2e          for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i
   ) {
a1437d              a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);
e0b1f1              carry = a[i] < 0;
da53a6              if (carry) a[i] += BASE;
cbb184          }
0e329b          this->trim();
cbb184      }
d41d8c
89fb6b      BigInt operator += (const BigInt& v) {
8ea459          if (sign == v.sign) {
570069              __internal_add(v);
9d9745          } else {
ae3659              if (__compare_abs(*this, v) >= 0) {
e9815a                  __internal_sub(v);
9d9745              } else {
dcc3fe                  BigInt vv = v;
3c5f43                  swap(*this, vv);
fe0d8d                  __internal_sub(vv);
cbb184              }
cbb184          }
357a55          return *this;
cbb184      }
```

```
d41d8c
6b1a22      BigInt operator -= (const BigInt& v) {
8ea459          if (sign == v.sign) {
ae3659              if (__compare_abs(*this, v) >= 0) {
e9815a                  __internal_sub(v);
9d9745              } else {
dcc3fe                  BigInt vv = v;
3c5f43                  swap(*this, vv);
fe0d8d                  __internal_sub(vv);
0db96d                  this->sign = -this->sign;
cbb184              }
9d9745          } else {
570069              __internal_add(v);
cbb184          }
357a55          return *this;
cbb184      }
d41d8c
d41d8c      // Optimize operators + and - according to
d41d8c      // https://stackoverflow.com/questions/13166079/move-semantics-
   and-pass-by-rvalue-reference-in-overloaded-arithmetic
f1e02d      template< typename L, typename R >
81c687          typename std::enable_if<
4eceb0              std::is_convertible<L, BigInt>::value &&
c0db24              std::is_convertible<R, BigInt>::value &&
061102              std::is_lvalue_reference<R&&>::value,
6b2030              BigInt>::type friend operator + (L&& l, R&& r) {
46b960          BigInt result(std::forward<L>(l));
fbef75          result += r;
dc8384          return result;
cbb184      }
f1e02d      template< typename L, typename R >
81c687          typename std::enable_if<
4eceb0              std::is_convertible<L, BigInt>::value &&
c0db24              std::is_convertible<R, BigInt>::value &&
bccc2f              std::is_rvalue_reference<R&&>::value,
6b2030              BigInt>::type friend operator + (L&& l, R&& r) {
5f09ae          BigInt result(std::move(r));
a5a040          result += l;
dc8384          return result;
cbb184      }
d41d8c
f1e02d      template< typename L, typename R >
81c687          typename std::enable_if<
4eceb0              std::is_convertible<L, BigInt>::value &&
6ca6cc              std::is_convertible<R, BigInt>::value,
1612ea              BigInt>::type friend operator - (L&& l, R&& r) {
46b960          BigInt result(std::forward<L>(l));
1d15a0          result -= r;
dc8384          return result;
cbb184      }
```

```
d41d8c
d41d8c      // -------------------- Operators * / % --------------------
a179f4      friend pair<BigInt, BigInt> divmod(const BigInt& a1, const BigInt
   & b1) {
872d46          assert(b1 > 0);  // divmod not well-defined for b < 0.
d41d8c
25f4e9          long long norm = BASE / (b1.a.back() + 1);
7c41dc          BigInt a = a1.abs() * norm;
ecd4f4          BigInt b = b1.abs() * norm;
da5ddc          BigInt q = 0, r = 0;
90ee93          q.a.resize(a.a.size());
d41d8c
72b5b8          for (int i = a.a.size() - 1; i >= 0; i--) {
79aca3              r *= BASE;
0caac0              r += a.a[i];
0eeb4e              long long s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.
   size()];
bc1a99              long long s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a
   .size() - 1];
0ebba0              long long d = ((long long) BASE * s1 + s2) / b.a.back();
5d4f85              r -= b * d;
612239              while (r < 0) {
bd3902                  r += b, --d;
cbb184              }
5898c8              q.a[i] = d;
cbb184          }
d41d8c
535024          q.sign = a1.sign * b1.sign;
a29af3          r.sign = a1.sign;
36a918          q.trim();
9a35fd          r.trim();
38a539          auto res = make_pair(q, r / norm);
458098          if (res.second < 0) res.second += b1;
b5053e          return res;
cbb184      }
547e4b      BigInt operator/(const BigInt &v) const {
ce8f7c          return divmod(*this, v).first;
cbb184      }
d41d8c
ee46c3      BigInt operator%(const BigInt &v) const {
7a671a          return divmod(*this, v).second;
cbb184      }
d41d8c
c2998e      void operator/=(int v) {
d1ee66          assert(v > 0);  // operator / not well-defined for v <= 0.
dd9f94          if (llabs(v) >= BASE) {
85cc00              *this /= BigInt(v);
505b97              return ;
cbb184          }
8e679f          if (v < 0)
```

```
20198f              sign = -sign, v = -v;
8e5533          for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
cbe153              long long cur = a[i] + rem * (long long) BASE;
8d1e71              a[i] = (int) (cur / v);
cb35e0              rem = (int) (cur % v);
cbb184          }
0ebb65          trim();
cbb184      }
d41d8c
49658a      BigInt operator/(int v) const {
d1ee66          assert(v > 0);  // operator / not well-defined for v <= 0.
d41d8c
dd9f94          if (llabs(v) >= BASE) {
ed0225              return *this / BigInt(v);
cbb184          }
18bf1f          BigInt res = *this;
37184f          res /= v;
b5053e          return res;
cbb184      }
3b4fa6      void operator/=(const BigInt &v) {
e51f70          *this = *this / v;
cbb184      }
d41d8c
54c35d      long long operator%(long long v) const {
d1ee66          assert(v > 0);  // operator / not well-defined for v <= 0.
a1e888          assert(v < BASE);
cbed95          int m = 0;
947442          for (int i = a.size() - 1; i >= 0; --i)
95269a              m = (a[i] + m * (long long) BASE) % v;
9af577          return m * sign;
cbb184      }
d41d8c
a0b62a      void operator*=(int v) {
dd9f94          if (llabs(v) >= BASE) {
014cdd              *this *= BigInt(v);
505b97              return ;
cbb184          }
8e679f          if (v < 0)
20198f              sign = -sign, v = -v;
c6279c          for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i)
   {
74ab7d              if (i == (int) a.size())
ddfb75                  a.push_back(0);
d09f08              long long cur = a[i] * (long long) v + carry;
98cd39              carry = (int) (cur / BASE);
861843              a[i] = (int) (cur % BASE);
d41d8c              //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur),
   "c"(base));
d41d8c              /*
97f03f               int val;
```

```
ab8362                __asm {
bab6b5                lea esi, cur
6cd1f3                mov eax, [esi]
d5ad3f                mov edx, [esi+4]
378c50                mov ecx, base
d88250                div ecx
e3e615                mov carry, eax
6f8726                mov val, edx;
cbb184                }
26a9ce                a[i] = val;
c4c9bd                */
cbb184            }
0ebb65            trim();
cbb184        }
d41d8c
d1d185        BigInt operator*(int v) const {
dd9f94            if (llabs(v) >= BASE) {
42696e                return *this * BigInt(v);
cbb184            }
18bf1f            BigInt res = *this;
6b38f1            res *= v;
b5053e            return res;
cbb184        }
d41d8c
d41d8c        // Convert BASE 10^old --> 10^new.
ead252        static vector<int> convert_base(const vector<int> &a, int
   old_digits, int new_digits) {
943071            vector<long long> p(max(old_digits, new_digits) + 1);
c4bbd4            p[0] = 1;
85cf8d            for (int i = 1; i < (int) p.size(); i++)
7cc6c9                p[i] = p[i - 1] * 10;
02fb60            vector<int> res;
c6278d            long long cur = 0;
6427c9            int cur_digits = 0;
c0e004            for (int i = 0; i < (int) a.size(); i++) {
b28c31                cur += a[i] * p[cur_digits];
e4696c                cur_digits += old_digits;
5ebda5                while (cur_digits >= new_digits) {
6f203f                    res.push_back((long long)(cur % p[new_digits]));
1cec8a                    cur /= p[new_digits];
318982                    cur_digits -= new_digits;
cbb184                }
cbb184            }
a5eaaa            res.push_back((int) cur);
c5a021            while (!res.empty() && !res.back())
efcb65                res.pop_back();
b5053e            return res;
cbb184        }
d41d8c
009dfc        void fft(vector<complex<double> > & a, bool invert) const {
```

```
8ec808          int n = (int) a.size();
d41d8c
677a94          for (int i = 1, j = 0; i < n; ++i) {
4af5d7              int bit = n >> 1;
425aec              for (; j >= bit; bit >>= 1)
b39a0f                  j -= bit;
297413              j += bit;
9dcc5c              if (i < j)
33275d                  swap(a[i], a[j]);
cbb184          }
d41d8c
eb733a          for (int len = 2; len <= n; len <<= 1) {
2f82ea              double ang = 2 * 3.14159265358979323846 / len * (invert ?
    -1 : 1);
a0b444              complex<double> wlen(cos(ang), sin(ang));
6c8781              for (int i = 0; i < n; i += len) {
c2eaad                  complex<double> w(1);
876230                  for (int j = 0; j < len / 2; ++j) {
371eda                      complex<double> u = a[i + j];
0c0391                      complex<double> v = a[i + j + len / 2] * w;
6c3014                      a[i + j] = u + v;
273255                      a[i + j + len / 2] = u - v;
3e4104                      w *= wlen;
cbb184                  }
cbb184              }
cbb184          }
2111a0          if (invert)
6cb8cc              for (int i = 0; i < n; ++i)
b098a6                  a[i] /= n;
cbb184      }
d41d8c
0d5969      void multiply_fft(const vector<int> &a, const vector<int> &b,
   vector<int> &res) const {
58dd64          vector<complex<double> > fa(a.begin(), a.end());
249aaa          vector<complex<double> > fb(b.begin(), b.end());
43ec81          int n = 1;
727e5e          while (n < (int) max(a.size(), b.size()))
c149a4              n <<= 1;
c149a4          n <<= 1;
37aa6c          fa.resize(n);
870070          fb.resize(n);
d41d8c
3a13f2          fft(fa, false);
c76760          fft(fb, false);
6cb8cc          for (int i = 0; i < n; ++i)
940eb7              fa[i] *= fb[i];
959d01          fft(fa, true);
d41d8c
f38aa2          res.resize(n);
6e20af          long long carry = 0;
```

```
baeb9e          for (int i = 0; i < n; ++i) {
6e6901              long long t = (long long) (fa[i].real() + 0.5) + carry;
9e18f0              carry = t / 1000;
bb5b3b              res[i] = t % 1000;
cbb184          }
cbb184      }
d41d8c
d64466      BigInt mul_simple(const BigInt &v) const {
02a624          BigInt res;
325cfe          res.sign = sign * v.sign;
4bc9af          res.a.resize(a.size() + v.a.size());
7a7093          for (int i = 0; i < (int) a.size(); ++i)
b40a68              if (a[i])
761845                  for (int j = 0, carry = 0; j < (int) v.a.size() ||
   carry; ++j) {
df3e98                      long long cur = res.a[i + j] + (long long) a[i] *
    (j < (int) v.a.size() ? v.a[j] : 0) + carry;
98cd39                      carry = (int) (cur / BASE);
ff01d5                      res.a[i + j] = (int) (cur % BASE);
cbb184                  }
d7ee6d          res.trim();
b5053e          return res;
cbb184      }
d41d8c
ad1556      typedef vector<long long> vll;
d41d8c
4d42f9      static vll karatsubaMultiply(const vll &a, const vll &b) {
94d5f8          int n = a.size();
1fb0e0          vll res(n + n);
44d3ec          if (n <= 32) {
83008c              for (int i = 0; i < n; i++)
f90a6b                  for (int j = 0; j < n; j++)
8dd9af                      res[i + j] += a[i] * b[j];
b5053e              return res;
cbb184          }
d41d8c
af0b16          int k = n >> 1;
f9fca2          vll a1(a.begin(), a.begin() + k);
72c0c7          vll a2(a.begin() + k, a.end());
48ebf6          vll b1(b.begin(), b.begin() + k);
88c9a6          vll b2(b.begin() + k, b.end());
d41d8c
03c868          vll a1b1 = karatsubaMultiply(a1, b1);
e56678          vll a2b2 = karatsubaMultiply(a2, b2);
d41d8c
40d6ad          for (int i = 0; i < k; i++)
c20ed7              a2[i] += a1[i];
40d6ad          for (int i = 0; i < k; i++)
b009cc              b2[i] += b1[i];
d41d8c
```

```
6a2f29          vll r = karatsubaMultiply(a2, b2);
be9bd2          for (int i = 0; i < (int) a1b1.size(); i++)
47fef2              r[i] -= a1b1[i];
cf04ec          for (int i = 0; i < (int) a2b2.size(); i++)
00a00c              r[i] -= a2b2[i];
d41d8c
5951a9          for (int i = 0; i < (int) r.size(); i++)
1bf61e              res[i + k] += r[i];
be9bd2          for (int i = 0; i < (int) a1b1.size(); i++)
d6cf88              res[i] += a1b1[i];
cf04ec          for (int i = 0; i < (int) a2b2.size(); i++)
ab9916              res[i + n] += a2b2[i];
b5053e          return res;
cbb184      }
d41d8c
287510      BigInt mul_karatsuba(const BigInt &v) const {
48c647          vector<int> a6 = convert_base(this->a, BASE_DIGITS, 6);
f64a05          vector<int> b6 = convert_base(v.a, BASE_DIGITS, 6);
e1cb30          vll a(a6.begin(), a6.end());
5ed74f          vll b(b6.begin(), b6.end());
1a813e          while (a.size() < b.size())
ddfb75              a.push_back(0);
0d118e          while (b.size() < a.size())
c40831              b.push_back(0);
634b60          while (a.size() & (a.size() - 1))
eed3fb              a.push_back(0), b.push_back(0);
16bf35          vll c = karatsubaMultiply(a, b);
02a624          BigInt res;
325cfe          res.sign = sign * v.sign;
6e20af          long long carry = 0;
7dbc9f          for (int i = 0; i < (int) c.size(); i++) {
dc97b8              long long cur = c[i] + carry;
cdf472              res.a.push_back((int) (cur % 1000000));
735fb2              carry = cur / 1000000;
cbb184          }
7b10c4          res.a = convert_base(res.a, 6, BASE_DIGITS);
d7ee6d          res.trim();
b5053e          return res;
cbb184      }
d41d8c
933d02      void operator*=(const BigInt &v) {
fa4bc1          *this = *this * v;
cbb184      }
24478f      BigInt operator*(const BigInt &v) const {
de6792          if (a.size() * v.a.size() <= 1000111) return mul_simple(v);
fec548          if (a.size() > 500111 || v.a.size() > 500111) return mul_fft(
    v);
a67c32          return mul_karatsuba(v);
cbb184      }
d41d8c
```

```
0f0ce5      BigInt mul_fft(const BigInt& v) const {
02a624          BigInt res;
325cfe          res.sign = sign * v.sign;
d1a018          multiply_fft(convert_base(a, BASE_DIGITS, 3), convert_base(v.
   a, BASE_DIGITS, 3), res.a);
74be5c          res.a = convert_base(res.a, 3, BASE_DIGITS);
d7ee6d          res.trim();
b5053e          return res;
cbb184      }
d41d8c
d41d8c      // -------------------- Misc --------------------
9f0aff      BigInt abs() const {
18bf1f          BigInt res = *this;
3ccc69          res.sign *= res.sign;
b5053e          return res;
cbb184      }
a0fac1      void trim() {
b03a9b          while (!a.empty() && !a.back())
4685a5              a.pop_back();
e28510          if (a.empty())
ce6fc2              sign = 1;
cbb184      }
d41d8c
88d324      bool isZero() const {
5c0518          return a.empty() || (a.size() == 1 && !a[0]);
cbb184      }
d41d8c
e7ccd6      friend BigInt gcd(const BigInt &a, const BigInt &b) {
183a15          return b.isZero() ? a : gcd(b, a % b);
cbb184      }
7977e6      friend BigInt lcm(const BigInt &a, const BigInt &b) {
8b81ac          return a / gcd(a, b) * b;
cbb184      }
d41d8c
2f7166      friend BigInt sqrt(const BigInt &a1) {
b25149          BigInt a = a1;
53b77e          while (a.a.empty() || a.a.size() % 2 == 1)
8a6b34              a.a.push_back(0);
d41d8c
0c5896          int n = a.a.size();
d41d8c
f9194d          int firstDigit = (int) sqrt((double) a.a[n - 1] * BASE + a.a[
   n - 2]);
3c7b49          int norm = BASE / (firstDigit + 1);
b65c20          a *= norm;
b65c20          a *= norm;
53b77e          while (a.a.empty() || a.a.size() % 2 == 1)
8a6b34              a.a.push_back(0);
d41d8c
8a28a4          BigInt r = (long long) a.a[n - 1] * BASE + a.a[n - 2];
```

```
4e5685          firstDigit = (int) sqrt((double) a.a[n - 1] * BASE + a.a[n -
   2]);
97c0e8          int q = firstDigit;
02a624          BigInt res;
d41d8c
a1054f          for(int j = n / 2 - 1; j >= 0; j--) {
e63f29              for(; ; --q) {
592185                  BigInt r1 = (r - (res * 2 * BigInt(BASE) + q) * q) *
   BigInt(BASE) * BigInt(BASE) + (j > 0 ? (long long) a.a[2 * j - 1] * BASE +
    a.a[2 * j - 2] : 0);
60f563                  if (r1 >= 0) {
01144f                      r = r1;
c2bef1                      break;
cbb184                  }
cbb184              }
d2c0d8              res *= BASE;
f2637e              res += q;
d41d8c
e79d0e              if (j > 0) {
febb34                  int d1 = res.a.size() + 2 < r.a.size() ? r.a[res.a.
   size() + 2] : 0;
baacce                  int d2 = res.a.size() + 1 < r.a.size() ? r.a[res.a.
   size() + 1] : 0;
78b193                  int d3 = res.a.size() < r.a.size() ? r.a[res.a.size()
   ] : 0;
7d925d                  q = ((long long) d1 * BASE * BASE + (long long) d2 *
   BASE + d3) / (firstDigit * 2);
cbb184              }
cbb184          }
d41d8c
d7ee6d          res.trim();
28ae5c          return res / norm;
cbb184      }
2145c1  };
f1f35b
```

7.2 FFT

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
18a91e    Modular Inverse:
f7b296      FFT allows multiplication of two polynomials in O(n log n).
420c7a      This can also be used to multiply two long numbers faster.
c00ff6      Other applications:
c35b73      - All possible sums of two arrays.
1da5a4      - Dot product of vector a with every cyclic shift of vector b.
49afe3      - Attaching two boolean stripes without two 1s next to each other
   .
52f6a3      - String matching.
```

```
d41d8c
b95cae    Usage:
afb0f5      long double is a lot slower. 3s with ld and 0.7 with double for
   10^6 size vectors.
d41d8c
1d1558    Source: https://cp-algorithms.com/algebra/fft.html
c4c9bd  */
d41d8c
f4f8e6  using cd = complex<long double>;
c4f8de  const ld PI = acos(-1.0L);
d41d8c
9b5b94  void fft(vector<cd> &a, bool invert)
f95b70  {
94d5f8    int n = a.size();
d41d8c
d94885    for (int i = 1, j = 0; i < n; i++)
f95b70    {
4af5d7      int bit = n >> 1;
474fac      for (; j & bit; bit >>= 1)
53c7ca        j ^= bit;
53c7ca      j ^= bit;
d41d8c
9dcc5c      if (i < j)
33275d        swap(a[i], a[j]);
cbb184    }
d41d8c
2fe9ad    for (int len = 2; len <= n; len <<= 1)
f95b70    {
c19c97      ld ang = 2 * PI / len * (invert ? -1 : 1);
808a0b      cd wlen(cos(ang), sin(ang));
3dd9d3      for (int i = 0; i < n; i += len)
f95b70      {
8c3c80        cd w(1);
5594fb        for (int j = 0; j < len / 2; j++)
f95b70        {
cf0824          cd u = a[i + j], v = a[i + j + len / 2] * w;
6c3014          a[i + j] = u + v;
273255          a[i + j + len / 2] = u - v;
3e4104          w *= wlen;
cbb184        }
cbb184      }
cbb184    }
d41d8c
2111a0    if (invert)
f95b70    {
0b5665      for (cd &x : a)
b6d31b        x /= n;
cbb184    }
cbb184  }
d41d8c
```

```
d41d8c  // Input a[0] + a[1]x + a[2]x^2 ...
d41d8c  // Returns polynomial of size equal to the smallest power of two at
   least
d41d8c  // as large as a.size() + b.size(). This can have some extra zeros.
d41d8c  // Use long double if using long long.
4fce64  template <class T>
a3a2ed  vector<T> multiply(vector<T> const &a, vector<T> const &b)
f95b70  {
6fa6b9    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
43ec81    int n = 1;
cd5a64    while (n < a.size() + b.size())
c149a4      n <<= 1;
37aa6c    fa.resize(n);
870070    fb.resize(n);
d41d8c
3a13f2    fft(fa, false);
c76760    fft(fb, false);
83008c    for (int i = 0; i < n; i++)
940eb7      fa[i] *= fb[i];
959d01    fft(fa, true);
d41d8c
ebf3b6    vector<T> result(n);
83008c    for (int i = 0; i < n; i++)
4c9bb2      result[i] = round(fa[i].real()); // Remember to remove rounding
   if working with floats.
dc8384    return result;
cbb184  }
0402dc
```

7.3 Fraction

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
390211    Fraction representation:
4d1181      All operations run in O(gcd) = O(log).
d41d8c
b95cae    Usage:
70e7d7      Don't modify internal values, use constructor.
408ef0      Some nice things about the constructor: frac() = 0/1, frac(5) =
   5/1.
d41d8c
b8d28c      Be careful that the numerator and denominator might overflow if
   lcm is too big.
20fa30      In those cases, you can always do frac<big_int>, but that will be
    painful to code.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
```

```
4fce64  template <class T>
4cf1ca  struct frac
f95b70  {
e75828    T a, b; // b can't be negative, very important.
d41d8c
191fc6    explicit frac(T a = 0, T b = 1) : a(a), b(b) { simpl(); }
d41d8c
7d70f7    void simpl()
f95b70    {
8eb5bb      T g = __gcd(abs(a), abs(b)) * sign(b); // Make b positive.
fe7245      a /= g;
ee2d42      b /= g;
cbb184    }
d41d8c
d59b8a    bool operator<(const frac &rhs) const
f95b70    {
5c6427      return a * rhs.b < rhs.a * b;
cbb184    }
d41d8c
7ebf19    bool operator>(const frac &rhs) const
f95b70    {
2ab79c      return rhs < *this;
cbb184    }
d41d8c
d60bf3    bool operator==(const frac &rhs) const // TODO: untested.
f95b70    {
77c0b8      return !(*this < rhs) && !(rhs < *this);
cbb184    }
d41d8c
473b74    frac operator*(const frac &rhs) const
f95b70    {
f0117d      return frac(a * rhs.a, b * rhs.b);
cbb184    }
d41d8c
04b5a1    frac operator+(const frac &rhs) const
f95b70    {
3ff11f      T m = (b * rhs.b) / __gcd(b, rhs.b);
24edd6      return frac(a * (m / b) + rhs.a * (m / rhs.b), m);
cbb184    }
d41d8c
c8ca1d    frac operator-(void) const
f95b70    {
132fb3      return frac(-a, b);
cbb184    }
d41d8c
de243f    frac operator-(const frac &rhs) const
f95b70    {
111760      return (*this) + (-rhs);
cbb184    }
d41d8c
```

```
d63a85    frac operator/(const frac &rhs) const
f95b70    {
f5299b      return (*this) * frac(rhs.b, rhs.a);
cbb184    }
d41d8c
9e018a    friend ostream &operator<<(ostream &os, const frac &f)
f95b70    {
891d94      return os << f.a << "/" << f.b;
cbb184    }
2145c1  };
d41d8c
c8862e
```

7.4 Integration

```
d41d8c  /*
f64ead    Numerical Integration:
49d5e8      Given a function f and an interval [a, b] estimates integral of f
   (x) dx from a to b.
bfe460      Error is in theory inversely proportional to n^4.
d41d8c
b95cae    Usage:
be1ead      n, the number of intervals must be even.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
044d82  template <class F>
7d9945  double simpsons(const F &f, int n /* even */, double a, double b)
f95b70  {
46af34    double retv = f(a) + f(b);
d025af    double h = (b - a) / n;
acfc81    for (int i = 1; i < n; i += 2)
900086      retv += 4 * f(a + i * h);
1c3900    for (int i = 2; i < n; i += 2)
6c1313      retv += 2 * f(a + i * h);
d41d8c
055fe5    retv *= h / 3;
6272cf    return retv;
cbb184  }
d41d8c
d41d8c  // Sample usage:
d41d8c  // int main(void)
d41d8c  // {
d41d8c  //  printf("%.20lf\n", simpsons([](double x) { return pow(sin(M_PI *
   x / 2.0), 3.2);}, 2000, 0, 2));
d41d8c  // }
caa0e5
```

7.5 linalg

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
f92339    Vector and matrix operations:
687bbc      Details are given in each function.
3ab55f      vec inherits from vector<T>, so there is a lot you can do with it
   .
5ae524      Also, mat inherits from vector<vec<T>>.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
d41d8c
1ef4c8    Source: some of it from https://github.com/kth-competitive-
   programming/kactl/blob/master/content/numerical/MatrixInverse.h
c4c9bd  */
d41d8c
4fce64  template <class T>
fe4002  struct vec : vector<T>
f95b70  {
469362    vec(int n) : vector<T>(n) {}
d41d8c
d41d8c    // c = a*x + b*y
e918cb    static void linear_comb(const vec &a, T x, const vec &b, T y, vec &
   c)
f95b70    {
8fe753      for (int i = 0; i < sz(a); i++)
75e753        c[i] = a[i] * x + b[i] * y;
cbb184    }
d41d8c
d41d8c    // return a*x + b*y
250f88    static vec linear_comb(vec a, T x, const vec &b, T y)
f95b70    {
4fec85      linear_comb(a, x, b, y, a);
3f5343      return a;
cbb184    }
2145c1  };
d41d8c
4fce64  template <class T>
dade1f  struct mat : vector<vec<T>>
f95b70  {
d41d8c    // Creates a zero-filled matrix of n rows and m columns.
2d2b5d    mat(int n, int m) : vector<vec<T>>(n, vec<T>(m)) {}
d41d8c
d41d8c    // c = a * x + b * y
762fbc    static void linear_comb(const mat &a, T x, const mat &b, T y, mat &
   c)
f95b70    {
8fe753      for (int i = 0; i < sz(a); i++)
f47ed7        for (int j = 0; j < sz(a[i]); j++)
4f844b          c[i][j] = a[i][j] * x + b[i][j] * y;
cbb184    }
```

```
d41d8c
d41d8c    // return a * x + b * y
08e6ea    static mat linear_comb(mat a, T x, const mat &b, T y)
f95b70    {
4fec85      linear_comb(a, x, b, y, a);
3f5343      return a;
cbb184    }
d41d8c
13fd2a    mat operator-(const mat &b) const { return linear_comb(*this, T(1),
    b, T(-1)); }
d41d8c
0138fa    mat operator+(const mat &b) const { return linear_comb(*this, T(1),
    b, T(1)); }
d41d8c
93d3e8    mat operator*(const T &x) { return linear_comb(*this, x, *this, T
   (0)); }
d41d8c
d41d8c    // Absolutely does not work for int.
72c1fd    mat operator/(const T &x) const { return linear_comb(*this, T(1) /
   x, *this, T(0)); }
d41d8c
d41d8c    // Returns inverse of matrix (assuming it is square and non-
   singular). Runs in O(n^3).
d41d8c    // Absolutely does not work for int.
14566d    mat inverse() // TODO: test singular.
f95b70    {
d23a72      int n = sz(*this);
bca455      mat a(n, 2 * n);  // A is Nx2N: X|I.
f7f2d1      vector<int> col(n); // Will be using column pivoting, so need to
   remember original columns.
83008c      for (int i = 0; i < n; i++)
f95b70      {
f90a6b        for (int j = 0; j < n; j++)
c1c7c0          a[i][j] = (*this)[i][j];
34ac5b        a[i][i + n] = T(1);
6dcd38        col[i] = i;
cbb184      }
d41d8c
83008c      for (int i = 0; i < n; i++)
f95b70      {
903ccf        int r = i, c = i;
775cab        for (int j = i; j < n; j++)
90f1d8          for (int k = i; k < n; k++)
f78c7f            if (abs(a[j][k]) > abs(a[r][c]))
d4c894              r = j, c = k;
d41d8c
d41d8c        // assert(abs(a[r][c]) > EPS); Uncomment to check singular
   matrix
a2fa24        swap(a[i], a[r]);
d41d8c
```

```
f90a6b        for (int j = 0; j < n; j++)
c8cc8f          swap(a[j][i], a[j][c]), swap(a[j][i + n], a[j][c + n]);
c1d48e        swap(col[i], col[c]);
d41d8c
b70d15        vec<T>::linear_comb(a[i], T(1) / a[i][i], a[i], T(0), a[i]);
67830d        a[i][i] = T(1);
d41d8c
197ab1        for (int j = i + 1; j < n; j++)
3704dc          vec<T>::linear_comb(a[j], T(1), a[i], -a[j][i], a[j]);
cbb184      }
d41d8c
d41d8c      // Right now A is:
d41d8c      //
d41d8c      //  1 * *
d41d8c      //  0 1 *
d41d8c      //  0 0 1
d41d8c      //
d41d8c      // Next we remove non-1s from right to left.
d41d8c
917d8b      for (int i = n - 1; i > 0; i--)
c791cd        for (int j = 0; j < i; j++)
3704dc          vec<T>::linear_comb(a[j], T(1), a[i], -a[j][i], a[j]);
d41d8c
c70ad2      mat retv(n, n);
83008c      for (int i = 0; i < n; i++)
f90a6b        for (int j = 0; j < n; j++)
4eb40a          retv[col[i]][col[j]] = a[i][j + n];
6272cf      return retv;
cbb184    }
2145c1  };
2457f9
```

7.6 Simplex

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
458b90    Simplex:
6956ec      Optimizes a linear program of the form:
15b127        maximize c*x, s.t. a*x <ops> b, x >= 0.
7b88d6      Each constraint can use a different operator from {<= >= ==}.
8aa76d      Not polynomial, but got AC 150 ms with 4000 constraints and 200
   variables.
d41d8c
b95cae    Usage:
e8b3b7      Call run_simplex, with the number of constraints and variables, a
   , b, ops and c (as specified above).
34036d      Return value is ok if solution was found, unbounded if objective
   value can be infinitely large
eb42f2      or infeasible if there is no solution given the constraints.
```

```
d41d8c
2baa60      The value of each variable is returned in vector res. Objective
   function optimal value is also returned.
060dc4      Sample usage is commented below.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
4fce64  template <class T>
fe4002  struct vec : vector<T>
f95b70  {
469362    vec(int n) : vector<T>(n) {}
d41d8c
d41d8c    // c = a*x + b*y
e918cb    static void linear_comb(const vec &a, T x, const vec &b, T y, vec &
   c)
f95b70    {
8fe753      for (int i = 0; i < sz(a); i++)
75e753        c[i] = a[i] * x + b[i] * y;
cbb184    }
2145c1  };
d41d8c
4fce64  template <class T>
dade1f  struct mat : vector<vec<T>>
f95b70  {
d41d8c    // Creates a zero-filled matrix of n rows and m columns.
2d2b5d    mat(int n, int m) : vector<vec<T>>(n, vec<T>(m)) {}
d41d8c
d41d8c    // Erase row O(n^2).
82436c      void erase_row(int i)
f95b70      {
7c9f9f          this->erase(this->begin() + i);
cbb184      }
d41d8c
d41d8c    // Erase column O(n^2).
1b22c6      void erase_col(int j)
f95b70      {
798fc8          for (int i = 0; i < sz(*this); i++)
a7796a              (*this)[i].erase((*this)[i].begin() + j);
cbb184      }
2145c1  };
d41d8c
d3ff82  namespace simplex
f95b70  {
d41d8c  // Any value within [-EPS, +EPS] will be considered equal to 0.
05667a  const double EPS = 1e-6;
d41d8c
5e6f5b  enum op { ge, le, eq };
d41d8c
242dbb  enum optimization_status { ok, unbouded, infeasible };
```

```
d41d8c
4d9580  int get_entering_var(mat<double> &tab)
f95b70  {
d41d8c    // Get first non-artificial variable with negative objective
   coeficient. If none, return -1. (could instead return most negative, but
   that could cycle)
682f62    for (int i = 0; i < sz(tab[0]) - 1; i++)
72e0d2      if (tab[0][i] < -EPS)
d9a594        return i;
daa4d1    return -1;
cbb184  }
d41d8c
201003  int get_exiting_var_row(mat<double> &tab, int entering_var)
f95b70  {
d41d8c    // Get smallest value of val and first in case of tie. If none,
   return -1.
fcb2fc    int retv = -1;
6213b9    double val = -1.0;
a07064    for (int i = 1; i < sz(tab); i++)
f95b70    {
d41d8c      // If strictly positive, it bounds the entering var.
dcda72      if (tab[i][entering_var] > EPS)
f95b70      {
d41d8c        // Entering var will be bounded by tab[i][tab.size().second -
   1] / tab[i][entering_var].
d41d8c        // val could be slightly negative if tab[i][tab.size().second -
    1] = -0.
393d3f        if (val == -1.0 || tab[i][sz(tab[i]) - 1] / tab[i][entering_var
   ] < val)
f95b70        {
78d87c          val = tab[i][sz(tab[i]) - 1] / tab[i][entering_var];
52cece          retv = i;
cbb184        }
cbb184      }
cbb184    }
d41d8c
6272cf    return retv;
cbb184  }
d41d8c
ed25d2  optimization_status solve_tab(mat<double> &tab, vector<int> &
   basic_var)
f95b70  {
d41d8c    // artificial_count is the number of variables at the end we should
    ignore.
a17ec7    int entering_var;
6b7846    while ((entering_var = get_entering_var(tab)) != -1)
f95b70    {
6c0a23      int exiting_var_row = get_exiting_var_row(tab, entering_var);
d41d8c
d41d8c      // If no exiting variable bounds the entering variable, the
```

```
   objective is unbounded.
813335      if (exiting_var_row == -1)
914a2e        return optimization_status::unbouded;
d41d8c
d41d8c      // Set new basic var coeficient to 1.
89c7a2      vec<double>::linear_comb(tab[exiting_var_row], (1.0 / tab[
   exiting_var_row][entering_var]), tab[exiting_var_row], 0.0, tab[
   exiting_var_row]);
d41d8c
d41d8c      // Gaussian elimination of the other rows.
c7a773      for (int i = 0; i < sz(tab); i++)
81c379        if (i != exiting_var_row)
ed2730          if (abs(tab[i][entering_var]) > EPS)
7ad878            vec<double>::linear_comb(tab[i], 1.0, tab[exiting_var_row],
    -tab[i][entering_var], tab[i]);
d41d8c
64dd6a      basic_var[exiting_var_row] = entering_var;
cbb184    }
d41d8c
c52f1c    return optimization_status::ok;
cbb184  }
d41d8c
d41d8c  // maximize c*x, s.t. a*x <ops> b. x >= 0.
f1a105  optimization_status run_simplex(int num_constraints, int num_vars,
   mat<double> a, vec<op> ops, vec<double> b, vec<double> c, vec<double> &res
   , double &obj_val)
f95b70  {
334f46    for (int i = 0; i < num_constraints; i++)
5f946c      if (ops[i] == op::ge)
f95b70      {
d41d8c        // Beyond this point "ge" constraints won't exist.
44438f        vec<double>::linear_comb(a[i], -1, a[i], 0, a[i]); // a[i] *=
   -1;
250b4d        b[i] *= -1;
1c38d4        ops[i] = op::le;
cbb184      }
d41d8c
0264da    int num_artificial_variables = 0;
371f2b    int num_slack_variables = 0;
334f46    for (int i = 0; i < num_constraints; i++)
f95b70    {
0ec40f      if (ops[i] == op::le)
f95b70      {
37acf9        num_slack_variables++;
cbb184      }
d41d8c
359aa4      if ((ops[i] == op::le && b[i] < -EPS) || ops[i] == op::eq)
f95b70      {
d41d8c        // If we have rhs strictly negative in a inequality or an
   equality constraint, we need an artificial val.
```

```
fc36e6        num_artificial_variables++;
cbb184      }
cbb184    }
d41d8c
854c33    mat<double> tab(num_constraints + 1, num_vars + num_slack_variables
    + num_artificial_variables + 1);
9a9a70    vector<int> basic_var(num_constraints + 1);
775265    vector<int> slack_cols, artificial_cols;
7f63aa    for (int i = num_vars; i < num_vars + num_slack_variables; i++)
10c71f      slack_cols.push_back(i);
e0b615    for (int i = num_vars + num_slack_variables; i < num_vars +
   num_slack_variables + num_artificial_variables; i++)
eafbfb      artificial_cols.push_back(i);
c70a50    int rhs_col = num_vars + num_slack_variables +
   num_artificial_variables;
d41d8c
d41d8c    // First objective will be to have artificial variables equal to 0.
017565    for (int i : artificial_cols)
b98201      tab[0][i] = 1;
d41d8c
9c49f5    for (int i = 0, k = 0, l = 0; i < num_constraints; i++)
f95b70    {
861a15      for (int j = 0; j < num_vars; j++)
e3832e        tab[i + 1][j] = a[i][j];
d41d8c
0ec40f      if (ops[i] == op::le)
141495        tab[i + 1][slack_cols[l++]] = 1;
d41d8c
142f37      tab[i + 1][rhs_col] = b[i];
d41d8c
359aa4      if ((ops[i] == op::le && b[i] < -EPS) || ops[i] == op::eq) //
   Basic var will be artificial
f95b70      {
2a6978        if (b[i] < -EPS)
009fda          vec<double>::linear_comb(tab[i + 1], -1, tab[i + 1], 0, tab[i
    + 1]); // a[i] *= -1;
d41d8c
86fab4        tab[i + 1][artificial_cols[k++]] = 1;
116454        basic_var[i + 1] = artificial_cols[k - 1];
d41d8c
06db08        vec<double>::linear_comb(tab[0], 1.0, tab[i + 1], -1.0, tab[0])
   ;
cbb184      }
2954e9      else // Basic var will be slack var.
f95b70      {
ae77b6        basic_var[i + 1] = slack_cols[l - 1];
cbb184      }
cbb184    }
d41d8c
df8d17    assert(solve_tab(tab, basic_var) == optimization_status::ok);
```

```
d41d8c
d41d8c    // Best solution could not bring artificial variables to 0 (
   objective max Z = sum(-xa)).
fe0d64    if (tab[0][sz(tab[0]) - 1] < -EPS)
94b8a3      return optimization_status::infeasible;
d41d8c
d41d8c    // If we have an artificial variable on the base with xb = 0, we
   need to remove it.
e6411b    for (int i = 1; i < sz(basic_var); i++)
0778cb      if (basic_var[i] >= num_vars + num_slack_variables)
f95b70      {
d41d8c        // Find non-artificial replacement.
e2f213        for (int j = 0; j < sz(tab[i]) - 1 - num_artificial_variables;
   j++)
f95b70        {
d41d8c          // If non-zero value in row, we can replace.
a8880b          if (j != basic_var[i] && abs(tab[i][j]) > EPS)
f95b70          {
d41d8c            // Remove from the other rows.
b5fa44            vec<double>::linear_comb(tab[i], 1.0 / tab[i][j], tab[i],
   0, tab[i]);
d41d8c
443db5            for (int k = 0; k < sz(tab); k++)
635b4c              if (k != i)
f95b70              {
e76184                if (abs(tab[k][j]) > EPS)
4b6b27                  vec<double>::linear_comb(tab[k], 1, tab[i], -tab[k][j
   ], tab[k]);
cbb184              }
d41d8c
d41d8c            // Basic variable replacemente done, so proceed to next
   basic_var.
7e0f27            basic_var[i] = j;
c2bef1            break;
cbb184          }
cbb184        }
cbb184      }
d41d8c
ca2210    for (int i = sz(tab) - 1; i > 0; i--)
0778cb      if (basic_var[i] >= num_vars + num_slack_variables)
f95b70      {
d41d8c        // Could not replace basic var, so constraint is redundant.
2cd1fb        tab.erase_row(i);
fe14c7        basic_var.erase(basic_var.begin() + i);
cbb184      }
d41d8c
d41d8c    // Remove artificial variable columns.
5c3178    for (int i = sz(artificial_cols) - 1; i >= 0; i--)
9a226e      tab.erase_col(artificial_cols[i]);
d41d8c
```

```
1311b7    for (int i = 0; i < sz(tab[0]); i++)
d2677f      tab[0][i] = 0;
f17293    for (int i = 0; i < num_vars; i++)
94256d      tab[0][i] = -c[i];
d41d8c
a07064    for (int i = 1; i < sz(tab); i++)
b39526      vec<double>::linear_comb(tab[0], 1, tab[i], -tab[0][basic_var[i
   ]], tab[0]);
d41d8c
54ad02    optimization_status status = solve_tab(tab, basic_var);
d41d8c
b68670    res = vec<double>(num_vars);
e6411b    for (int i = 1; i < sz(basic_var); i++)
047b20      if (basic_var[i] < num_vars)
81f54e        res[basic_var[i]] = tab[i][sz(tab[i]) - 1];
d41d8c
a3473e    obj_val = tab[0][sz(tab[0]) - 1];
d41d8c
62d3d5    return status;
cbb184  }
cbb184  } // namespace simplex
d41d8c
d41d8c  /*
13a4b1  int main(void)
f95b70  {
14e0a7    int n, m;
aa3380    cin >> n >> m;
d41d8c
37ce14    int num_constraints = m, num_vars = n;
d41d8c
d41d8c    // maximize c*x, s.t. a*x <ops> b. x >= 0.
2626bb    mat<double> a(num_constraints, num_vars);
84d434    vec<double> b(num_constraints);
01b2af    vec<simplex::op> ops(num_constraints);
dabb12    vec<double> c(num_vars);
40ca17    vec<double> res(num_vars);
d41d8c
83008c    for (int i = 0; i < n; i++)
a733f7      cin >> c[i];
d41d8c
94f72b    for (int i = 0; i < m; i++)
f95b70    {
7ba74c      int l, r, x;
15994b      cin >> l >> r >> x;
0dfebd      for (int j = l - 1; j <= r - 1; j++)
a21125        a[i][j] = 1;
df0b9d      b[i] = x;
80367f      ops[i] = simplex::op::le;
cbb184    }
d41d8c
```

```
1afc12    double ans;
dd6c28    simplex::run_simplex(num_constraints, num_vars, a, ops, b, c, res,
   ans);
d41d8c
530b75    cout << ((long long)(ans + 0.5)) << endl;
cbb184  }
c4c9bd  */
46f321
```

8 String

8.1 KMP

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
8dec4f    Prefix Function and KMP:
e45403      Computes prefix function for a given string in O(n).
16bb22      String matching in O(n + m).
37f784      No need to be strings, you can use vector<int> since the
be2fe6      algorithms don't depend on the alphabet size, they only perform
   equality comparisons.
b5efd9      Usage is explained in each function.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
d41d8c  // Returns the prefix function for the given string.
d41d8c  // pi[i] for 0 <= i <= s.size() (s.size() + 1 elements).
d41d8c  // pi[i] considers the prefix of string s having size i.
d41d8c  // pi[i] is the size of its (the prefix's) largest proper prefix
   which is also a suffix.
d41d8c  // For "aabaaab", pi is is {0,0,1,0,1,2,2,3}
4fce64  template <class T>
8fa849  vector<int> prefix_function(T s)
f95b70  {
d2c5d5    vector<int> pi(s.size() + 1, 0);
a94e4a    for (int i = 2; i <= s.size(); i++)
f95b70    {
3f878c      int j = pi[i - 1];          // j is the size of the candidate
   prefix to expand.
4b3f35      while (j > 0 && s[j] != s[i - 1]) // While we still have a
   candidate prefix and it can't be expanded.
187475        j = pi[j];            // Go to the next candidate prefix.
d41d8c
d41d8c      // If candidate prefix can be expanded, do it. Otherwise, there
   is no prefix that is also a suffix.
f986f8      pi[i] = s[j] == s[i - 1] ? j + 1 : 0;
cbb184    }
d41d8c
81d1a2    return pi;
cbb184  }
d41d8c
d41d8c  // Returns a sorted list of all positions in the text string where
   begins an ocurrence of the key string.
d41d8c  // e.g. kmp("aabaaab", "aab") returns {0, 4}.
4fce64  template <class T>
15b377  vector<int> kmp(T text, T key)
f95b70  {
```

```
aeb888    vector<int> retv;
7fa638    vector<int> pi = prefix_function(key);
5d936d    for (int i = 0, match = 0; i < text.size(); i++) // There is no
   need to have the entire text in memory, you could do this char by char.
f95b70    {
d41d8c      // match stores the size of the prefix of the key which is a
   suffix of the current processed text.
9d984d      while (match > 0 && text[i] != key[match])
7eb4cc        match = pi[match];
db8319      if (text[i] == key[match])
24b638        match++;
d41d8c
dd8c14      if (match == key.size())
f95b70      {
7b8421        retv.push_back(i - match + 1);
7eb4cc        match = pi[match]; // To avoid access to key[key.size()] in
   next iteration.
cbb184      }
cbb184    }
d41d8c
6272cf    return retv;
cbb184  }
415801
```

8.2 Acho Corasick

```
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
30562e    Aho-Corasick: O(alpha_size * string_sum)
4e9057      In general, multiple pattern string matching tree/automaton.
d41d8c
fbc6b5      Keep in mind that find_all can be O(N*sqrt(N)) if no duplicate
   patterns. (N is total string length)
d41d8c
ca2095    Constraints:
00d37a      chars in the string are all in the interval [first, first +
   alpha_size - 1].
3da079      This will not free some memory on object destruction.
390590      Duplicate patterns are allowed, empty patterns are not.
d41d8c
b95cae    Usage:
df3a72      Set alpha_size and the first char in the alphabet.
e98cb2      Call constructor passing the list of pattern strings.
0a657b      Use one of find, find_all ... to process a text or do your own
   thing.
acdd39      To find the longest words that start at each position, reverse
   all input.
3439d1      Bottleneck in this code is memory allocation.
91a84c      For 10^6 total string size, memory usage can be up to 300 Mb.
```

```
d41d8c
b34145      You can save time:
93df09        list_node, match_list, match_list_last are only needed to list
   all matches.
57e4db        atm automaton table can be cut to reduce memory usage.
018d49        The text processing stuff is also optional.
02e3ad        Node memory can be one big array instead of vector.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
e7f92a  struct aho_corasick
f95b70  {
da45ec    enum
f95b70    {
033315      alpha_size = 26,  // Number of chars in the alphabet.
b3d02f      first = 'a'     // First char.
2145c1    };
d41d8c
fc487b    struct list_node    // Simple linked list node struct.
f95b70    {
53e65f      int id;
6ec94b      list_node *next;
ff56a7      explicit list_node(int id, list_node *next) : id(id), next(next)
   {}
2145c1    };
d41d8c
e4accb    struct node
f95b70    {
ca8b7e      int fail = -1;      // node failure link (aka suffix link).
2eb620      int nmatches = 0;   // Number of matches ending in this node.
9005b9      int next[alpha_size]; // Next node in trie for each letter.
   Replace with unordered_map or list if memory is tight.
c0f747      int atm[alpha_size];  // Optional: Automaton state transition
   table. Simpler text processing.
d41d8c
44edb6      list_node *match_list = nullptr;    // Pointer to first node in
   linked list of matches. List ends with null pointer.
01009c      list_node *match_list_last = nullptr; // Internal: pointer to
   last node in list of matches (before bfs), or null if empty list.
d41d8c
e6fb82      node() { memset(next, -1, sizeof(next)); } // Start with all
   invalid transitions.
2145c1    };
d41d8c
b9ea22    vector<node> nodes;
d41d8c
9b61f6    aho_corasick(const vector<string> &pats)
f95b70    {
225eb3      nodes.emplace_back(); // Make root node 0.
```

```
b5bf96      for (int i = 0; i < sz(pats); i++)
f95b70      {
b3da3c        int cur = 0;  // Start from root.
9f5c69        for (int j = 0; j < sz(pats[i]); j++)
f95b70        {
ec0388          int k = pats[i][j] - first;
d41d8c
10937b          if (nodes[cur].next[k] <= 0)  // Make new node if needed.
f95b70          {
976fa3            nodes[cur].next[k] = sz(nodes);
225eb3            nodes.emplace_back();
cbb184          }
d41d8c
47b49f          cur = nodes[cur].next[k];
cbb184        }
d41d8c
d41d8c        // Add logic here if additional data is needed on matched
   strings.
4daeea        nodes[cur].nmatches++;
45f177        nodes[cur].match_list = new list_node(i, nodes[cur].match_list)
   ;  // Add string to node list of matches.
fe38fe        if (nodes[cur].nmatches == 1)
947da5          nodes[cur].match_list_last = nodes[cur].match_list;
cbb184      }
d41d8c
26a528      queue<int> q;
6733a6      for (int i = 0; i < alpha_size; i++) // Define fail for first
   level.
f95b70      {
e8dc83        if (nodes[0].next[i] == -1) // Invalid transitions from 0 now
   become valid self transitions.
fb628f          nodes[0].next[i] = 0;
d41d8c
7d3171        nodes[0].atm[i] = nodes[0].next[i]; // Automaton state
   transition table.
d41d8c
bc34bf        if (nodes[0].next[i] > 0) // Single letter nodes have fail = 0
   and go in the queue.
f95b70        {
eded92          q.push(nodes[0].next[i]);
9b22e6          nodes[nodes[0].next[i]].fail = 0;
cbb184        }
cbb184      }
d41d8c
ee6bdd      while (!q.empty()) // Use bfs to compute fail for next level.
f95b70      {
69faa7        int cur = q.front();
833270        q.pop();
d41d8c
6733a6        for (int i = 0; i < alpha_size; i++)
```

```
af4a6e          if (nodes[cur].next[i] > 0) // Don't use -1 and don't use
   transition to root.
f95b70          {
3ecdd3            nodes[cur].atm[i] = nodes[cur].next[i]; // Unrelated to
   code below, filling automaton.
d41d8c
d41d8c            // Computing fail for next node and putting it in the queue
   .
3ae7da            int prox = nodes[cur].next[i];
53ef92            q.push(prox);
d41d8c
f252cb            int state = nodes[cur].fail;
c66324            while (nodes[state].next[i] == -1)
d712e2              state = nodes[state].fail;
d41d8c
7836db            nodes[prox].fail = nodes[state].next[i];
d41d8c
d41d8c            // Add logic here if additional data is needed on matched
   strings.
2940ed            nodes[prox].nmatches += nodes[nodes[prox].fail].nmatches;
d41d8c
d41d8c            // Add in O(1) list from fail link to next node's list.
   Operation: a->b->null c->null to a->b->c->null.
59ed4d            (nodes[prox].match_list_last ? nodes[prox].match_list_last
   ->next : nodes[prox].match_list) = nodes[nodes[prox].fail].match_list;
cbb184          }
2954e9          else
f95b70          {
a04598            nodes[cur].atm[i] = nodes[nodes[cur].fail].atm[i];
cbb184          }
cbb184      }
cbb184    }
d41d8c
d41d8c    // Optional
d41d8c    // Returns a vector retv such that, for each text position i:
d41d8c    // retv[i] is the index of the largest pattern ending at position i
    in the text.
d41d8c    // If retv[i] == -1, no pattern ends at position i.
32246d    vector<int> find(const string &text)
f95b70    {
107323      vector<int> retv(sz(text));
b3da3c      int cur = 0;
d41d8c
77447e      for (int i = 0; i < sz(text); i++)
f95b70      {
13dae2        cur = nodes[cur].atm[text[i] - first];
29e58f        retv[i] = (nodes[cur].match_list ? nodes[cur].match_list->id :
   -1);
cbb184      }
d41d8c
```

```
6272cf      return retv;
cbb184    }
d41d8c
d41d8c    // Optional
d41d8c    // Returns a vector retv such that, for each text position i:
d41d8c    // retv[i] is the number of pattern matches ending at position i in
    the text.
48d0f2    vector<int> count(const string &text)
f95b70    {
107323      vector<int> retv(sz(text));
b3da3c      int cur = 0;
d41d8c
77447e      for (int i = 0; i < sz(text); i++)
f95b70      {
13dae2        cur = nodes[cur].atm[text[i] - first];
1a43d3        retv[i] = nodes[cur].nmatches;
cbb184      }
d41d8c
6272cf      return retv;
cbb184    }
d41d8c
d41d8c    // Optional
d41d8c    // Returns a vector retv such that, for each text position i:
d41d8c    // retv[i] is a list of indexes to the patterns ending at position
   i in the text.
d41d8c    // These lists will be sorted from largest to smallest pattern
   length.
d41d8c    // Keep in mind that find_all can be O(N*sqrt(N)) if no duplicate
   patterns. (N is total string length)
4e5a4c    vector<vector<int>> find_all(const string &text)
f95b70    {
77b54a      vector<vector<int>> retv(sz(text));
b3da3c      int cur = 0;
d41d8c
77447e      for (int i = 0; i < sz(text); i++)
f95b70      {
13dae2        cur = nodes[cur].atm[text[i] - first];
d82b0e        for (auto n = nodes[cur].match_list; n != nullptr; n = n->next)
4c4784          retv[i].push_back(n->id);
cbb184      }
d41d8c
6272cf      return retv;
cbb184    }
d41d8c
d41d8c    // Optional
d41d8c    // Returns a vector retv such that:
d41d8c    // retv is a list of indexes to the patterns ending at position pos
    in the text.
d41d8c    // This list will be sorted from largest to smallest pattern length
   .
```

```
251c66    vector<int> find_all_at_pos(const string &text, int pos)
f95b70    {
aeb888      vector<int> retv;
b3da3c      int cur = 0;
d41d8c
77447e      for (int i = 0; i < sz(text); i++)
f95b70      {
13dae2        cur = nodes[cur].atm[text[i] - first];
d41d8c
c57c6f        if (i == pos)
d82b0e          for (auto n = nodes[cur].match_list; n != nullptr; n = n->
   next)
1ad617            retv.push_back(n->id);
cbb184      }
d41d8c
6272cf      return retv;
cbb184    }
2ec64b  };
```

8.3 Suffix Array

```
d41d8c
5d1131  #include "../../contest/header.hpp"
d41d8c
d41d8c  /*
1f77e9    Suffix array:
be00ca      Build suffix array and LCP array in O((n + lim) log n) using O(n
   + lim) memory, where lim is the alphabet size.
d41d8c
643e15      sa[i] is the starting index of the suffix which is i-th in the
   sorted suffix array.
0d5e62      The returned vector is of size s.size()+1, and sa[0] == s.size().
    The '\0' char at the end is considered
29ea73      part of the string, so sa[0] = "\0", the prefix starting at index
    s.size().
d41d8c
ee7035      The lcp array contains longest common prefixes for neighbouring
   strings
317e94      in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0.
d41d8c
81eeab    Example:
981b73      Computing the LCP and the SA of "GATAGACA"
d33e7b        i sa[i] lcp[i]  suffix
fd774f        0 8   0   ""
cac682        1 7   0   "A"
430b3a        2 5   1   "ACA"
d30cc0        3 3   1   "AGACA"
c895f5        4 1   1   "ATAGACA"
1a04b3        5 6   0   "CA"
```

```
b1b780        6 4   0   "GACA"
2999cd        7 0   2   "GATAGACA"
08e6dc        8 2   0   "TAGACA"
d41d8c
b95cae    Usage:
1c63e0      Important: the input string must not contain any zero values.
   Must use C++11 or above.
b847d4      You can use this for strings of integers, just change the
   alphabet size.
d41d8c
1d1558    Source: https://github.com/kth-competitive-programming/kactl/blob/
   master/content/strings/SuffixTree.h
c4c9bd  */
d41d8c
15a9b6  struct suffix_array
f95b70  {
71675a    vector<int> sa, lcp;
092958    suffix_array(const string &s, int lim = 256) // or basic_string<int
   > for integer strings.
f95b70    {
e72340      int n = sz(s) + 1, k = 0, a, b;
f6a0db      vector<int> x(s.begin(), s.end() + 1), y(n), ws(max(n, lim)),
   rank(n);
85469f      sa = lcp = y;
eb75f9      iota(sa.begin(), sa.end(), 0);
7707f7      for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p)
f95b70      {
8dff9b        p = j;
00aec0        iota(y.begin(), y.end(), n - j);
83008c        for (int i = 0; i < n; i++)
e9b19c          if (sa[i] >= j)
d0873d            y[p++] = sa[i] - j;
450a8a        fill(ws.begin(), ws.end(), 0);
83008c        for (int i = 0; i < n; i++)
799bb0          ws[x[i]]++;
7d6bd3        for (int i = 1; i < lim; i++)
f256af          ws[i] += ws[i - 1];
5df399        for (int i = n; i--;)
d01b67          sa[--ws[x[y[i]]]] = y[i];
9dd20c        swap(x, y);
017be6        p = 1;
16ab1b        x[sa[0]] = 0;
d41d8c
aa4866        for (int i = 1; i < n; i++)
f95b70        {
fcb940          a = sa[i - 1];
2d820b          b = sa[i];
0cc036          x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
cbb184        }
cbb184      }
```

```
d41d8c
aa4866      for (int i = 1; i < n; i++)
2f33c5        rank[sa[i]] = i;
d41d8c
05cb2b      for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
487069        for (k &&k--, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; k++)
9eecb7          ;
cbb184    }
2145c1  };
87092f
```