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 unsigned long long ull;
typedef long double ld;
typedef vector<int> vi;

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

#define sz(x) ((int)(x).size())
#define all(x) x.begin(), x.end()

// 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);
// }
```

1.2 Sample Debug

```
32c #include "header.hpp"
d41
13a int main(void)
f95 {
3e8   int a = 11, b = 12, c = 13;
b9e   vector<vector<int>> v = {{a, b}, {c}, {0, 1}};
2a8   set<int> s = {a, b};
6b3   map<double, int> m;
af2   m[2.5] = 2;
37a   m[-3.1] = 3;
d41
632   map<string, int> tab;
88e   tab["abc"] = (int) 'a' + 'b' + 'c';
699   tab["abz"] = (int) 'a' + 'b' + 'z';
bd6   int array[3] = {1, 2, 5};
d41
593   debug(a, b, c);
fb9   debug(v);
b45   debug(s, m);
3cf   debug(tab);
d95   debug(array); // This one does not work.
cbb }
Full file hash: 50cc4b
```

1.3 Hash Code (Line)

```
#!/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-3 | tr -d '[:space:]';
  echo "  $line"; # Before $line is a tab.
done
```

1.4 Hash Code (File)

```
#!/bin/bash
# Hashes a file, ignoring lines with #include, all whitespace
     and comments

sed -e "/#include/d" | cpp -dD -P -fpreprocessed | tr -d '[:
     space:]' | md5sum | cut -c-6
```

1.5 Diff Script

```
#!/bin/bash

set -ex

for ((a=1; ; a++))
do
  ./gen.out $a > in.txt
  ./width.out < in.txt > out1
  ./width_brute.out < in.txt > out2
  diff out1 out2
done
```

1.6 Submit Script

```
#!/bin/bash

if [ "$1" != "$2.cpp" ];
then
  echo "mismatch"
  exit 1
fi
```

2 Data Structures

2.1 2D Segment Tree

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
fc6   2D Segment Tree:
924     2D point update and 2D range query in O(log(n)*log(m))
     per
035     operation,
da2     where n is the number of rows and m is the number of
     columns.
d41
65b     Uses O(Num_Updates * log(n)*log(m)) memory.
d41
765     Given as an example using gcd.
d41
b95   Usage:
5b4     See main and comments.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
d41 // Number of rows and columns (inner nodes need to know
     this).
14e int n, m;
d41
fc0 struct nodes
f95 {
348   ll val;
af0   nodes *left, *right;
2fd   nodes() : val(0), left(NULL), right(NULL) {}
d41
d41   // Update leaf tree.
7ac   void update(int l, int r, int a, ll x)
f95   {
bd3     if (l == r)
c43       val = x;
295     else
f95     {
ae0       int mid = (l + r) / 2;
a49       if (a <= mid)
ff3         (left ? left : (left = new nodes()))->update(l, mid
, a, x);
295       else
ee9         (right ? right : (right = new nodes()))->update(mid
 + 1, r, a, x);
d41
6a8       val = __gcd(left ? left->val : 0, right ? right->val
: 0);
cbb     }
cbb   }
d41
d41   // Update non-leaf tree by joining the two child trees
     along the
d41   // modified path.
fe4   void updateb(int l, int r, int a, ll x, nodes *o, nodes *
p)
f95   {
bd3     if (l == r)
8eb       val = __gcd(o ? o->val : 0, p ? p->val : 0);
295     else
f95     {
ae0       int mid = (l + r) / 2;
a49       if (a <= mid)
0db         (left ? left : (left = new nodes()))->updateb(l,
     mid, a, x, o ? o->left : NULL, p ? p->left : NULL);
295       else
58b         (right ? right : (right = new nodes()))->updateb(
     mid + 1, r, a, x, o ? o->right : NULL, p ? p->right : NULL
     );
d41
8eb       val = __gcd(o ? o->val : 0, p ? p->val : 0);
cbb     }
cbb   }
d41
158   ll get(int l, int r, int a, int b)
f95   {
472     if (l == a && r == b)
d94       return val;
295     else
f95     {
ae0       int mid = (l + r) / 2;
f89       if (b <= mid)
ac5         return left ? left->get(l, mid, a, b) : 0;
a54       else if (a > mid)
1c7         return right ? right->get(mid + 1, r, a, b) : 0;
295       else
61c         return __gcd(left ? left->get(l, mid, a, mid) : 0,
     right ? right->get(mid + 1, r, mid + 1, b) : 0);
cbb     }
cbb   }
214 };
d41
7fe struct nodef
f95 {
848   nodes *val;
16e   nodef *left, *right;
da6   nodef() : left(NULL), right(NULL) { val = new nodes(); }
d41
0ab   void update(int l, int r, int a, int b, ll x)
f95   {
```

```
bd3     if (l == r)
b7e       val->update(0, m - 1, b, x);
295     else
f95     {
ae0       int mid = (l + r) / 2;
a49       if (a <= mid)
e5a         (left ? left : (left = new nodef()))->update(l, mid
     , a, b, x);
295       else
25f         (right ? right : (right = new nodef()))->update(mid
     + 1, r, a, b, x);
d41
c63       val->updateb(0, m - 1, b, x, left ? left->val : NULL,
     right ? right->val : NULL);
cbb     }
cbb   }
d41
0bc   ll get(int l, int r, int a, int b, int c, int d)
f95   {
472     if (l == a && r == b)
2b5       return val->get(0, m - 1, c, d);
295     else
f95     {
ae0       int mid = (l + r) / 2;
f89       if (b <= mid)
466         return left ? left->get(l, mid, a, b, c, d) : 0;
a54       else if (a > mid)
d6a         return right ? right->get(mid + 1, r, a, b, c, d) :
     0;
295       else
90c         return __gcd(left ? left->get(l, mid, a, mid, c, d)
     : 0, right ? right->get(mid + 1, r, mid + 1, b, c, d) :
     0);
cbb     }
cbb   }
214 };
d41
13a int main(void)
f95 {
8de   ll a;
66f   nodef *root = new nodef();
0dc   int q, tp, x, y, z, w;
cdc   scanf("%d %d %d", &n, &m, &q);
a95   while (q--)
f95   {
64d     scanf("%d", &tp);
abc     if (tp == 1)
f95     {
425       scanf("%d %d %lld", &x, &y, &a);
dbf       root->update(0, n - 1, x, y, a);
cbb     }
295     else
f95     {
bb2       scanf("%d %d %d %d", &x, &y, &z, &w);
c00       printf("%lld\n", root->get(0, n - 1, x, z, y, w));
cbb     }
cbb   }
cbb }
Full file hash: 1f17b0
```

2.2 BIT

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
0e8   BIT: element update, range sum query and sum lower_bound
     in
4a7   O(log(N)).
271   Represents an array of elements in range [1, N].
c4c */
d41
4fc template <class T>
2d5 struct bit
f95 {
b9a   int n, LOGN;
226   vector<T> val;
695   bit(int _n) : n(_n), LOGN(log2(n + 1)), val(_n + 1, 0) {}
d41
d41   // val[pos] += x
b29   void update(int pos, T x)
f95   {
259     for (int i = pos; i <= n; i += -i & i)
ac5       val[i] += x;
cbb   }
d41
d41   // sum of range [1, pos]
8a8   T query(int pos)
f95   {
566     T retv = 0;
ac4     for (int i = pos; i > 0; i -= -i & i)
106       retv += val[i];
627     return retv;
cbb   }
d41
d41   // min pos such that sum of [1, pos] >= sum, or n + 1 if
     none
d41   // exists.
79d   int lower_bound(T x)
f95   {
501     T sum = 0;
bec     int pos = 0;
d41
51d     for (int i = LOGN; i >= 0; i--)
032       if (pos + (1 << i) <= n && sum + val[pos + (1 << i)]
     < x)
420         sum += val[pos += (1 << i)];
d41
7e2     return pos + 1; // pos will have position of largest
     value
d41             // less than x.
cbb   }
214 };
Full file hash: 6acfcc
```

2.3 BIT2D

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
21f   BIT: element update, range sum query in O(log(n) * log(m)
     ).
849   This can also be generalized for 3d.
a6c   Represents a matrix of elements in range [1 ... n][1 ...
     m].
c4c */
d41
4fc template <class T>
f6f struct bit2d
f95 {
14e   int n, m;
f7e   vector<vector<T>> val;
9c8   bit2d(int _n, int _m) : n(_n), m(_m), val(_n + 1, vector<
     T>(_m + 1, 0)) {}
d41
d41   // val[i][j] += x
446   void update(int r, int c, T x)
f95   {
9e4     for (int i = r; i <= n; i += -i & i)
13d       for (int j = c; j <= m; j += -j & j)
ff2         val[i][j] += x;
cbb   }
d41
d41   // sum of positions (1 ... r, 1 ... c)
450   T query(int r, int c)
f95   {
566     T retv = 0;
bc7     for (int i = r; i > 0; i -= -i & i)
d53       for (int j = c; j > 0; j -= -j & j)
86d         retv += val[i][j];
627     return retv;
cbb   }
d41
d41   // sum of positions (ri ... rf, ci ... cf). (1 <= ri <=
     rf <= n)
d41   // and (1 <= ci <= cf <= m). TODO: test me.
bdc   T query_rect(int ri, int ci, int rf, int cf)
f95   {
607     return query(rf, cf) - query(rf, ci - 1) - query(ri -
     1, cf) + query(ri - 1, ci - 1);
cbb   }
214 };
Full file hash: a4a33c
```

2.4 Dynamic Segment Tree

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
699   Segment tree with dynamic memory allocation and arbitrary
468   interval.
91e     Every operation is O(log(r-l))
644     Uses O(min(r-l, n*log(r-l)) memory, where n is the
     number of
1ce     insertions.
d41
ca2   Constraints:
3dc     Segment tree range [l, r] must be such that 0 <= l <= r
     .
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
4fc template<class T>
e4a struct node
f95 {
f48   T val;
af3   node *left, *right;
d41
995   T get(int l, int r, int a, int b)
f95   {
472     if (l == a && r == b)
d94       return val;
814     int mid = (l + 0ll + r) / 2;
f89     if (b <= mid)
ac5       return left ? left->get(l, mid, a, b) : 0;
a54     else if (a > mid)
1c7       return right ? right->get(mid + 1, r, a, b) : 0;
295     else
9b1       return (left ? left->get(l, mid, a, mid) : 0) + (
     right ? right->get(mid + 1, r, mid + 1, b) : 0);
cbb   }
d41
14d   void update(int l, int r, int a, T x)
f95   {
bd3     if (l == r)
c43       val = x;
295     else
f95     {
814       int mid = (l + 0ll + r) / 2;
a49       if (a <= mid)
```

```
1ec         (left ? left : (left = new node()))->update(l, mid,
      a, x);
295       else
92f         (right ? right : (right = new node()))->update(mid
     + 1, r, a, x);
d41
dd5       val = (left ? left->val : 0) + (right ? right->val :
     0);
cbb     }
cbb   }
214 };
d41
Full file hash: eef0d8
```

2.5 Linear Container

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
20b   Line Container (most common for convex hull trick).
     Amortized
2ff   O(log N) per operation.
d7b     Container where you can add lines of the form kx+m, and
      query
2d6     maximum values at points x.
dc4     Useful for dynamic programming.
d41
1d1   Source: https://github.com/kth-competitive-programming/
     kactl/
c12       blob/master/content/contest/template.cpp
c4c */
d41
3fe struct line
f95 {
3e2   mutable ll k, m, p;
889   bool operator<(const line &o) const { return k < o.k; }
abf   bool operator<(ll x) const { return p < x; }
214 };
d41
0c8 struct line_container : multiset<line, less<>>
f95 {
d41   // (for doubles, use inf = 1/.0, div(a,b) = a/b)
f5e   const ll inf = LLONG_MAX;
d41
960   ll div(ll a, ll b)
f95   { // floored division
353     return a / b - ((a ^ b) < 0 && a % b);
cbb   }
d41
9c0   bool isect(iterator x, iterator y)
f95   {
f95     if (y == end())
f95     {
09a       x->p = inf;
d1f       return false;
cbb     }
3cc     if (x->k == y->k)
83e       x->p = x->m > y->m ? inf : -inf;
295     else
b42       x->p = div(y->m - x->m, x->k - y->k);
870     return x->p >= y->p;
cbb   }
d41
928   void add(ll k, ll m)
f95   {
116     auto z = insert({k, m, 0}), y = z++, x = y;
2d9     while (isect(y, z))
96c       z = erase(z);
d94     if (x != begin() && isect(--x, y))
c07       isect(x, y = erase(y));
57d     while ((y = x) != begin() && (--x)->p >= y->p)
774       isect(x, erase(y));
cbb   }
d41
e8b   ll query(ll x)
f95   {
229     assert(!empty());
7d1     auto l = *lower_bound(x);
96a     return l.k * x + l.m;
cbb   }
214 };
d41
Full file hash: 66b35a
```

2.6 Merge Sort Tree

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
abc   Merge Sort Tree:
419     Build segment tree where each node stores a sorted
     version
ea7     of the underlying range.
dc8     O(n log n) to build, O(log^2 n) each query (in this
     case).
2a5     This example uses kth number in interval queries.
d41
b95   Usage:
67c     In this example, instead of building the merge sort
     tree with
656     the given vector (e.g. {1, 5, 2, 6, 3, 7, 4}), we sort
     a
d2e     vector of indices by their value in the vector
6bf     (e.g. {0, 2, 4, 6, 1, 3, 5} for the vector above).
d41
5cd     This way, each node in the tree is responsible for a
     range of
1ff     sorted elements in the vector and we can ask it how
     many of
d55     those have indices in the range [a, b].
c4c */
d41
4ee #define left(i) ((i) << 1)
56e #define right(i) (((i) << 1) + 1)
d41
05a struct merge_sort_tree
f95 {
990   vector<int> v; // original vector.
dc0   vector<vector<int>> val;
1be   vector<int> indices; // indices sorted by value in v.
d41
1b6   merge_sort_tree(vector<int> v) : v(v), val(4 * (sz(v) +
     1))
f95   {
9ad     for (int i = 0; i < sz(v); i++)
f2b       indices.push_back(i);
788     sort(all(indices), [&v](int i, int j) { return v[i] < v
     [j]; });
d41
5ba     build(1, 0, sz(v) - 1);
cbb   }
d41
b73   void build(int id, int l, int r)
f95   {
bd3     if (l == r)
1b1       val[id].push_back(indices[l]);
295     else
f95     {
ae0       int mid = (l + r) / 2;
c7a       build(left(id), l, mid), build(right(id), mid + 1, r)
     ;
0d7       val[id] = vector<int>(r - l + 1);
2c5       merge(all(val[left(id)]), all(val[right(id)]), val[id
     ].begin());
cbb     }
cbb   }
d41
d41   // How many elements in this node have indices in the
     range [a, b]
a5c   int count_interval(int id, int a, int b)
f95   {
a4c     return (int)(upper_bound(all(val[id]), b) - lower_bound
     (all(val[id]), a));
cbb   }
d41
bea   int get(int id, int l, int r, int a, int b, int x)
f95   {
bd3     if (l == r)
bc4       return v[val[id].back()];
ae0     int mid = (l + r) / 2;
7c1     int lcount = count_interval(left(id), a, b);
87e     if (lcount >= x)
7a3       return get(left(id), l, mid, a, b, x);
295     else
f29       return get(right(id), mid + 1, r, a, b, x - lcount);
cbb   }
d41
5c9   int kth(int a, int b, int k)
f95   {
492     return get(1, 0, sz(v) - 1, a, b, k);
cbb   }
214 };
Full file hash: 284e0c
```

2.7 Min Queue

```
5d1 #include "../../contest/header.hpp"
d41 /*
958   max(min) queue with O(1) get_max(min).
d41
f67   Tips:
c53     - Useful for sliding window 1D and 2D.
e41     - For 2D problems, you will need to pre-compute another
c71     matrix, by making a row-wise traversal, and calculating
      the
c0a     min/max value beginning in each cell. Then you just
     make a
dff     column-wise traverse as they were each an independent
     array.
c4c */
d41
8f0 struct max_queue
f95 {
848   queue<ll> q;
889   deque<ll> s;
d41
dbb   int size()
f95   {
593     return (int)q.size();
cbb   }
d41
a1f   void push(ll val)
f95   {
d41     // while (!s.empty() && s.back() > val) -> for a
     min_queue
1cb     while (!s.empty() && s.back() < val) // for a max_queue
342       s.pop_back();
fcc     s.push_back(val);
d41
```

```
380     q.push(val);
cbb   }
d41
d99   void pop()
f95   {
7a8     ll u = q.front();
833     q.pop();
d41
de7     if (!s.empty() && s.front() == u)
784       s.pop_front();
cbb   }
d41
ba2   ll get_max()
f95   {
ecc     return s.front(); // same for min and max queue
cbb   }
d41
214 };
Full file hash: 82549d
```

2.8 Persistent Segment Tree

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
a03   Persistent Segment Tree:
115     Segment tree that stores all previous versions of itself.
91e     Every operation is O(log(r-l))
ad6     Uses O(n*log(r-l)) memory, where n is the number of updates.
d41
b95   Usage:
4e3     A new root is created for every persistent update (p_update)
db6     and returned.
92a     Queries can be performed on any root as if it were a usual
3b2     segment tree.
61a     You should keep a list of roots. Something like:
072       vector<node *> roots = {new node()};
e02       roots.push_back(p_update(roots.back(), 0,
d75                     2*MAXV, a[i] + MAXV, v + 1));
d41
ca2   Constraints:
3dc     Segment tree range [l, r] must be such that 0 <= l <= r.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
e4a struct node
f95 {
97f   int val;
af3   node *left, *right;
d41
1f6   node(int x=0) : val(x), left(NULL), right(NULL) {}
2f7   node(node *l, node *r) : left(l), right(r) { val = (left ? left->val : 0) + (right ? right->val : 0); }
d41
f21   int get(int l, int r, int a, int b)
f95   {
472     if (l == a && r == b)
d94       return val;
814     int mid = (l + 0ll + r) / 2;
f89     if (b <= mid)
ac5       return left ? left->get(l, mid, a, b) : 0;
a54     else if (a > mid)
1c7       return right ? right->get(mid + 1, r, a, b) : 0;
295     else
9b1       return (left ? left->get(l, mid, a, mid) : 0) + (right ? right->get(mid + 1, r, mid + 1, b) : 0);
cbb   }
214 };
d41
63f node *p_update(node *prev, int l, int r, int a, int x)
f95 {
bd3   if (l == r)
134     return new node(x);
d41
814   int mid = (l + 0ll + r) / 2;
a49   if (a <= mid)
b73     return new node(p_update(prev ? prev->left : NULL, l, mid, a, x), prev ? prev->right : NULL);
295   else
460     return new node(prev ? prev->left : NULL, p_update(prev ? prev->right : NULL, mid + 1, r, a, x));
cbb }
d41
Full file hash: 707f69
```

2.9 Segment Tree

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
51c   Segment Tree:
28d     Point update and range query in O(log(n))
251     Given as an example using maximum.
d41
b95   Usage:
4d3     Only valid if all numbers are >= 0.
c4c */
d41
4ee #define left(i) ((i) << 1)
56e #define right(i) (((i) << 1) + 1)
d41
d35 struct segtree
f95 {
8c4   vector<int> val;
1a8   int n;
d41
ea6   segtree(int n) : val(4 * (n + 1), 0), n(n) {}
d41
aa6   void update(int id, int l, int r, int a, int x)
f95   {
bd3     if (l == r)
916       val[id] = x;
295     else
f95     {
ae0       int mid = (l + r) / 2;
a49       if (a <= mid)
4c0         update(left(id), l, mid, a, x);
295       else
814         update(right(id), mid + 1, r, a, x);
d41
56a       val[id] = max(val[left(id)], val[right(id)]);
cbb     }
cbb   }
d41
9fe   int get(int id, int l, int r, int a, int b)
f95   {
472     if (l == a && r == b)
a03       return val[id];
295     else
f95     {
ae0       int mid = (l + r) / 2;
f89       if (b <= mid)
c55         return get(left(id), l, mid, a, b);
a54       else if (a > mid)
26d         return get(right(id), mid + 1, r, a, b);
295       else
84b         return max(get(left(id), l, mid, a, mid), get(right(id), mid + 1, r, mid + 1, b));
cbb     }
cbb   }
d41
0fb   int get(int a, int b)
f95   {
f78     if (a > b)
bb3       return 0;
ec7     return get(1, 0, n - 1, a, b);
cbb   }
d41
44c   void update(int a, int x)
f95   {
c44     update(1, 0, n - 1, a, x);
cbb   }
214 };
Full file hash: 547480
```

2.10 Lazy Segment Tree

```
2b7 #include <bits/stdc++.h>
ca4 using namespace std;
d41
d41 /*
6f5   Segment Tree with Lazy updates:
d8b     Range update and range query in O(log(MAX_RANGE))
c32     Binary search on tree in O(log(MAX_RANGE))
3d7     Given as an example since it is not worth it to copy a
c17     generic tree during a contest.
d41
e3c   Solves: https://codeforces.com/contest/1179/problem/C
c4c */
d41
ab0 #define MAX_RANGE 1123456
d41
fd8 int val[4 * MAX_RANGE];
802 int delta[4 * MAX_RANGE];
d41
4ee #define left(i) ((i) << 1)
56e #define right(i) (((i) << 1) + 1)
d41
037 void prop(int id, int l, int r)
f95 {
cfd   if (l != r)
f95   {
d41     // Updates need to be numerically stackable (e.g. not valid
d41     // to have a list of updates).
df5     delta[left(id)] += delta[id];
966     delta[right(id)] += delta[id];
cbb   }
d41
21c   val[id] += delta[id]; // Node value needs to be obtainable without
d41             // propagating all the way to root.
0a8   delta[id] = 0;
cbb }
d41
d41 // Sum x in all elements in range [a, b].
f2b void update(int id, int l, int r, int a, int b, int x)
f95 {
add   if (a == l && b == r)
f95   {
d50     delta[id] += x;
b62     prop(id, l, r);
cbb   }
295   else
```

```
f95   {
b62     prop(id, l, r);
ae0     int mid = (l + r) / 2;
f89     if (b <= mid)
f95     {
6db       update(left(id), l, mid, a, b, x);
384       prop(right(id), mid + 1, r);
cbb     }
a54     else if (a > mid)
f95     {
859       update(right(id), mid + 1, r, a, b, x);
221       prop(left(id), l, mid);
cbb     }
295     else
f95     {
fc7       update(left(id), l, mid, a, mid, x);
04c       update(right(id), mid + 1, r, mid + 1, b, x);
cbb     }
d41
caf     val[id] = min(val[left(id)], val[right(id)]);
cbb   }
cbb }
d41
d41 // Get the minimum value in range [a, b].
9fe int get(int id, int l, int r, int a, int b)
f95 {
b62   prop(id, l, r);
add   if (a == l && b == r)
a03     return val[id];
295   else
f95   {
ae0     int mid = (l + r) / 2;
f89     if (b <= mid)
c55       return get(left(id), l, mid, a, b);
a54     else if (a > mid)
26d       return get(right(id), mid + 1, r, a, b);
295     else
5e3       return min(get(left(id), l, mid, a, mid), get(right(
    id), mid + 1, r, mid + 1, b));
cbb   }
cbb }
d41
d41 // Find index of rightmost element which is less than x. (
    works
d41 // because this is a seg of min)
052 int bsearch(int id, int l, int r, int x)
f95 {
b62   prop(id, l, r);
d41
bd3   if (l == r)
f7d     return (val[id] < x) ? l : -1;
295   else
f95   {
ae0     int mid = (l + r) / 2;
221     prop(left(id), l, mid);
384     prop(right(id), mid + 1, r);
f01     if (val[right(id)] < x)
018       return bsearch(right(id), mid + 1, r, x);
295     else
bad       return bsearch(left(id), l, mid, x);
cbb   }
cbb }
d41
103 #define MAXN 312345
d41
a58 int a[MAXN];
c4b int b[MAXN];
d41
13a int main(void)
f95 {
b06   int n, m, q, tp, x, y;
d69   scanf("%d %d", &n, &m);
535   for (int i = 1; i <= n; i++)
f95   {
937     scanf("%d", &a[i]);
49e     update(1, 1, 1000000, 1, a[i], -1);
cbb   }
d41
8ea   for (int i = 1; i <= m; i++)
f95   {
264     scanf("%d", &b[i]);
472     update(1, 1, 1000000, 1, b[i], 1);
cbb   }
d41
4aa   scanf("%d", &q);
a95   while (q--)
f95   {
960     scanf("%d %d %d", &tp, &x, &y);
abc     if (tp == 1)
f95     {
996       update(1, 1, 1000000, 1, a[x], 1);
e60       a[x] = y;
28c       update(1, 1, 1000000, 1, a[x], -1);
cbb     }
295     else
f95     {
8db       update(1, 1, 1000000, 1, b[x], -1);
046       b[x] = y;
bc1       update(1, 1, 1000000, 1, b[x], 1);
cbb     }
d41
584     int tmp = bsearch(1, 1, 1000000, 0);
d41
d41     // Test of get and bsearch. Make sure all to the right
    are
d41     // non-negative.
5a5     if (tmp != 1000000)
5df       assert(get(1, 1, 1000000, tmp == -1 ? 1 : (tmp + 1),
    1000000) >= 0);
c3e     if (tmp != -1)
1d9       assert(get(1, 1, 1000000, tmp, tmp) < 0);
d41
b03     printf("%d\n", tmp);
cbb   }
cbb }
Full file hash: 90a905
```

2.11 Key Treap

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
197   Treap:
0d1     This treap implements something like a c++ set with
    additional
b36     operations: find the k-th element and count elements
    less than
2fe     a given value.
d41
4c8   Time: O(log N) per operation.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
41c namespace treap
f95 {
e4a struct node
f95 {
97f   int val; // node key.
ee1   int p;   // node heap priority.
59a   int num; // node subtree size.
af3   node *left, *right;
d41
710   node(int _val) : val(_val), p(rand()), num(1), left(NULL)
    , right(NULL) {}
214 };
d41
48f int get_num(node *root)
f95 {
424   return (root == NULL) ? 0 : root->num;
cbb }
d41
68f void update_num(node *root)
f95 {
47a   root->num = get_num(root->left) + get_num(root->right) +
    1;
cbb }
d41
afd node *rotate_left(node *root)
f95 {
d25   node *a = root;
a95   node *b = root->right;
d41
b51   a->right = b->left;
e7e   b->left = a;
a5e   update_num(a);
2b1   update_num(b);
73f   return b;
cbb }
d41
f17 node *rotate_right(node *root)
f95 {
d25   node *a = root;
eb0   node *b = root->left;
d41
a09   a->left = b->right;
735   b->right = a;
a5e   update_num(a);
2b1   update_num(b);
73f   return b;
cbb }
d41
d41 // Insert new node with key x in treap rooted at root if
    not already
d41 // there.
960 node *insert(node *root, int x)
f95 {
0ed   if (root == NULL)
134     return new node(x);
6b2   if (x > root->val)
34c     root->right = insert(root->right, x);
ba0   else if (x < root->val)
12f     root->left = insert(root->left, x);
d41
622   update_num(root);
d41
4f4   if (root->right && root->right->p > root->p)
041     root = rotate_left(root);
c93   if (root->left && root->left->p > root->p)
3f3     root = rotate_right(root);
e2f   return root;
cbb }
d41
d41 // Remove node with key x in treap rooted at root if
    present.
d0b node *remove(node *root, int x)
f95 {
0ed   if (root == NULL)
ea9     return NULL;
6b2   if (x > root->val)
```

```
fed     root->right = remove(root->right, x);
ba0   else if (x < root->val)
6cf     root->left = remove(root->left, x);
fb8   else if (root->left == NULL)
4de     root = root->right;
a15   else if (root->right == NULL)
2d4     root = root->left;
386   else if (root->left->p > root->right->p)
f95   {
3f3     root = rotate_right(root);
fed     root->right = remove(root->right, x);
cbb   }
295   else
f95   {
041     root = rotate_left(root);
6cf     root->left = remove(root->left, x);
cbb   }
e6a   if (root)
622     update_num(root);
e2f   return root;
cbb }
d41
d41 // Return the k-th smallest element in tree rooted at root.
357 int kth(node *root, int k)
f95 {
f9e   if (get_num(root->left) >= k)
747     return kth(root->left, k);
f3e   else if (get_num(root->left) + 1 == k)
ae0     return root->val;
295   else
235     return kth(root->right, k - get_num(root->left) - 1);
cbb }
d41
d41 // Return the number of elements smaller than x in tree
     rooted at root
194 int count(node *root, int x)
f95 {
0ed   if (root == NULL)
bb3     return 0;
830   if (x < root->val)
da7     return count(root->left, x);
08e   else if (x == root->val)
140     return get_num(root->left);
295   else
b73     return get_num(root->left) + 1 + count(root->right, x);
cbb }
cbb } // namespace treap
Full file hash: 85f362
```

2.12 Sequential Treap

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
197   Treap:
763     A short self-balancing tree. It acts as a sequential
     container
88c     with log-time splits/joins, and is easy to augment with
2d7     additional data.
d41
4c8   Time: O(log N) per operation.
d41
ca2   Constraints:
c1b     Acts as a vector of size N, with positions in range [0,
     N-1].
d41
1d1   Source: https://github.com/kth-competitive-programming/
     kactl/blob/
e1d       master/content/data-structures/Treap.h
d41
b95   Usage:
bfe     To insert elements, create one node treaps.
396     (e.g. treap::ins(root, new treap::node(x), i))
2ab     To augment with extra data you should mostly add stuff
     to the
24b     recalc function. (e.g. to make it work like a seg tree)
03b     See applications for more usage examples.
c4c */
d41
41c namespace treap
f95 {
e4a struct node
f95 {
8f5   node *l = 0, *r = 0;
97f   int val;   // Any value associated with node.
ee1   int p;   // Node heap priority.
c6a   int c = 1; // Node subtree size.
674   node(int val) : val(val), p(rand()) {}
86d   void recalc();
214 };
d41
853 int cnt(node *n) { return n ? n->c : 0; }
9af void node::recalc() { c = cnt(l) + cnt(r) + 1; }
d41
d41 // Apply function f on each tree node in order.
044 template <class F>
d54 void each(node *n, F f)
f95 {
f63   if (n)
f95   {
cbc     each(n->l, f);
ed3     f(n->val);
f5a     each(n->r, f);
cbb   }
cbb }
d41
d41 // Split treap rooted at n in two treaps containing
     positions [0, k)
d41 // and [k,...)
de9 pair<node *, node *> split(node *n, int k)
f95 {
a02   if (!n)
e70     return {NULL, NULL};
941   if (cnt(n->l) >= k) // "n->val >= k" for lower_bound(k)
f95   {
215     auto pa = split(n->l, k);
f3c     n->l = pa.second;
2f0     n->recalc();
c05     return {pa.first, n};
cbb   }
295   else
f95   {
7c2     auto pa = split(n->r, k - cnt(n->l) - 1); // and just "
     k"
d37     n->r = pa.first;
2f0     n->recalc();
7af     return {n, pa.second};
cbb   }
cbb }
d41
d41 // Merge treaps l and r keeping order (l first).
7f5 node *merge(node *l, node *r)
f95 {
0c9   if (!l)
4c1     return r;
6bf   if (!r)
792     return l;
a0a   if (l->p > r->p)
f95   {
ed7     l->r = merge(l->r, r);
bf6     l->recalc();
792     return l;
cbb   }
295   else
f95   {
654     r->l = merge(l, r->l);
cda     r->recalc();
4c1     return r;
cbb   }
cbb }
d41
d41 // Insert treap rooted at n into position pos of treap
     rooted at t.
3fc node *ins(node *t, node *n, int pos)
f95 {
ca9   auto pa = split(t, pos);
cc8   return merge(merge(pa.first, n), pa.second);
cbb }
d41
d41 // Remove node at position pos from treap rooted at t.
1e0 node *rem(node *t, int pos)
f95 {
abd   node *a, *b, *c;
cf9   tie(a, b) = split(t, pos);
005   tie(b, c) = split(b, 1);
d41
625   delete b;
a30   return merge(a, c);
cbb }
d41
d41 // Example application: do a query in range [l, r].
047 node *query(node *t, int l, int r)
f95 {
abd   node *a, *b, *c;
a83   tie(a, b) = split(t, l);
89f   tie(b, c) = split(b, r - l + 1);
d41
d41   // printf("%lld\n", b->tab);
d41
53a   return merge(merge(a, b), c);
cbb }
d41
d41 // Example application: move the range [l, r) to index k.
b51 void move(node *&t, int l, int r, int k)
f95 {
abd   node *a, *b, *c;
a83   tie(a, b) = split(t, l);
e81   tie(b, c) = split(b, r - l);
152   if (k <= l)
eeb     t = merge(ins(a, b, k), c);
295   else
646     t = merge(a, ins(c, b, k - r));
cbb }
cbb } // namespace treap
Full file hash: 02c35c
```

2.13 Union Find

```
5d1 #include "../../contest/header.hpp"
d41
10c struct union_find
f95 {
fb5   vector<int> p, size;
aa0   union_find(int n) : p(n), size(n, 1)
f95   {
919     iota(p.begin(), p.end(), 0);
cbb   }
d41
7f9   int find(int a)
f95   {
```

```
0fc     return (p[a] == a) ? a : (p[a] = find(p[a]));
cbb   }
d41
d72   void join(int a, int b)
f95   {
bca     a = find(a);
b88     b = find(b);
ae9     if (a == b)
505       return;
9cf     if (size[a] < size[b])
257       swap(a, b);
264     p[b] = a;
60c     size[a] += size[b];
cbb   }
214 };
d41
Full file hash: bb32ca
```

3 Geometry

3.1 2D

```
5d1 #include "../../contest/header.hpp"
d41
d41 // 2D geometry operations. This file should not have
     algorithms.
d41 // Author: some of it by Arthur Pratti Dadalto.
d41 // Source: some of it from https://github.com/
d41 // kth-competitive-programming/kactl/blob/master/content/
     geometry/.
d41 // Usage: avoid int unless necessary.
d41
d41 // When increasing EPS, keep in mind that
d41 // sqrt(1e9^2 + 1) = 1e9 + 5e-10.
22c const double EPS = 1e-12;
d41
d41 // Point struct implementation. Some methods are useful
     only when
d41 // using this to represent vectors.
4fc template <class T>
4be struct point
f95 {
5dc   typedef point<T> P;
645   T x, y;
d41
571   explicit point(T x = 0, T y = 0) : x(x), y(y) {}
d41
0d0   bool operator<(P p) const { return tie(x, y) < tie(p.x, p
     .y); }
d41
ec7   bool operator==(P p) const { return tie(x, y) == tie(p.x,
      p.y); }
d41
279   P operator+(P p) const { return P(x + p.x, y + p.y); }
d41
40d   P operator-(P p) const { return P(x - p.x, y - p.y); }
d41
e03   P operator*(T d) const { return P(x * d, y * d); }
d41
0b9   P operator/(T d) const { return P(x / d, y / d); }
d41
57b   T dot(P p) const { return x * p.x + y * p.y; }
d41
460   T cross(P p) const { return x * p.y - y * p.x; }
d41
d41   // product sign: right hand rule from a to b.
b3f   T cross(P a, P b) const { return (a - *this).cross(b - *
     this); }
d41
d41   // Distance squared to origin.
f68   T dist2() const { return x * x + y * y; }
d41
d41   // Vector norm (distance to origin).
18b   double dist() const { return sqrt((double)dist2()); }
d41
d41   // angle to x-axis in interval [-pi, pi]
907   double angle() const { return atan2(y, x); }
d41
d41   // makes dist()=1 (unit vector).
6f5   point<double> unit() const { return *this / dist(); }
d41
d41   // rotates +90 degrees around origin.
200   P perp() const { return P(-y, x); }
d41
d41   // perpendicular unit vector.
567   point<double> normal() const { return perp().unit(); }
d41
d41   // returns point rotated 'a' radians ccw around the
     origin.
82f   point<double> rotate(double a) const
f95   {
80d     return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(
     a));
cbb   }
d41
d41   // Returns projection of vector p on this vector.
f1e   point<double> proj(P p) const
f95   {
e0b     double d = (double)dot(p);
38d     return point<double>((double)x * d, (double)y * d) / (
     double)dist2();
cbb   }
d41
d41   // Angle between the vectors in interval [-pi, pi].
     Positive if p
d41   // is ccw from this.
8ad   double angle(P p) const { return p.rotate(-angle()).angle
     (); }
214 };
d41
d41 // Solves the linear system {a * x + b * y = e
d41 //                          {c * x + d * y = f
d41 // Returns {1, {x, y}} if solution is unique, {0, {0,0}} if
      no
d41 // solution and {-1, {0,0}} if infinite solutions.
d41 // If using integer function type, this will give wrong
     answer if
d41 // answer is not integer.
d41 // TODO: test me with integer and non-integer.
4fc template <class T>
562 pair<int, point<T>> linear_solve2(T a, T b, T c, T d, T e,
     T f)
f95 {
468   point<T> retv;
256   T det = a * d - b * c;
d41
57f   if (det == 0) // Maybe do EPS compare if using floating
     point.
f95   {
cdd     if (b * f == d * e && a * f == c * e)
3d7       return {-1, point<T>()};
37d     return {0, point<T>()};
cbb   }
d41
d41   // In case solution needs to be integer, use something
     like the
d41   // line below.
d41   // assert((e * d - f * b) % det == 0 &&
d41     //        (a * f - c * e) % det == 0);
d41
848   return {1, point<T>((e * d - f * b) / det, (a * f - c * e
     ) / det)};
cbb }
d41
d41 // Represents line segments defined by two points.
4fc template <class T>
4b2 struct segment
f95 {
5dc   typedef point<T> P;
efb   P pi, pf; // Initial and final points.
d41
a76   explicit segment(P a = P(), P b = P()) : pi(a), pf(b) {}
d41
d41   // Distance from this segment to a given point.
d41   // ***IMPORTANT*** DOES NOT WORK FOR LONG LONG IF X >
     1000.
325   double dist(P p)
f95   {
58f     if (pi == pf)
ade       return (p - pi).dist();
96a     auto d = (pf - pi).dist2();
ff5     auto t = min(d, max((T)0, (p - pi).dot(pf - pi)));
0b5     return ((p - pi) * d - (pf - pi) * t).dist() / (double)
     d;
cbb   }
d41
d41   // Checks if given point belongs to segment. Use dist(p)
     <= EPS
d41   // instead when using point<double>.
0e3   bool on_segment(P p)
f95   {
50f     return p.cross(pi, pf) == 0 && (pi - p).dot(pf - p) <=
     0;
cbb   }
d41
d41   // If a unique intersection point between the line
     segments exists
d41   // then it is returned.
d41   // If no intersection point exists an empty vector is
     returned.
d41   // If infinitely many exist a vector with 2 elements is
     returned,
d41   // containing the endpoints of the common line segment.
d41   // The wrong position will be returned if P is point<ll>
     and the
d41   // intersection point does not have integer coordinates.
d41   // However, no problem in using it to check if intersects
      or not
d41   // in this case (size of vector will be correct).
d41   // *** IMPORTANT *** Products of **three** coordinates
     are used in
d41   // intermediate steps so watch out for overflow if using
     int or
d41   // long long.
f3f   vector<P> intersect(segment rhs)
f95   {
9b1     auto oa = rhs.pi.cross(rhs.pf, pi), ob = rhs.pi.cross(
     rhs.pf, pf),
1d4          oc = pi.cross(pf, rhs.pi), od = pi.cross(pf, rhs.pf)
     ;
d41
d41     // Checks if intersection is single non-endpoint point.
288     if (sign(oa) * sign(ob) < 0 && sign(oc) * sign(od) < 0)
655       return {(pi * ob - pf * oa) / (ob - oa)};
d41
4c1     set<P> s;
037     if (rhs.on_segment(pi))
f07       s.insert(pi);
672     if (rhs.on_segment(pf))
```

```
      s.insert(pf);
    if (on_segment(rhs.pi))
      s.insert(rhs.pi);
    if (on_segment(rhs.pf))
      s.insert(rhs.pf);
    return vector<P>(s.begin(), s.end());
  }
};

// Represents a line by its equation in the form a * x + b * y = c.
// Can be created from two points or directly from constants.
template <class T>
struct line
{
  typedef point<T> P;
  T a, b, c; // line a * x + b * y = c

  explicit line(P p1, P p2) // TODO: test me.
  {
    assert(!(p1 == p2));
    a = p2.y - p1.y;
    b = p1.x - p2.x;
    c = a * p1.x + b * p1.y;

    // In case of int, it is useful to scale down by gcd (e.g to
    // use in a set).
    // Might be useful to normalize here.
  }

  explicit line(T _a, T _b, T _c) : a(_a), b(_b), c(_c) {}

  // Distance from this line to a given point. TODO: test me.
  double dist(P p)
  {
    return (double)abs(a * p.x + b * p.y - c) / sqrt((double)(a * a + b * b));
  }

  // Intersects this line with another given line. See linear_solve2
  // for usage. TODO: test me.
  pair<int, P> intersect(line rhs)
  {
    return linear_solve2(a, b, rhs.a, rhs.b, c, rhs.c);
  }

    // Returns orthonogonal projection of p on the line
  point<double> proj(P p){
    point<double> ans;

    if(abs(a) > EPS)
          p.x -= c/a;
      else
          p.y -= c/b;

    double scale = (-b*p.x + a*p.y)/(a*(double)a + b*(double)b);
      ans.x = scale*-b;
      ans.y = scale*a;

    if(abs(a) > EPS)
      ans.x += c/a;
    else
      ans.y += c/b;

        return ans;
  }

  // Returns parallel line that passes by p
  line<T> parallel(P p){
    T new_c = p.x*a + p.y*b;
    return line(a, b, new_c);
  }


  // Normalize line to c >= 0, a*a + b*b == 1. Only use with double.
  line normalize()
  {
    double d = P(a, b).dist() * (c < 0 ? -1 : 1);
    return line(a / d, b / d, c / d);
  }

  // Reflects point in current line
  P reflect(P p)
  {
    P res;

    res.x = ((b * b - a * a) * p.x - 2 * a * b * p.y + 2 * a * c) / (a * a + b * b);
    res.y = ((a * a - b * b) * p.y - 2 * a * b * p.x + 2 * b * c) / (a * a + b * b);

    return res;
  }
};

// Represents a circle by its center and radius. Mostly only works
// with double.
template <class T>
struct circle
{
  typedef point<T> P;
  P center;
  T r;

  // Intersects circle with a given line. This does not work with
  // integer types.
  // If there is no intersection, returns 0 and retv is whatever.
  // If intersection is a single point, returns 1 and retv is a pair
  // of equal points.
  // If intersection is two points, return 2 and retv is the two
  // intersection points.
  // Assume points are given in no particular order. If you really
  // need it, should be leftmost first when looking from center of
  // the circle.
  int intersect(line<T> l, pair<P, P> &retv)
  {
    l = l.normalize();
    l.c -= l.a * center.x + l.b * center.y; // Recenter so that we
                            // can consider circle
                            // center in origin.
    P v(l.a, l.b);
    P p0 = v * l.c; // p0 is the point in the line closest to
              // origin.

    if (p0.dist() > r + EPS) // No intersection.
      return 0;
    else if (p0.dist() > r - EPS) // dist in [r - EPS, r + EPS] ->
                    //single point intersection at
                    // p0.
    {
      retv = {p0, p0};
      return 1;
    }

    double d = sqrt(r * r - l.c * l.c); // d is distance from p0
                        // to the intersection
                        // points.
    retv = {center + p0 + v.normal() * d, center + p0 - v.normal() * d};
    return 2;
  }

  // Intersects circle with another circle. This does not work with
  // integer types.
  // This assumes the circles do not have the same center. Check
  // this case if needed, can have 0 or infinite intersection
  // points.
  // If there is no intersection, returns 0 and retv is whatever.
  // If intersection is a single point, returns 1 and retv is a pair
  // of equal points.
  // If intersection is two points, return 2 and retv is the two
  // intersection points.
  // 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.
  int intersect(circle rhs, pair<P, P> &retv)
  {
    rhs.center = rhs.center - center;
    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);
    retv.first = retv.first + center;
    retv.second = retv.second + center;
    return num;
  }

  // Returns a pair of the two points on the circle whose tangent
  // lines intersect p.
  // If p lies within the circle NaN-points are returned. P is
  // intended to be Point<double>.
  // The first point is the one to the right as seen from the point
  // p towards the circle.
  pair<P, P> tangents(P p)
  {
    p = p - center;
    double k1 = r * r / p.dist2();
    double k2 = sqrt(k1 - k1 * k1);
    return {center + p * k1 + p.perp() * k2, center + p * k1 - p.perp() * k2};
  }

  // Finds all the outter tangent lines between current
```

```
    circle and
d41   // 'other'.
d41   // Returns the points in the current circle crossed by
    those
d41   // tangents in retV1, and in retV2 the points in the
    circle
d41   // 'other'.
d41   // First point of each pair is one line, and second point
     of each
d41   // pair is the other.
d41   // IMPORTANT: You have to verify if one circle is not
    strictly
d41   // inside the other.
d41   // IMPORTANT: Only use with double.
d41   // In the case that one circle is inside the other with
    one
d41   // tangent point p, first points equals to p, and second
    points
d41   // are out of the circles and in the tangent line;
26f   void outter_tangents(circle other, pair<P, P> &retV1,
    pair<P, P> &retV2)
f95   {
799     T a1 = asin((other.r - r) / (center - other.center).
    dist());
8f9     T a2 = -atan2(other.center.y - center.y, other.center.x
     - center.x);
57b     T a3 = asin(1) - a2 + a1;
d41
132     retV1.first = P(center.x + r * cos(a3), center.y + r *
    sin(a3));
68d     retV2.first = P(other.center.x + other.r * cos(a3),
    other.center.y + other.r * sin(a3));
d41
d41     // In the case there is one tangent point (and circles
    are
d41     // external),
d41     // sets second point in a way that the tangent line can
     be
d41     // found.
448     if (abs((center - other.center).dist() + min(r, other.r
    ) - max(r, other.r)) < EPS)
f95     {
260       P vec = center - retV1.first;
6ea       retV1.second = retV2.second = retV1.first + vec.
    rotate(asin(1));
cbb     }
d41
295     else
f95     {
7c2       line<double> l = line<double>(center, other.center);
6ff       retV1.second = l.reflect(retV1.first);
b78       retV2.second = l.reflect(retV2.first);
cbb     }
cbb   }
d41
d41   // Finds all the inner tangent lines between current
    circle
d41   // and 'other'.
d41   // Returns the points in the current circle crossed by
    those
d41   // tangents in retV1, and in retV2 the points in the
    circle
d41   // 'other'.
d41   // First point of each pair is one line, and second point
     of each
d41   // pair is the other.
d41   // IMPORTANT: You have to verify if one circle does not
    intersect
d41   // the other in more than one point (verify centers
    distance vs
d41   // r + other.r).
d41   // IMPORTANT: Only use with double.
d41   // In the case that the circles intersect in one point p
    and are
d41   // exterior to one another, points returned as first will
     be p,
d41   // and points returned as second
d41   // will be points outside the circles in the tangent line
    )
f8a   void inner_tangents(circle other, pair<P, P> &retV1, pair
    <P, P> &retV2)
f95   {
d41     // Point where inner tangents cross (when they are the
    same
d41     // line, it's the point in the segment between circle
    centers)
575     P cp = (other.center * r + center * other.r) / (r +
    other.r);
d41
d41     //Finds points for current circle
1e9     double u = r / (center - cp).dist();
32d     double angle = acos(u);
d1c     P vec = cp - center;
10b     retV1 = {center + vec.rotate(angle) * u, center + vec.
    rotate(-angle) * u};
d41
d41     //find points for other circle
9db     u = other.r / (other.center - cp).dist();
32f     angle = acos(u);
0ff     vec = cp - other.center;
8d1     retV2 = {other.center + vec.rotate(angle) * u, other.
    center + vec.rotate(-angle) * u};
d41
d41     //In the case there is one tangent point (and circles
    are
d41     // external), sets second point in a way that the
    tangent line
d41     // can be found.
b49     if (abs(r + other.r - (center - other.center).dist()) <
     EPS)
8a2       retV1.second = retV2.second = cp + vec.rotate(asin(1)
    );
cbb   }
214 };
d41
d41 // The circumcircle of a triangle is the circle
    intersecting all
d41 // three vertices.
d41 // Returns the unique circle going through points A, B and
    C (given
d41 // in no particular order).
d41 // This assumes that the triangle has non-zero area.
d41 // TODO: test specifically.
113 circle<double> circumcircle(const point<double> &A, const
    point<double> &B, const point<double> &C)
f95 {
b10   circle<double> retv;
6d2   point<double> a = C - B, b = C - A, c = B - A;
1d9   retv.r = a.dist() * b.dist() * c.dist() / abs(c.cross(b))
    / 2;
0d1   retv.center = A + (b * c.dist2() - c * b.dist2()).perp()
    / b.cross(c) / 2;
627   return retv;
cbb }
d41
d41 // Returns TWO TIMES the area of the SIMPLE (non self
    intersecting)
d41 // polygon defined in pol.
d41 // The area is NEGATIVE if the polygon is in CLOCKWISE.
4fc template <class T>
945 T area_polygon2(vector<point<T>> pol)
f95 {
3b0   T area = 0;
76f   for (int i = 0; i < (int)pol.size() - 1; i++)
861     area += pol[i].cross(pol[i + 1]);
d41
f3d   area += pol[pol.size() - 1].cross(pol[0]);
d41
742   return area;
cbb }
d41
4fc template <class T>
aa6 ostream &operator<<(ostream &os, point<T> p)
f95 {
d80   return os << "(" << p.x << ", " << p.y << ")";
cbb }
Full file hash: ec5756
```

3.2 3D

```
5d1 #include "../../contest/header.hpp"
d41 /**
630  * 3D geometry operations.
1be  * Status: tested, except for phi and theta
08b  * Source: https://github.com/kth-competitive-programming/
    kactl
c4c  */
d41
4fc template <class T>
23a struct point3D
f95 {
9cd     typedef point3D P;
d0e     typedef const P &R;
329     T x, y, z;
d41
477     explicit point3D(T x = 0, T y = 0, T z = 0) : x(x), y(y
    ), z(z) {}
c83     bool operator<(R p) const
f95     {
448         return tie(x, y, z) < tie(p.x, p.y, p.z);
cbb     }
8f1     bool operator==(R p) const
f95     {
469         return tie(x, y, z) == tie(p.x, p.y, p.z);
cbb     }
9ae     P operator+(R p) const { return P(x + p.x, y + p.y, z +
     p.z); }
54a     P operator-(R p) const { return P(x - p.x, y - p.y, z -
     p.z); }
743     P operator*(T d) const { return P(x * d, y * d, z * d);
    }
17b     P operator/(T d) const { return P(x / d, y / d, z / d);
    }
e49     T dot(R p) const { return x * p.x + y * p.y + z * p.z;
    }
8d1     P cross(R p) const
f95     {
923         return P(y * p.z - z * p.y, z * p.x - x * p.z, x *
    p.y - y * p.x);
cbb     }
b70     T dist2() const { return x * x + y * y + z * z; }
18b     double dist() const { return sqrt((double)dist2()); }
d41     //Azimuthal angle (longitude) to x-axis in interval [-
    pi, pi]
3d6     double phi() const { return atan2(y, x); }
d41     //Zenith angle (latitude) to the z-axis in interval [0,
     pi]
0fa     double theta() const { return atan2(sqrt(x * x + y * y)
    , z); }
```

```
55e     P unit() const { return *this / (T)dist(); } //makes
    dist()=1
d41     //returns unit vector normal to *this and p
685     P normal(P p) const { return cross(p).unit(); }
d41     //returns point rotated 'angle' radians ccw around axis
37a     P rotate(double angle, P axis) const
f95     {
1b5         double s = sin(angle), c = cos(angle);
989         P u = axis.unit();
6b7         return u * dot(u) * (1 - c) + (*this) * c - cross(u
    ) * s;
cbb     }
214 };
d41
d41 // Returns the shortest distance on the sphere with radius
    "radius"
d41 // between the points with azimuthal angles (longitude) f1
    and f2
d41 // from x axis and zenith angles (latitude) t1 and t2
d41 // from z axis. All angles measured in radians.
d41 // The algorithm starts by converting the spherical
    coordinates
d41 // to cartesian coordinates so if that is what you have you
     can
d41 // use only the two last rows. dx*radius is then the
    difference
d41 // between the two points in the x direction and d*radius
    is the
d41 // total distance between the points.
56c double spherical_distance(double f1, double t1,
0fa                           double f2, double t2, double
    radius)
f95 {
825     double dx = cos(t2) * cos(f2) - cos(t1) * cos(f1);
852     double dy = cos(t2) * sin(f2) - cos(t1) * sin(f1);
e95     double dz = sin(t2) - sin(t1);
c09     double d = sqrt(dx * dx + dy * dy + dz * dz);
154     return radius * 2 * asin(d / 2);
Full file hash: 1fa9d8
```

3.3 Convex Polygon and Circle Intersection

```
d41 // https://codeforces.com/gym/101158/
d41
272 #include "../../misc/ternary_search/
    ternary_search_continuous.cpp"
ad5 #include "../2d/2d.cpp"
d41
e89 #define point point<double>
76d #define line line<double>
0f3 #define circle circle<double>
1ba #define segment segment<double>
d41
d41 // Returns the intersection area between a convex polygon
    and a
d41 // circle.
d41 // Only works if circle center is inside the polygon and
d41 // the points in p are given in counter-clockwise order.
d41 // Has some precision issues, so EPS value is very relevant
    .
cea double circle_convex_polygon_intersection(const vector<
    point> &p, circle c)
f95 {
989   double retv = 0;
90f   int n = sz(p);
830   for (int i = 0; i < n; i++)
f95   {
244     line l(p[i], p[(i + 1) % n]);
7ae     segment s(p[i], p[(i + 1) % n]);
cba     pair<point,point> res;
d41
b56     vector<point> bd; // Boundary points (either in segment
    or
d41             // segment-circle intersection).
874     bd.push_back(p[i]);
acf     bd.push_back(p[(i + 1) % n]);
d41
115     if (c.intersect(l, res) == 2)
f95     {
0a2       if (s.dist(res.first) < EPS)
c5c         bd.push_back(res.first);
ad5       if (s.dist(res.second) < EPS)
98c         bd.push_back(res.second);
cbb     }
d41
b28     sort(bd.begin() + 1, bd.end(), [&bd] (point lhs, point
    rhs) { return (lhs-bd[0]).dist2() < (rhs-bd[0]).dist2();
    });
d41
f0c     if (bd.size() == 2)
f95     {
fca       if ((bd[0] - c.center).dist() < c.r + EPS && (bd[1] -
    c.center).dist() < c.r + EPS) // Segment completely
    inside.
787         retv += c.center.cross(bd[0], bd[1]) / 2;
295       else // Segment completely outside.
824         retv += c.r * c.r * (bd[0] - c.center).angle(bd[1]
    - c.center) / 2;
cbb     }
b92     else if (bd.size() == 3) // One point inside circle and
    one
d41                  // outside.
f95     {
c60       if ((bd[0] - c.center).dist() < c.r + EPS)
f95       {
d41         // Point 0 is inside.
787         retv += c.center.cross(bd[0], bd[1]) / 2;
874         retv += c.r * c.r * (bd[1] - c.center).angle(bd[2]
    - c.center) / 2;
cbb       }
295       else
f95       {
d41         // Point 2 is inside
2c3         retv += c.center.cross(bd[1], bd[2]) / 2;
824         retv += c.r * c.r * (bd[0] - c.center).angle(bd[1]
    - c.center) / 2;
cbb       }
cbb     }
295     else
f95     {
824       retv += c.r * c.r * (bd[0] - c.center).angle(bd[1] -
    c.center) / 2;
2c3       retv += c.center.cross(bd[1], bd[2]) / 2;
85e       retv += c.r * c.r * (bd[2] - c.center).angle(bd[3] -
    c.center) / 2;
cbb     }
cbb   }
d41
627   return retv;
cbb }
d41
d41 // This finds the maximum intersection between convex
    polygon and
d41 // any circle of a given radius.
d41 // Has some precision issues, review before using.
b1f double max_circle_intersection(const vector<point> &p,
    double r)
f95 {
184   circle retv;
0af   retv.r = r;
d41
0a3   auto f1 = [&](double x) {
e86     auto f2 = [&](double y) {
065       return -circle_convex_polygon_intersection(p, {point(
    x, y), r});
214     };
d41
a5b     double bot = 1e18;
781     double top = -1e18;
fe1     for (int i = 0; i < sz(p); i++)
f95     {
e22       segment s1(p[i], p[(i + 1) % sz(p)]);
36c       segment s2(point(x, -1e3), point(x, 1e3));
d41
6c4       auto inter = s2.intersect(s1);
632       if (inter.size() > 0)
f95       {
4f4         for (point a : inter)
f95         {
cba           bot = min(bot, a.y);
3b0           top = max(top, a.y);
cbb         }
cbb       }
cbb     }
d41
043     retv.center.y = ternary_search(f2, bot, top, EPS);
50a     return f2(retv.center.y);
214   };
d41
fdd   double botx = 1e18;
f58   double topx = -1e18;
fe1   for (int i = 0; i < sz(p); i++)
f95   {
f38     botx = min(botx, p[i].x);
de5     topx = max(topx, p[i].x);
cbb   }
d41
b05   retv.center.x = ternary_search(f1, botx, topx, EPS);
d41
fdb   return -f1(retv.center.x);
cbb }
d41
13a int main(void)
f95 {
1a8   int n;
c12   double r;
a68   cin >> n >> r;
cfb   vector<point> p(n);
830   for (int i = 0; i < n; i++)
243     cin >> p[i].x >> p[i].y;
d41
79d   printf("%.20lf\n", max_circle_intersection(p, r));
cbb }
Full file hash: 50f2e3
```

3.4 Closest Pair of points

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
435   Closest Pair of points O(n * log n):
bd7     Finds the closest pair of points from a set of given
    points.
d41
b95   Usage:
96d     Call closest_pair with the array of points and the
    number.
bf2     The function will modify the array.
```

```
079     Then, get the squared distance from ans and the indexes
1a5     of both points from idx.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
69a #define MAXN 112345
d41
4be struct point
f95 {
0be   ll x, y;
53e   int id;
214 };
d41
828 namespace closest_pair
f95 {
fef point tmp[MAXN];
d41
f5b ll ans = infll;
b73 int idx[2];
d41
8ce void update(point i, point j)
f95 {
fd0   ll dist = (i.x - j.x) * (i.x - j.x) + (i.y - j.y) * (i.y
     - j.y);
573   if (dist < ans)
f95   {
234     ans = dist;
7f6     idx[0] = min(i.id, j.id);
a4f     idx[1] = max(i.id, j.id);
cbb   }
cbb }
d41
581 bool compx(point a, point b) { return a.x < b.x; }
d41
71b bool compy(point a, point b) { return a.y < b.y; }
d41
98e void solve(point p[], int l, int r)
f95 {
c00   if (r - l <= 3)
f95   {
245     for (int i = l; i <= r; i++)
039       for (int j = i + 1; j <= r; j++)
2af         update(p[i], p[j]);
aa3     sort(p + l, p + r + 1, compy);
505     return;
cbb   }
d41
ae0   int mid = (l + r) / 2;
709   ll xmid = p[mid].x;
d41
b28   solve(p, l, mid), solve(p, mid + 1, r);
6c2   merge(p + l, p + mid + 1, p + mid + 1, p + r + 1, tmp,
     compy);
0dc   copy(tmp, tmp + r - l + 1, p + l);
d41
1fc   int sz = 0;
245   for (int i = l; i <= r; i++)
7a8     if ((p[i].x - xmid) * (p[i].x - xmid) < ans)
f95     {
ed1       for (int j = sz - 1; j >= 0 && (p[i].y - tmp[j].y) *
     (p[i].y - tmp[j].y) < ans; j--)
28f         update(p[i], tmp[j]);
db9       tmp[sz++] = p[i];
cbb     }
cbb }
d41
d50 void closest_pair(point p[], int n)
f95 {
ac1   ans = infll;
c55   sort(p, p + n, compx);
f8f   solve(p, 0, n - 1);
cbb }
214 }; // namespace closest_pair
d41
Full file hash: 77bdaa
```

3.5 Convex Hull - Sweep Line

```
d41
5d1 #include "../../contest/header.hpp"
d41
d41 /*
d95   Convex hull:
db7     Computes lower and upper convex hull for a set of
     points in
93e     O(n * log n).
915     Using lower and upper convex hull you can also check if
      a
cce     point belongs
6e2     to the polygon in O(log n) with the point_in_ch
     function.
d41
b95   Usage:
0fe     Upper/lower hulls start at lowest x (tie broken by
     lowest y)
f33     and end at highest x (tie broken by highest y).
f18     Points can be collinear, but convex hull will not
     contain
f9c     collinear points.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
4be struct point
f95 {
0be   ll x, y;
d41
e45   explicit point(ll x = 0, ll y = 0) : x(x), y(y) {}
d41
760   ll cross(point p1, point p2) { return (p1.x - x) * (p2.y
     - y) - (p2.x - x) * (p1.y - y); }
d41
5bb   bool operator<(const point &rhs) const { return tie(x, y)
      < tie(rhs.x, rhs.y); }
214 };
d41
249 void convex_hull(vector<point> p, vector<point> &upper,
     vector<point> &lower)
f95 {
905   sort(p.begin(), p.end());
c79   auto build = [&p](vector<point> &ch, ll tp) {
2b3     ch.push_back(p[0]), ch.push_back(p[1]);
074     for (int i = 2; i < sz(p); i++)
f95     {
d63       while (ch.size() >= 2 && tp * ch[sz(ch) - 2].cross(ch
     .back(), p[i]) >= 0)
9d9         ch.pop_back();
3db       ch.push_back(p[i]);
cbb     }
214   };
d41
5ad   build(upper, 1);
6be   build(lower, -1);
cbb }
d41
d41 // Optional.
d41 // Checks if point o is inside the convex hull area in O(
     log n).
d41 // Also returns true if point is on the convex hull
     perimeter.
c90 bool point_in_ch(point o, vector<point> &upper, vector<
     point> &lower)
f95 {
329   if (o.x < upper[0].x || o.x > upper.back().x)
d1f     return false;
d41
e22   auto check = [o](vector<point> &ch, ll tp) {
f37     int i = lower_bound(ch.begin(), ch.end(), o, [](point a
     , point b) { return a.x < b.x; }) - ch.begin();
e66     if ((i != 0 && tp * ch[i - 1].cross(ch[i], o) > 0) ||
2d9       (i + 1 < sz(ch) && tp * ch[i].cross(ch[i + 1], o) >
     0) ||
f11       (i + 2 < sz(ch) && tp * ch[i + 1].cross(ch[i + 2], o)
      > 0))
d1f       return false;
d41
8a6     return true;
214   };
d41
654   return check(upper, 1) && check(lower, -1);
cbb }
Full file hash: b7da8b
```

3.6 Convex Hull - Graham Scan

```
ad5 #include "../2d/2d.cpp"
d41
d41 /*
a55   Solution for convex hull problem (minimum polygon
     covering a set
6bc   of points) based on ordering points by angle.
324   * Finds the subset of points in the convex hull in O(Nlog
     (N)).
568   * This version works if you either want intermediary
     points in
900   segments or not (see comments delimited by //)
01b   * This version works when all points are collinear
65c   * This version works for repeated points if you add a
     label to
327   struct, and use this label in overloaded +, - and =.
c4c */
d41
d41 //Only uses 'struct point' form 2d.cpp. Apply following
     changes to use
d41 //with double:
d41 //Double version: bool operator<(P p) const { return  fabs(
     x - p.x) <
d41 //                              EPS ? y < p.y :
d41 //                              x < p.x; }
d41 //Double version: bool operator==(P p) const { return fabs(
     x - p.x) <
d41 //                              EPS &&
d41 //                          fabs(y - p.y) < EPS; }
d41
d41 /* Compara primeiro por angulo em relacao a origem e depois
      por
810   distancia para a origem */
67a template <typename T>
f30 bool cmp(point<T> a, point<T> b)
f95 {
a9b   if (a.cross(b) != 0)
c33     return a.cross(b) > 0;
ba7   return a.dist2() < b.dist2();
cbb }
d41
67a template <typename T>
2fc vector<point<T>> CH(vector<point<T>> points)
f95 {
```

```
d41   /* Encontra pivo (ponto extremos que com ctz faz parte do
      CH) */
95b   point<T> pivot = points[0];
e40   for (auto p : points)
e01     pivot = min(pivot, p);
d41
d41   /* Desloca conjunto para pivo ficar na origem e ordena
      potos pelo
e87    angulo e distancia do pivo */
9ac   for (int i = 0; i < (int)points.size(); i++)
301     points[i] = points[i] - pivot;
d41
e2c   sort(points.begin(), points.end(), cmp<ll>);
d41
9ac   for (int i = 0; i < (int)points.size(); i++)
eda     points[i] = points[i] + pivot;
d41
d41   /* Ponto extra para fechar o poligono */
36b   points.push_back(points[0]);
d41
620   vector<point<T>> ch;
d41
e40   for (auto p : points)
f95   {
d41     /* Enquanto o proximo ponto gera uma curva para a
      direita,
e6d      retira ultimo ponto atual */
d41     /* Segunda comparaÃğÃčo serve para caso especial de
      pontos
153      colineares quando se quer eliminar os intermediarios
      */
d41     // Trocar terceira comparacao pra <= para descartar
      pontos do
d41     // meio de arestas no ch
d41     // Double: trocar terceira comparaÃğÃčo por < EPS (
      descarta
d41     // pontos em arestas) ou < -EPS (mantem pinto em aresta
29f     while (ch.size() > 1 && !(p == ch[ch.size() - 2]) && ch
      [ch.size() - 2].cross(ch[ch.size() - 1], p) < 0)
9d9       ch.pop_back();
d2e     ch.push_back(p);
cbb   }
d41
d41   /*Elimina ponto extra*/
9d9   ch.pop_back();
d41
66c   return ch;
cbb }
Full file hash: 19c056
```

3.7 Min Enclosing Circle (randomized)

```
ad5 #include "../2d/2d.cpp"
d41
d41 /*
744   Minimum Enclosing Circle:
004     Given a list of points, returns a circle of minimum
      radius
e9a     such that all given points are within the circle.
eea     Runs in O(n) expected time (in practice 200 ms for 10^5
682     points).
d41
ca2   Constraints:
99b     Non-empty list of points.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
e89 #define point point<double>
0f3 #define circle circle<double>
d41
41e circle min_enclosing_circle(vector<point> p)
f95 {
b4d   shuffle(p.begin(), p.end(), mt19937(time(0)));
2e0   point o = p[0];
761   double r = 0, eps = 1 + 1e-8;
fe1   for (int i = 0; i < sz(p); i++)
197     if ((o - p[i]).dist() > r * eps)
f95     {
ba3       o = p[i], r = 0;
c79       for (int j = 0; j < i; j++)
f59         if ((o - p[j]).dist() > r * eps)
f95         {
d2b           o = (p[i] + p[j]) / 2;
065           r = (o - p[i]).dist();
674           for (int k = 0; k < j; k++)
355             if ((o - p[k]).dist() > r * eps)
f95             {
7fb               o = circumcircle(p[i], p[j], p[k]).center;
065               r = (o - p[i]).dist();
cbb             }
cbb         }
cbb     }
d41
645   return {o, r};
cbb }
d41
Full file hash: 5d3836
```

3.8 Min Enclosing Circle (ternary search)

```
ad5 #include "../2d/2d.cpp"
272 #include "../../misc/ternary_search/
    ternary_search_continuous.cpp"
d41
d41 /*
744   Minimum Enclosing Circle:
004     Given a list of points, returns a circle of minimum
      radius
e9a     such that all given points are within the circle.
a29     Runs in O(n * log^2((top - bot) / eps)) (in practice
      2.5s at
652     best for 10^5 points).
d41
ca2   Constraints:
99b     Non-empty list of points.
d41
b95   Usage:
63c     The coordinates of the circle's center must be in the
      range
85b     [bot, top].
bf4     eps specifies the precision of the result, but set it
      to a
9e1     higher value than necessary since the error in x
      affects the
2b1     y value.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
e89 #define point point<double>
0f3 #define circle circle<double>
d41
e17 circle min_enclosing_circle(const vector<point> &p, double
     bot = -1e9, double top = 1e9, double eps = 1e-9)
f95 {
184   circle retv;
d41
0a3   auto f1 = [&](double x) {
d99     auto f2 = [&](double y)
f95     {
996       double r = 0;
fe1       for (int i = 0; i < sz(p); i++)
62a         r = max(r, (p[i].x - x)*(p[i].x - x) + (p[i].y - y)
      *(p[i].y - y));
4c1       return r;
214     };
410     retv.center.y = ternary_search(f2, bot, top, eps);
50a     return f2(retv.center.y);
214   };
d41
596   retv.center.x = ternary_search(f1, bot, top, eps);
3b2   retv.r = sqrt(f1(retv.center.x));
d41
627   return retv;
cbb }
Full file hash: 2acede
```

3.9 Rotating Calipers - Antipodal

```
b79 #include "../graham_scan_convex_hull/graham_scan.cpp"
d41
d41 /*
63c   Antipodal pairs O(n):
159     Uses rotating calipers techinique to find all antipodal
      pairs.
1c0     Returned list will be such the the entire polygon lies
      between
d26     the line defined by (p[retv[i].first],
c13                  p[(retv[i].first + 1) % n])
c53     and a parallel line passing by p[retv[i].second].
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
d41 // p is a convex hull in ccw order with no duplicate or
    collinear
d41 // points. Might not work as expected for two points.
9bb vector<pii> antipodal_pairs(const vector<point<ll>> &p)
f95 {
159   int j = 1, n = sz(p);
070   vector<pii> retv;
830   for (int i = 0; i < n; i++)
f95   {
d41     // While j + 1 is farther from segment {i, i+1} than j.
c68     while (p[i].cross(p[(i + 1) % n], p[(j + 1) % n]) > p[i
      ].cross(p[(i + 1) % n], p[j]))
600       j = (j + 1) % n;
d41
d89     retv.push_back({i, j});
d41
d41     // If j + 1 is at the same distance as j, both pairs
      are
d41     // antipodal.
24f     if (p[i].cross(p[(i + 1) % n], p[(j + 1) % n]) == p[i].
      cross(p[(i + 1) % n], p[j]))
9b9       retv.push_back({i, (j + 1) % n});
cbb   }
d41
627   return retv;
cbb }
Full file hash: 8bebf6
```

3.10 Rotating Calipers - Convex Polygon Bouding Box

```
b79 #include "../graham_scan_convex_hull/graham_scan.cpp"
d41
d41 /*
685   Bounding Box O(n):
0d0     Finds the smallest perimeter for a rotated rectangle
ce1     that covers the entire given convex polygon.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
d41 // p is a convex hull in ccw order with no duplicate or
d41 // collinear points. Might not work as expected for two
     points.
6b7 double min_bounding_box_perimeter(const vector<point<ll>> &
    p)
f95 {
159   int j = 1, n = sz(p);
3bf   int k = 1, l = 1;
d41
49d   double ans = 1e18;
830   for (int i = 0; i < n; i++)
f95   {
d41     // While j + 1 is farther from segment {i, i+1} than j.
c68     while (p[i].cross(p[(i + 1) % n], p[(j + 1) % n]) > p[i
    ].cross(p[(i + 1) % n], p[j]))
600       j = (j + 1) % n;
d41
147     if (i == 0)
e37       l = j;
d41
b6d     while ((p[(i + 1) % n] - p[i]).dot(p[(k + 1) % n] - p[k
    ]) > 0)
399       k = (k + 1) % n;
d41
881     while ((p[(i + 1) % n] - p[i]).dot(p[(l + 1) % n] - p[l
    ]) < 0)
f24       l = (l + 1) % n;
d41
6c5     line<ll> ln(p[i], p[(i + 1) % n]);
94c     ans = min(ans, 2 * ln.dist(p[j]) +
716               2 * (p[(i + 1) % n] - p[i]).proj(p[k] - p[i
    ]).dist() +
ad9               2 * (p[(i + 1) % n] - p[i]).proj(p[l] - p[i
    ]).dist());
cbb   }
d41
ba7   return ans;
cbb }
Full file hash: 3eb318
```

3.11 Rotating Calipers - Convex Polygon Diameter

```
21e #include "antipodal_pairs.cpp"
d41
d41 /*
31a   Polygon Diameter O(n):
c98     Gets the largest distance for a pair of points
f25     in a convex polygon.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
d41 // p is a convex hull in ccw order with no duplicate or
d41 // collinear points.
cf5 double convex_polygon_diameter(const vector<point<ll>> &p)
f95 {
fc8   vector<pii> anti = antipodal_pairs(p);
d41
989   double retv = 0;
623   for (pii a : anti)
f95   {
285     if ((p[a.first] - p[a.second]).dist() > retv)
46f       retv = (p[a.first] - p[a.second]).dist();
d41
6bd     if ((p[(a.first + 1) % sz(p)] - p[a.second]).dist() >
    retv)
5f5       retv = (p[(a.first + 1) % sz(p)] - p[a.second]).dist
    ();
cbb   }
d41
627   return retv;
cbb }
Full file hash: fd37ea
```

3.12 Rotating Calipers - Convex Polygon Width

```
21e #include "antipodal_pairs.cpp"
d41
d41 /*
bd4   Polygon Width O(n):
eff     Gets the smallest width for a "tunnel" by which you can
b9a     slide the convex polygon.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
d41 // p is a convex hull in ccw order with no duplicate or
d41 // collinear points.
c40 double convex_polygon_width(const vector<point<ll>> &p)
f95 {
fc8   vector<pii> anti = antipodal_pairs(p);
d41
80b   double retv = 1e18;
fe4   for (int i = 0; i < sz(anti); i++)
f95   {
794     line<ll> l(p[anti[i].first], p[(anti[i].first + 1) % sz
    (p)]);
63e     if (l.dist(p[anti[i].second]) < retv)
b79       retv = l.dist(p[anti[i].second]);
cbb   }
d41
627   return retv;
cbb }
Full file hash: 353b1b
```

4 Graph

4.1 2-Sat

```
d41
5d1 #include "../../contest/header.hpp"
d41
d41 /*
e43   2-SAT O(N + E):
55f     Calculates a valid assignment to boolean variables a, b
    , c,...
60b     to a 2-SAT problem, so that an expression of the type (
    a || b)
4ec     && (!a || c) && (d || !b) && ...
be1     becomes true, or reports that it is unsatisfiable.
d41
ca2   Constraints:
340     Variables are labeled form 0 to n-1.
d41
b95   Usage:
ba3     Negated variables are represented by bit-inversions (~x
    ).
d41
064   Usage sample:
1f5     two_sat ts(number of boolean variables);
229     ts.either(0, ~3); // Var 0 is true or var 3 is false
25a     ts.set_true(2); // Var 2 is true
3bc     ts.set_true(~0); // Var 0 is false
433     ts.at_most_one({0,~1,2}); // <= 1 of vars 0, ~1 and 2
    are true
742     ts.solve(); // Returns true iff it is solvable
031     ts.values[0..N-1] holds the assigned values to the vars
d41
1d1   Source: https://github.com/kth-competitive-programming/
    kactl/blob/
4ca       master/content/graph/2sat.h
c4c */
d41
48c struct two_sat
f95 {
1a8   int n;
309   vector<vector<int>> graph;
21f   vector<int> values; // 0 = false, 1 = true
d41
16c   two_sat(int n = 0) : n(n), graph(2 * n) {}
d41
d41   // a || b.
c34   void either(int a, int b)
f95   {
80f     a = max(2 * a, -1 - 2 * a);
b0d     b = max(2 * b, -1 - 2 * b);
c5f     graph[a].push_back(b ^ 1);
87d     graph[b].push_back(a ^ 1);
cbb   }
d41
d41   // x == true.
ac4   void set_true(int x) { either(x, x); } // (optional)
d41
6bd   int add_var() // (optional)
f95   {
b3b     graph.emplace_back();
b3b     graph.emplace_back();
695     return n++;
cbb   }
d41
d41   // Zero or one of variables in the list must be true.
d41   // This will create auxiliary variables.
485   void at_most_one(const vector<int> &li) // (optional)
f95   {
3e5     if (sz(li) <= 1)
505       return;
da9     int cur = ~li[0];
0b7     for (int i = 2; i < sz(li); i++)
f95     {
786       int next = add_var();
909       either(cur, ~li[i]);
86e       either(cur, next);
d1a       either(~li[i], next);
072       cur = ~next;
cbb     }
d41
ed7     either(cur, ~li[1]);
cbb   }
d41
71b   vector<int> val, comp, z;
```

```
da4   int time = 0;
d41
9a6   int dfs(int i)
f95   {
9de     int low = (val[i] = ++time), x;
c5b     z.push_back(i);
a17     for (int e : graph[i])
7c7       if (!comp[e])
0e8         low = min(low, val[e] ? val[e] : dfs(e));
284     if (low == val[i])
f95     {
d45       do
f95       {
792         x = z.back();
a04         z.pop_back();
7cc         comp[x] = low;
142         if (values[x >> 1] == -1)
378           values[x >> 1] = x & 1;
fb5       } while (x != i);
cbb     }
d41
3e1     return val[i] = low;
cbb   }
d41
d41   // Returns true if solution exists and values[0..n-1] holds the
d41   // assigned values to the vars.
fcd   bool solve()
f95   {
bf1     values.assign(n, -1);
c5b     val.assign(2 * n, 0);
e20     comp = val;
3df     for (int i = 0; i < 2 * n; i++)
f89       if (!comp[i])
1e5         dfs(i);
830     for (int i = 0; i < n; i++)
17e       if (comp[2 * i] == comp[2 * i + 1])
d1f         return false;
8a6     return true;
cbb   }
214 };
Full file hash: fff52a
```

4.2 Biconnected Components

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
df7   Finding bridges, articulation points and biconnected components in
5bd     O(V + E):
9ca     A bridge is an edge whose removal splits the graph in two
81e     connected components.
b99     An articulation point is a vertex whose removal splits the
26a     graph in two connected components.
d41
db8     A biconnected component (or 2VCC) is a maximal subgraph where
081     the removal of any vertex doesn't
4ee         make the subgraph disconnected. In other words, it is a
ca6     maximal 2-vertex-connected (2VC) subgraph.
d41
d9b         A 2-connected graph is a 2VC one, except that a---b is
3c1     considered 2VC but not 2-connected.
d41
3a5     Useful theorems:
d41
f3f         A 2-edge connected (2EC) graph is a graph without bridges.
7bd     Any 2-connected graph is also 2EC.
d41
655         Let G be a graph on at least 2 vertices. The following
858     propositions are equivalent:
81e             * (i) G is 2-connected;
576             * (ii) any two vertices are in a cycle; (a cycle can't
804               repeat vertices)
f8a             * (iii) any two edges are in a cycle and degree (G) >= 2;
654             * (iv) for any three vertices x,y et z, there is a
b4b               (x,z)-path containing y.
81e         Let G be a graph on at least 3 vertices. The following
858     propositions are equivalent:
346             * (i) G is 2-edge-connected;
488             * (ii) any edge is in a cycle;
1ee             * (iii) any two edges are in a tour and degree >= 1;
96e             * (iv) any two vertices are in a tour
cf2               (a tour can repeat vertices)
d41
c57     If G is 2-connected and not bipartite, all vertices belong to
b96     some odd cycle. And any two vertices are in a odd cycle (not
e7e     really proven).
d41
bcc         If G is 2-edge-connected (proof by AC):
1a2           For any two vertices x, y and one edge e, there is a
509       (x, y)-walk containing e without repeating edges.
d41
727         A graph admits a strongly connected orientation if and only
56e     if it is 2EC.
002         A strong orientation of a given bridgeless undirected graph
093     may be found in linear time by performing a depth first search
fd5     of the graph, orienting all edges in the depth first search
29b     tree away from the tree root, and orienting all the remaining
215     edges (which must necessarily connect an ancestor and a
833         descendant in the depth first search tree) from the
edf     descendant to the ancestor.
d41
ca2   Constraints:
b9a     ***undirected*** graph.
80b     Vertices are labeled from 0 to n (inclusive).
1e2     Graph is connected (but for unconnected just replace single
cdc     dfs call with a loop).
d41
b95   Usage:
5b7     Create the struct setting the starting vertex (a), the maximum
ed5     vertex label (n),
525       the graph adjacency list (graph) and a callback f to apply on
6f4     the biconnected components.
f8f     Afterwards, art[i] == true if i is an articulation point.
71c     If the pair {a, i} is on the bridges list, then the edge
2db       {a, graph[a][i]} is a bridge.
7e9         The callback must receive a vector of edges {a, b} that are
6f0     in the same biconnected component.
bbe     Remember that for a single vertex, the biconnected callback
e22     will not be called.
d41
e15     Sample Usage:
0ec         auto rdm = apb(1, n, graph, [&](vector<pii> v){
f4e     set<int> s;
9ad     for (int i = 0; i < sz(v); i++)
f95     {
f19       s.insert(v[i].first);
085       s.insert(v[i].second);
cbb     }
d41
0fe     ans = max(ans, sz(s));
c0c       });
c4c */
d41
f11 struct apb
f95 {
9cf vector<int> *graph;
9cf vector<bool> art;
c90 vector<int> num /* dfs order of vertices starting at 1 */, low;
c83 vector<pii> bridges;
919 vector<pii> st;
53e int id;
d41
044 template<class F>
09c apb(int a, int n, vector<int> graph[], const F &f) : graph(graph), art(n + 1, false), num(n + 1), low(n + 1)
f95 {
0f6   id = 1;
cca   dfs(a, a, f);
cbb }
d41
044 template<class F>
dc5 void dfs(int a, int p, const F &f)
f95 {
7be   low[a] = num[a] = id++;
348   int comp = 0;
d41
142   for (int i = 0; i < sz(graph[a]); i++)
f95   {
b7a     if (num[graph[a][i]] == 0)
f95     {
d40             int si = sz(st);
f30       comp++;
8ec             st.push_back({a, graph[a][i]}); // Tree edge.
d41
fc5       dfs(graph[a][i], a, f);
085       low[a] = min(low[a], low[graph[a][i]]);
d41
bb6       if (low[graph[a][i]] >= num[a])
f95             {
558               if (a != 1)
016             art[a] = true;
d41
b91               f(vector<pii>(st.begin() + si, st.end()));
901               st.resize(si);
cbb             }
d41
0e9       if (low[graph[a][i]] > num[a])
b3c         bridges.push_back({a, i});
cbb     }
624     else if (graph[a][i] != p && num[graph[a][i]] < num[a])
```

```
f95         {
d41       // Back edge.
066             low[a] = min(low[a], num[graph[a][i]]);
8ec             st.push_back({a, graph[a][i]});
cbb         }
cbb   }
d41
85e   if (a == p && comp > 1)
016     art[a] = true;
cbb }
214 };
d41
Full file hash: 5cb0b8
```

4.3 Bipartite Matching (Hopcroft Karp)

```
2b7 #include <bits/stdc++.h>
ca4 using namespace std;
d41
d41 /*
ec2   Hopcroft-Karp:
eae     Bipartite Matching O(sqrt(V)E)
d41
ca2   Constraints:
998     Vertices are labeled from 1 to l + r (inclusive).
682     DO NOT use vertex 0.
968     Vertices 1 to l belong to left partition.
a6a     Vertices l + 1 to l + r belong to right partition.
d41
b95   Usage:
d86     Set MAXV if necessary.
706     Call init passing l and r.
0f3     Add edges to the graph from left side to right side.
526     Call hopcroft to get the matching size.
661     Then, each vertex v has its pair indicated in p[v] (or
    0
259     for not paired).
c4c */
d41
dde namespace hopcroft
f95 {
998 const int inf = 0x3f3f3f3f;
ed5 const int MAXV = 112345;
d41
309 vector<vector<int>> graph;
0a3 int d[MAXV], q[MAXV], p[MAXV], l, r;
d41
402 void init(int _l, int _r)
f95 {
0eb   l = _l, r = _r;
221   graph = vector<vector<int>>(l + r + 1);
cbb }
d41
6a1 bool bfs()
f95 {
187   int qb = 0, qe = 0;
4f2   memset(d, 0x3f, sizeof(int) * (l + 1));
a89   for (int i = 1; i <= l; i++)
8b3     if (p[i] == 0)
248       d[i] = 0, q[qe++] = i;
d41
2ca   while (qb < qe)
f95   {
e8e     int a = q[qb++];
008     if (a == 0)
8a6       return true;
c4f     for (int i = 0; i < graph[a].size(); i++)
683       if (d[p[graph[a][i]]] == inf)
a8c         d[q[qe++] = p[graph[a][i]]] = d[a] + 1;
cbb   }
d41
d1f   return false;
cbb }
d41
075 bool dfs(int a)
f95 {
008   if (a == 0)
8a6     return true;
c4f   for (int i = 0; i < graph[a].size(); i++)
7d8     if (d[a] + 1 == d[p[graph[a][i]]])
a2f       if (dfs(p[graph[a][i]]))
f95       {
460         p[a] = graph[a][i];
51e         p[graph[a][i]] = a;
8a6         return true;
cbb       }
d41
343   d[a] = inf;
d1f   return false;
cbb }
d41
68f int hopcroft()
f95 {
9e3   memset(p, 0, sizeof(int) * (l + r + 1));
fc8   int matching = 0;
d59   while (bfs())
f95   {
a89     for (int i = 1; i <= l; i++)
8b3       if (p[i] == 0)
57e         if (dfs(i))
730           matching++;
cbb   }
d41
2af   return matching;
cbb }
cbb } // namespace hopcroft
Full file hash: 976bec
```

4.4 Bridges/Articulation Points

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
627   Finding bridges and articulation points in O(V + E):
9ca     A bridge is an edge whose removal splits the graph in
    two
81e     connected components.
b99     An articulation point is a vertex whose removal splits
    the
26a     graph in two connected components.
d24     This can also be adapted to generate the biconnected
480     components of a graph, since the articulation points
    split
ebc     components.
d41
d41
ca2   Constraints:
b9a     ***undirected*** graph.
80b     Vertices are labeled from 0 to n (inclusive).
1e6     Graph is connected (otherwise it doesn't make sense).
d41
b95   Usage:
5b7     Create the struct setting the starting vertex (a), the
    maximum
1f1     vertex label (n) and the graph adjacency list (graph).
6ff     Aftewards, art[i] == true if i is an articulation point
    .
71c     If the pair {a, i} is on the bridges list, then the
    edge
2db     {a, graph[a][i]} is a bridge.
c4c */
d41
f11 struct apb
f95 {
9cf vector<int> *graph;
9cf vector<bool> art;
c90 vector<int> num /* dfs order of vertices starting at 1 */,
    low;
c83 vector<pii> bridges;
53e int id;
d41
4dc apb(int a, int n, vector<int> graph[]) : graph(graph), art(
    n + 1, false), num(n + 1), low(n + 1)
f95 {
0f6   id = 1;
bb4   dfs(a, a);
cbb }
d41
69c void dfs(int a, int p)
f95 {
7be   low[a] = num[a] = id++;
348   int comp = 0;
d41
c4f   for (int i = 0; i < graph[a].size(); i++)
f95   {
b7a     if (num[graph[a][i]] == 0)
f95     {
f30       comp++;
783       dfs(graph[a][i], a);
085       low[a] = min(low[a], low[graph[a][i]]);
d41
b28       if (a != 1 && low[graph[a][i]] >= num[a])
016         art[a] = true;
d41
0e9       if (low[graph[a][i]] > num[a])
b3c         bridges.push_back({a, i});
cbb     }
2ca     else if (graph[a][i] != p && num[graph[a][i]] < low[a])
ed0       low[a] = num[graph[a][i]];
cbb   }
d41
85e   if (a == p && comp > 1)
016     art[a] = true;
cbb }
214 };
d41
Full file hash: 780b6d
```

4.5 Centroid Decomposition

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
057   Centroid Decomposition:
6fc     Solve tree problems by divide and conquer splitting the
    tree
2b5     repeatedly on centroid.
df4     Centroid is the vertex with smallest <largerst subtree
    >.
6f6     O(n log n if process is O(sz))
d41
b95   Usage:
54a     Call put_edge to initialize the tree edges.
f44     Then call decomp(i, n) for any vertex i in the tree,
    with n
e34     being the number of vertices.
9a4     Function process will be called for a centroid <a> with
```

```
    subtree total size sz.
      In process you can use:
      graph[a][i] - graph adjacency list
      block[a] - true if you should ignore the vertex.
      sub_size[a][i] - subtree size for edge a -> graph[a][i]
      (considering only non-blocked parts).

    if process can be O(sz + h * log) where h is subtree height it
    is a lot better constant than O(sz * log)

  PRINT APPLICATION WITH THIS.

  Author: Arthur Pratti Dadalto
*/

#define MAXN 112345

void process(int a, int sz);

vector<int> graph[MAXN];
vector<int> sub_size[MAXN];
bool block[MAXN];

int dfs_centroid(int a, int p, int sz, int &centroid, int &val)
{
  int sum = 0, mx = 0, pidx = -1;
  for (int i = 0; i < sz(graph[a]); i++)
    if (graph[a][i] != p && !block[graph[a][i]])
    {
      int x = dfs_centroid(graph[a][i], a, sz, centroid, val);

      sub_size[a][i] = x;
      mx = max(x, mx);
      sum += x;
    }
    else if (graph[a][i] == p && !block[graph[a][i]])
      pidx = i;

  if (pidx != -1)
  {
    sub_size[a][pidx] = sz - sum - 1;
    mx = max(mx, sub_size[a][pidx]);
  }

  if (mx < val)
    val = mx, centroid = a;

  return sum + 1;
}

void decomp(int a, int sz)
{
  int val = inf;
  dfs_centroid(a, a, sz, a, val);

  process(a, sz);

  block[a] = true;
  for (int i = 0; i < sz(graph[a]); i++)
    if (!block[graph[a][i]])
      decomp(graph[a][i], sub_size[a][i]);
}

void put_edge(int a, int b)
{
  graph[a].push_back(b);
  sub_size[a].push_back(0);
  graph[b].push_back(a);
  sub_size[b].push_back(0);
}
Full file hash: e63204
```

4.6 Euler Tour

```
/*
    [DEFINITION]
        a) Eulerian Path: visits every edge only once, but can repeat
           vertices.
        b) Eulerian Cycle: is a eulerian path that is a cycle
           (start vertice == end vertice)
        OBS: We disconsider vertices that have indegree==outdegree==0
           (we call them as useless vertices)

    [CONDITIONS]
        [Undirected graph]
            [Path/Cycle]
            a) The number of vertices with odd degrees is 2(Eulerian
               Path) or 0(Eulerian cycle)
            b) The graph of useful vertices (see OBS above) should be
               connected
            If either of the above condition fails Euler Path/Cycle
            can't exist.
        [Directed graph]
            [Cycle]
                a) All vertices should have (indegree==outdegree)
                b) The UNDIRECTED version of the graph of useful
                   vertices (see OBS above) should be connected
            [Path]
                a) Equal to Cycle's conditions, but:
                b) There should be a vertex in the graph which has
                   (indegree+1==outdegree)
                d) There should be a vertex in the graph which has
                   (indegree==outdegree+1)
            If either of the above condition fails Euler Path/Cycle
            can't exist.
        OBS: The "connected" condition it's not explicit tested by
             the algorithm because it's enough checking the size of
             the found path.

    [COMPLEXITY] O(V + E)

    [USAGE]
        You should initialize the following global variables
            [vertices]
                * 0-indexed
                * It's fine to include useless vertices

            [edges]
                * In undirected graphs be sure that you created just
                  one edge and u,v have this edge in the outs vector.

            [n] number of total vertices (including useless)
            [m] number of total edges

        You should call init() before call euler_tour(n_edges), the
        n_edges argument is how many edges you are expecting to
        traverse in the euler_tour/walk.

        !!WARNING!!: Never modify the graph after calling init(),
        that could invalidate the references.

        [return]
            An integer vector that represents the vertices' indexes
            of the found cycle (when exists) or the found path
            (when exists). If none was found, an empty vector is
            returned.
            You can change the return value to be an integer vector
            that represent the edges' indexes.
            OBS: You can check if the returned value is a path by
            checking if ret.front() != ret.back()

        [reset]
            If the problem has several testcases, don't forget to
            reset global vars

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

namespace euler
{

#define MAXM 112345
#define MAXN 112345

struct edge
{
    int u, v, id;
};

struct vertice
{
    vi outs;           // edges indexes
    int in_degree = 0; // not used with undirected graphs
};

int n, m;
edge edges[MAXM];
vertice vertices[MAXN];
vi::iterator its[MAXN];
bool used_edge[MAXM];

void init()
{
    for (int i = 0; i < n; i++)
    {
        its[i] = vertices[i].outs.begin();
    }
```

```
cbb }
d41
76b vi euler_tour(int n_edges, int src)
f95 {
dc1     vi ret_vertices;
d41     //vi ret_edges;
a42     vector<pii> s = {{src, -1}};
365     while (!s.empty())
f95     {
448         int x = s.back().first;
ad4         int e = s.back().second;
2e1         auto &it = its[x], end = vertices[x].outs.end();
d41
f7b         while (it != end && used_edge[*it])
0b0             ++it;
d41
dda         if (it == end)
f95         {
1e3             ret_vertices.push_back(x);
d41             //ret_edges.push_back(e);
342             s.pop_back();
cbb         }
295         else
f95         {
82e             auto edge = edges[*it];
27f             int v = edge.u == x ? edge.v : edge.u;
af5             s.push_back({v, *it});
101             used_edge[*it] = true;
cbb         }
cbb     }
e07     if (sz(ret_vertices) != n_edges + 1)
316         ret_vertices.clear(); // No Eulerian cycles/paths.
d41     /*
0fa   if (sz(ret_edges) != n_edges)
99f     ret_edges.clear(); // No Eulerian cycles/paths.
c4c   */
d41
d41     // Check if is cycle ret_vertices.front() ==
    ret_vertices.back()
d41
87d     reverse(all(ret_vertices));
325     return ret_vertices;
d41
d41     /*
e1e   reverse(all(ret_edges));
95f   return ret_edges;
c4c   */
cbb }
d41
cbb } // namespace euler
Full file hash: a07957
```

4.7 Max Flow (Dinic)

```
be6 #include "../../../contest/header.hpp"
d41
d41 /*
908   Dinic:
67c     Max-flow O(V^2E)
eae     Bipartite Matching O(sqrt(V)E)
d41
ca2   Constraints:
80b     Vertices are labeled from 0 to n (inclusive).
8f4     Edge capacities must fit int (flow returned is long
    long).
d41
b95   Usage:
d86     Set MAXV if necessary.
148     Call init passing n, the source and the sink.
2d6     Add edges to the graph by calling put_edge(_undirected)
    .
fe3     Call max_flow to get the total flow. Then, individual
    edge
df7     flows can be retrieved in the graph.
22c     Note that flow will be negative in return edges.
c4c */
d41
826 namespace dinic
f95 {
729 struct edge
f95 {
bf6   int dest, cap, re, flow;
214 };
d41
998 const int inf = 0x3f3f3f3f;
855 const int MAXV = 312345;
d41
8a3 int n, s, t, d[MAXV], q[MAXV], next[MAXV];
d8f vector<vector<edge>> graph;
d41
bc6 void init(int _n, int _s, int _t)
f95 {
c99   n = _n, s = _s, t = _t;
b72   graph = vector<vector<edge>>(n + 1);
cbb }
d41
7c8 void put_edge(int u, int v, int cap)
f95 {
506   graph[u].push_back({v, cap, (int)graph[v].size(), 0});
68e   graph[v].push_back({u, 0, (int)graph[u].size() - 1, 0});
cbb }
d41
d6a void put_edge_undirected(int u, int v, int cap)
f95 {
506   graph[u].push_back({v, cap, (int)graph[v].size(), 0});
fce   graph[v].push_back({u, cap, (int)graph[u].size() - 1, 0})
    ;
cbb }
d41
6a1 bool bfs()
f95 {
187   int qb = 0, qe = 0;
3c6   q[qe++] = s;
98f   memset(d, 0x3f, sizeof(int) * (n + 1));
d66   d[s] = 0;
2ca   while (qb < qe)
f95   {
e8e     int a = q[qb++];
c9a     if (a == t)
8a6       return true;
335     for (int i = 0; i < (int)graph[a].size(); i++)
f95     {
10e       edge &e = graph[a][i];
d94       if (e.cap - e.flow > 0 && d[e.dest] == inf)
f40         d[q[qe++] = e.dest] = d[a] + 1;
cbb     }
cbb   }
d41
d1f   return false;
cbb }
d41
1a1 int dfs(int a, int flow)
f95 {
c9a   if (a == t)
99d     return flow;
106   for (int &i = next[a]; i < (int)graph[a].size(); i++)
f95   {
10e     edge &e = graph[a][i];
c6f     if (d[a] + 1 == d[e.dest] && e.cap - e.flow > 0)
f95     {
5f3       int x = dfs(e.dest, min(flow, e.cap - e.flow));
5f7       if (x == 0)
5e2         continue;
7f9       e.flow += x;
4c5       graph[e.dest][e.re].flow -= x;
ea5       return x;
cbb     }
cbb   }
d41
343   d[a] = inf;
bb3   return 0;
cbb }
d41
afa long long max_flow()
f95 {
f01   long long total_flow = 0;
d59   while (bfs())
f95   {
ba9     memset(next, 0, sizeof(int) * (n + 1));
606     while (int path_flow = dfs(s, inf))
a0d       total_flow += path_flow;
cbb   }
d41
793   return total_flow;
cbb }
cbb } // namespace dinic
Full file hash: 574d3a
```

4.8 Max Flow (Dinic w/ Scaling)

```
be6 #include "../../../contest/header.hpp"
d41
d41 /*
367   Dinic with Scaling:
8ac     Max-flow O(VE * log(MAX_CAP)), but usually slower than
    regular
952     Dinic.
d41
ca2   Constraints:
80b     Vertices are labeled from 0 to n (inclusive).
8f4     Edge capacities must fit int (flow returned is long
    long).
d41
b95   Usage:
d86     Set MAXV if necessary.
148     Call init passing n, the source and the sink.
2d6     Add edges to the graph by calling put_edge(_undirected)
    .
fe3     Call max_flow to get the total flow. Then, individual
    edge
df7     flows can be retrieved in the graph.
22c     Note that flow will be negative in return edges.
c4c */
d41
826 namespace dinic
f95 {
729 struct edge
f95 {
bf6   int dest, cap, re, flow;
214 };
d41
998 const int inf = 0x3f3f3f3f;
855 const int MAXV = 312345;
d41
19c int n, s, t, lim, d[MAXV], q[MAXV], next[MAXV];
d8f vector<vector<edge>> graph;
d41
bc6 void init(int _n, int _s, int _t)
f95 {
```

```
c99   n = _n, s = _s, t = _t;
b72   graph = vector<vector<edge>>(n + 1);
cbb }
d41
7c8 void put_edge(int u, int v, int cap)
f95 {
506   graph[u].push_back({v, cap, (int)graph[v].size(), 0});
68e   graph[v].push_back({u, 0, (int)graph[u].size() - 1, 0});
cbb }
d41
d6a void put_edge_undirected(int u, int v, int cap)
f95 {
506   graph[u].push_back({v, cap, (int)graph[v].size(), 0});
fce   graph[v].push_back({u, cap, (int)graph[u].size() - 1, 0})
    ;
cbb }
d41
6a1 bool bfs()
f95 {
187   int qb = 0, qe = 0;
3c6   q[qe++] = s;
98f   memset(d, 0x3f, sizeof(int) * (n + 1));
d66   d[s] = 0;
2ca   while (qb < qe)
f95   {
e8e     int a = q[qb++];
c9a     if (a == t)
8a6       return true;
335     for (int i = 0; i < (int)graph[a].size(); i++)
f95     {
10e       edge &e = graph[a][i];
21a       if (e.cap - e.flow >= lim && d[e.dest] == inf)
f40         d[q[qe++] = e.dest] = d[a] + 1;
cbb     }
cbb   }
d41
d1f   return false;
cbb }
d41
1a1 int dfs(int a, int flow)
f95 {
c9a   if (a == t)
99d     return flow;
106   for (int &i = next[a]; i < (int)graph[a].size(); i++)
f95   {
10e     edge &e = graph[a][i];
cbf     if (d[a] + 1 == d[e.dest] && e.cap - e.flow >= lim /*
    >= 1 ? */)
f95     {
5f3       int x = dfs(e.dest, min(flow, e.cap - e.flow));
5f7       if (x == 0)
5e2         continue;
7f9       e.flow += x;
4c5       graph[e.dest][e.re].flow -= x;
ea5       return x;
cbb     }
cbb   }
d41
343   d[a] = inf;
bb3   return 0;
cbb }
d41
afa long long max_flow()
f95 {
f01   long long total_flow = 0;
aab     for (lim = (1 << 30); lim >= 1; lim >>= 1)
d59     while (bfs())
f95     {
ba9       memset(next, 0, sizeof(int) * (n + 1));
606       while (int path_flow = dfs(s, inf))
a0d         total_flow += path_flow;
cbb     }
d41
793   return total_flow;
cbb }
cbb } // namespace dinic
Full file hash: ac7da7
```

4.9 Min Cost Max Flow

```
2b7 #include <bits/stdc++.h>
ca4 using namespace std;
d41
d41 /*
dfc   Min-Cost Max-Flow: O(V^2E^2)
078     Finds the maximum flow of minimum cost.
d41
ca2   Constraints:
80b     Vertices are labeled from 0 to n (inclusive).
b01     Edge cost and capacities must fit int (flow and cost
3b8     returned are long long).
75e     Edge Cost must be non-negative.
d41
b95   Usage:
d86     Set MAXV if necessary.
148     Call init passing n, the source and the sink.
909     Add edges to the graph by calling put_edge.
926     Call mincost_maxflow to get the total flow and its cost
f08     (in this order).
772     Individual edge flows can be retrieved in the graph.
22c     Note that flow will be negative in return edges.
c4c */
d41
ad1 typedef long long ll;
d29 typedef pair<long long, long long> pll;
d41
e3d namespace mcmf
f95 {
729 struct edge
f95 {
60f   int dest, cap, re, cost, flow;
214 };
d41
ed5 const int MAXV = 112345;
6a4 const ll infll = 0x3f3f3f3f3f3f3f3fLL;
998 const int inf = 0x3f3f3f3f;
d41
128 int n, s, t, p[MAXV], e_used[MAXV];
97e bool in_queue[MAXV];
c97 ll d[MAXV];
d41
d8f vector<vector<edge>> graph;
d41
bc6 void init(int _n, int _s, int _t)
f95 {
c99   n = _n, s = _s, t = _t;
b72   graph = vector<vector<edge>>(n + 1);
cbb }
d41
a4a void put_edge(int u, int v, int cap, int cost)
f95 {
bd3   graph[u].push_back({v, cap, (int)graph[v].size(), cost,
    0});
2b8   graph[v].push_back({u, 0, (int)graph[u].size() - 1, -cost
    , 0});
cbb }
d41
b34 bool spfa()
f95 {
664   memset(in_queue, 0, sizeof(bool) * (n + 1));
9ee   memset(d, 0x3f, sizeof(ll) * (n + 1));
26a   queue<int> q;
d66   d[s] = 0;
e28   p[s] = s;
08b   q.push(s);
ee6   while (!q.empty())
f95   {
093     int a = q.front();
833     q.pop();
e72     in_queue[a] = false;
d41
c4f     for (int i = 0; i < graph[a].size(); i++)
f95     {
10e       edge &e = graph[a][i];
6fa       if (e.cap - e.flow > 0 && d[e.dest] > d[a] + e.cost)
f95       {
3bf         d[e.dest] = d[a] + e.cost;
6d6         p[e.dest] = a;
183         e_used[e.dest] = i;
277         if (!in_queue[e.dest])
b34           q.push(e.dest);
04f         in_queue[e.dest] = true;
cbb       }
cbb     }
cbb   }
d41
d1c   return d[t] < infll;
cbb }
d41
996 pll mincost_maxflow()
f95 {
f04   pll retv = pll(0, 0);
d93   while (spfa())
f95   {
e98     int x = inf;
c9b     for (int i = t; p[i] != i; i = p[i])
d4a       x = min(x, graph[p[i]][e_used[i]].cap - graph[p[i]][
    e_used[i]].flow);
c9b     for (int i = t; p[i] != i; i = p[i])
dc7       graph[p[i]][e_used[i]].flow += x, graph[i][graph[p[i
    ]][e_used[i]].re].flow -= x;
d41
759     retv.first += x;
1be     retv.second += x * d[t];
cbb   }
d41
627   return retv;
cbb }
cbb } // namespace mcmf
Full file hash: 5bd8df
```

4.10 Gomory Hu (Min cut)

```
5e4 #include "../flow/dinic/dinic.cpp"
d41
d41 /*
ecc   Gomory-Hu Tree construction O(V * flow_time) (so O(V^3E),
      but not
72b   really):
854     The Gomory-Hu tree of an undirected graph with
    capacities is a
4fa     weighted
33f     tree that represents the minimum s-t cuts for all s-t
    pairs in
676     the graph.
d41
e0f     The minimum cut cost between vertices s and t is the
    minimum
63a     cost of an edge on the path from s to t in the Gomory-
    Hu tree.
```

```
d41
ca2   Constraints:
ea5     Vertices are labeled from 0 to n-1 (inclusive).
ecd     Undirected graph.
d41
b95   Usage:
442     Check Dinic usage.
a79     Create struct and call add edge for each edge in the
    graph.
53b     Then, just call solve passing the number of vertices.
d41
994     The vector returned will have size n and for each i >
    0,
41f     retv[i] is a pair (cost, parent) representing an edge
c6d     (i, parent) in the Gomory-Hu tree.
212     retv[0] means nothing.
c4c */
d41
2a4 struct gomory_hu
f95 {
a20   struct edg
f95   {
765     int u, v, cap;
214   };
d41
c4f   vector<edg> edgs;
d41
3dd   void add_edge(int u, int v, int cap)
f95   {
265     edgs.push_back({u, v, cap});
cbb   }
d41
051   vector<int> vis;
d41
0cb   void dfs(int a)
f95   {
cd2     if (vis[a])
505       return;
18f     vis[a] = 1;
264     for (auto &e : dinic::graph[a])
0f5       if (e.cap - e.flow > 0)
7ca         dfs(e.dest);
cbb   }
d41
242   vector<pair<ll, int>> solve(int n)
f95   {
56a     vector<pair<ll, int>> retv(n); // if i > 0, stores pair
    (cost,
d41                       // parent).
aa4     for (int i = 1; i < n; i++)
f95     {
93c       dinic::init(n, i, retv[i].second);
d41
9e8       for (auto &e : edgs)
893         dinic::put_edge_undirected(e.u, e.v, e.cap);
d41
180       retv[i].first = dinic::max_flow();
d41
105       vis.assign(n, 0);
1e5       dfs(i);
d41
197       for (int j = i + 1; j < n; j++)
a32         if (retv[j].second == retv[i].second && vis[j])
9cc           retv[j].second = i;
cbb     }
d41
627     return retv;
cbb   }
214 };
Full file hash: 3fe14c
```

4.11 Heavy-Light Decomposition

```
2b7 #include<bits/stdc++.h>
d41
ca4 using namespace std;
d41
eed #define ll long long
efe #define pb push_back
d41
3a6 typedef vector<ll> vll;
990 typedef vector<int> vi;
d41
e06 #define MAXN 100010
d41
d41 //Vetor que guarda a arvore
698 vector<vi> adj;
d41
9e6 int subsize[MAXN], parent[MAXN];
d41 //Inciar chainHead com -1; e chainSize e chainNo com 0.
080 int chainNo = 0, chainHead[MAXN], chainPos[MAXN], chainInd[
    MAXN], chainSize[MAXN];
42a void hld(int cur){
cb4   if(chainHead[chainNo] == -1)
659     chainHead[chainNo] = cur;
d41
3a4   chainInd[cur] = chainNo;
220   chainPos[cur] = chainSize[chainNo];
6f0   chainSize[chainNo]++;
d41
891   int ind = -1, mai = -1;
9d9   for(int i = 0; i < (int)adj[cur].size(); i++){
9ff     if(adj[cur][i] != parent[cur] && subsize[adj[cur][i]] >
    mai){
31f       mai = subsize[adj[cur][i]];
b9b       ind  = i;
cbb     }
cbb   }
d41
27d   if(ind >= 0)
f23     hld(adj[cur][ind]);
d41
e50   for(int i = 0; i < (int)adj[cur].size(); i++)
6f7     if(adj[cur][i] != parent[cur] && i != ind){
959       chainNo++;
270       hld(adj[cur][i]);
cbb     }
cbb }
d41
d41 //usar LCA para garantir que v eh pai de u!!
f17 ll query_up(int u, int v){
c20   int uchain = chainInd[u], vchain = chainInd[v];
bdd   ll ans = 0LL;
d41
31e   while(1){
f52     if(uchain == vchain){
d41       //Query deve ir de chainPos[i] ate chainPos[v]
7d2       ll cur = /*sum(chainPos[u], uchain) - (chainPos[u] ==
    0? 0LL : sum(chainPos[v] - 1, vchain))*/;
d13       ans += cur;
c2b       break;
cbb     }
d41
d41     //Query deve ir de chainPos[i] ate o fim da estrutura
d41     //ll cur = sum(chainPos[u], uchain);
d13     ans += cur;
a25     u = chainHead[uchain];
803     u = parent[u];
cab     uchain = chainInd[u];
cbb   }
ba7   return ans;
cbb }
d41
b7a int dfs0(int pos, int prev = -1){
c92   int res  = 1;
978   for(int i = 0; i < (int)adj[pos].size(); i++){
ec4     int nx  = adj[pos][i];
773     if(nx != prev){
3f2       res += dfs0(nx, pos);
522       parent[nx] = pos;
cbb     }
cbb   }
a18   return subsize[pos] = res;
cbb }
d41
0b8 int main()
f95 {
d41   //Salvar arvore em adj
d41
d41   //Inicializa estrutura de dados
b75   memset(chainHead, -1, sizeof(chainHead));
d41
d41   //Ou 0, se for o no raiz
bf6   dfs0(1);
bac   hld(1);
d41
d41   //Inicializar estruturas usadas
cbb }
Full file hash: 90a698
```

4.12 Heavy-Light Decomposition (Dadalto)

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
e4c   Heavy Light Decomposition:
ea0     Splits a tree in a set of vertex disjoint heavy paths
    such
53c     that each path from a node to the root passes at most
    log(n)
829     different heavy paths.
1ab     This allows data structures to be implement with
    queries and
db4     updates on tree paths in log(n) * data_structure_time.
d41
b95   Usage:
bfc     Create the struct passing a tree root (a), the number
    of
8d2     vertices (n) and the graph. Tested with 1 <= a <= n,
    but
176     should work with 0 <= a <= n.
d41
b8d     The data structure DS class should implement single
    element
d42     updates or range updates as needed and range queries
9f6     according to the form defined in update, update_path
    and
20b     query_path.
104     DS should also have a constructor specifying the size
e5e     and should support operations in range [0, size - 1].
b6d     IMPORTANT: DS should handle empty queries [x + 1, x].
11c     IMPORTANT: function applied in DS should be commutative
185     and associative. (If not commutative, check out
    application
ac4     for GSS7)
d41
c91     VALUES_IN_VERTICES indicates if the tree values are in
    vertices
157     or in edges. In case of edges, update(v, value)
```

```
fcf     should be called for the downward vertex of each edge.
d41
fb2     See application for more information.
d41
6e9   Source: adapted from codeforces blog (https://codeforces.
    com/blog/
699                         entry/22072).
c4c */
d41
2cd template <class DS, bool VALUES_IN_VERTICES> // DS for data
     structure.
d41                          // Values in vertices,
d41                          // true or false.
62b struct heavy_light
f95 {
119   vector<int> p, heavy, h; // parent, heavy child of vertex
    ,
d41                 // height of vertex.
d41
fbf   vector<int> num;      // number of vertex (in an order
    where
d41                 // paths are contiguos intervals).
d41
be7   vector<int> root;     // root of heavy path of a given
    vertex.
ddc   DS ds;
d41
e5f   template <class G>
c10   heavy_light(int a, int n, const G &graph) : p(n + 1),
    heavy(n + 1, -1), h(n + 1), num(n + 1), root(n + 1), ds(n
    + 1)
f95   {
42e     p[a] = a;
d3d     h[a] = 0;
57e     dfs(graph, a);
cad     for (int i = 0, id = 0; i <= n; ++i)
3c7       if (heavy[p[i]] != i) // parent of the root is itself
    ,
d41                   // so this works.
fc8         for (int j = i; j != -1; j = heavy[j])
f95         {
6d9           root[j] = i;
9c5           num[j] = id++;
cbb         }
cbb   }
d41
e5f   template <class G>
57e   int dfs(const G &graph, int a)
f95   {
d0a     int size = 1, max_subtree = 0;
23e     for (int u : graph[a])
88c       if (u != p[a])
f95       {
6c3         p[u] = a;
ada         h[u] = h[a] + 1;
c1c         int subtree = dfs(graph, u);
9ea         if (subtree > max_subtree)
d98           heavy[a] = u, max_subtree = subtree;
48e         size += subtree;
cbb       }
1c6     return size;
cbb   }
d41
2f7   template <class BO> // BO for binary_operation
72a   void process_path(int u, int v, BO op)
f95   {
d42     for (; root[u] != root[v]; v = p[root[v]])
f95     {
2ce       if (h[root[u]] > h[root[v]])
7fa         swap(u, v);
857       op(num[root[v]], num[v]);
cbb     }
ce9     if (h[u] > h[v])
7fa       swap(u, v);
f0b     op(num[u] + (VALUES_IN_VERTICES ? 0 : 1), num[v]);
cbb   }
d41
4fc   template <class T>
449   void update(int v, const T &value)
f95   {
2bf     ds.update(num[v], value);
cbb   }
d41
4fc   template <class T>
67d   T query(int v)
f95   {
678     return ds.get(num[v], num[v]);
cbb   }
d41
4fc   template <class T>
47b   void update_path(int u, int v, const T &value)
f95   {
bef     process_path(u, v, [this, &value](int l, int r) { ds.
    update(l, r, value); });
cbb   }
d41
af3   template <class T, class F>
ed6   T query_path(int u, int v, T res /* initial value */, F
    join /* join value with query result */)
f95   {
968     process_path(u, v, [this, &res, &join](int l, int r) {
    res = join(res, ds.get(l, r)); });
b50     return res;
cbb   }
214 };
Full file hash: 80b4be
```

4.13 LCA

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
b47   LCA:
0e9     Solve lowest common ancestor queries in O(log(n))
19e     with O(n*log(n)) preprocessing time and O(n*log(n))
c60     memory.
d41
b95   Usage:
a13     Initialize struct with tree root, number of vertices
ed5     and graph. Has been tested with label in [1, n], but
    should
a7e     work for labels in [0, n].
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
4be struct lca_preprocess
f95 {
6c6   int lgn;
544   vector<int> h;
58b   vector<vector<int>> p;
9cf   vector<int> *graph;
d41
0cb   void dfs(int a)
f95   {
142     for (int i = 0; i < sz(graph[a]); i++)
cb4       if (graph[a][i] != p[0][a])
f95       {
6f4         h[graph[a][i]] = h[a] + 1;
7a7         p[0][graph[a][i]] = a;
fde         dfs(graph[a][i]);
cbb       }
cbb   }
d41
cf2   lca_preprocess(int root, int n, vector<int> graph[]) : h(
    n + 1), graph(graph)
f95   {
5ff     lgn = 31 - __builtin_clz(n + 1);
445     p.assign(lgn + 1, vector<int>(n + 1, 0));
d41
8c9     p[0][root] = root;
7a0     h[root] = 0;
14e     dfs(root);
d41
05d     for (int i = 1; i <= lgn; i++)
f63       for (int j = 0; j <= n; j++)
98f         p[i][j] = p[i - 1][p[i - 1][j]];
cbb   }
d41
4cd   int lca(int a, int b)
f95   {
be7     if (h[a] < h[b])
257       swap(a, b);
29e     for (int i = lgn; i >= 0; i--)
a97       if (h[p[i][a]] >= h[b])
e27         a = p[i][a];
d41
ae9     if (a == b)
3f5       return a;
d41
29e     for (int i = lgn; i >= 0; i--)
e5d       if (p[i][a] != p[i][b])
f95       {
e27         a = p[i][a];
634         b = p[i][b];
cbb       }
d41
d12     return p[0][a];
cbb   }
d41
d2d   int dist(int a, int b)
f95   {
718     return h[a] + h[b] - 2 * h[lca(a, b)];
cbb   }
214 };
Full file hash: 0872ca
```

4.14 Min-Cut Global

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
bed   Global Min Cut O(n^3):
b25     Given an undirected weighted graph, find the minimum
    cut
365     regardless of which set of vertices it splits.
d41
b95   Usage:
ec5     Vertices from 0 to n-1. Give adjecency matrix of
    weights.
18f     Function returns total cost of min cut and set of
    vertices
50b     forming one side of the cut.
dd8     Sum of edge weights must fit int.
c4c */
d41
320 pair<int, vector<int>> mincut(int n, vector<vector<int>> g
    /*adj matrix*/)
f95 {
d9a   int best_cost = inf;
```

```
6eb   vector<int> best_cut;
d41
c91   vector<int> v[n];
6cb   for (int i = 0; i < n; ++i)
9f7     v[i].assign(1, i);
214   int w[n];
9ec   bool exist[n], in_a[n];
99e   memset(exist, true, sizeof(exist));
f34   for (int ph = 0; ph < n - 1; ++ph)
f95   {
9e8     memset(in_a, false, sizeof(in_a));
e3e     memset(w, 0, sizeof(w));
548     for (int it = 0, prev; it < n - ph; ++it)
f95     {
0a8       int sel = -1;
6cb       for (int i = 0; i < n; ++i)
1b3         if (exist[i] && !in_a[i] && (sel == -1 || w[i] > w[
    sel]))
403           sel = i;
cb8       if (it == n - ph - 1)
f95       {
25f         if (w[sel] < best_cost)
5e0           best_cost = w[sel], best_cut = v[sel];
899         v[prev].insert(v[prev].end(), v[sel].begin(), v[sel
    ].end());
6cb         for (int i = 0; i < n; ++i)
d18           g[prev][i] = g[i][prev] += g[sel][i];
0e8         exist[sel] = false;
cbb       }
295       else
f95       {
96a         in_a[sel] = true;
6cb         for (int i = 0; i < n; ++i)
f57           w[i] += g[sel][i];
0b7         prev = sel;
cbb       }
cbb     }
cbb   }
d41
bc1   return pair<int, vector<int>>(best_cost, best_cut);
cbb }
d41
Full file hash: 40ea3a
```

4.15 Strongly Connected Components

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
eab   Strongly connected components in O(V + E):
970     Finds all strongly connected components of a graph.
ec1       A strongly connected component is a maximal set of
    vertices
5e5       such that
de0       every vertex can reach every other vertex in the
    component.
0a2       The graph where the SCCs are considered vertices is
     a DAG.
d41
ca2   Constraints:
000     Vertices are labeled from 1 to n (inclusive).
d41
b95   Usage:
049     Create the struct setting the maximum vertex label (n)
    and the
510       graph adjacency list (graph).
ee4     Aftewards, ncomp has the number of SCCs in the graph
    and
d13       scc[i] indicates the SCC i belongs to (1 <= scc[i]
    <= ncomp).
d41
57d       sorted is a topological ordering of the graph,
    byproduct of
b1c       the algorithm.
484       if edge a -> b exists, a appears before b in the
    sorted list.
c4c */
d41
d41
73e struct scc_decomp
f95 {
9cf     vector<int> *graph;
00b     vector<vector<int>> tgraph;
1b0     vector<int> scc;
1ee     vector<bool> been;
8d3     int ncomp;
203     list<int> sorted;
d41
487     scc_decomp(int n, vector<int> graph[]) : graph(graph),
    tgraph(n + 1), scc(n + 1, 0), been(n + 1, false), ncomp(0)
f95     {
535         for (int i = 1; i <= n; i++)
637             for (int j = 0; j < graph[i].size(); j++)
142                 tgraph[graph[i][j]].push_back(i);
d41
535         for (int i = 1; i <= n; i++)
018       if(!been[i])
1e5         dfs(i);
d41
16e     for(int a : sorted)
f49       if(scc[a] == 0)
f95       {
a8f               ncomp++;
4dd         dfst(a);
cbb       }
cbb     }
d41
0cb     void dfs(int a)
f95     {
168         been[a] = true;
c4f         for(int i = 0; i < graph[a].size(); i++)
b0c             if(!been[graph[a][i]])
fde                 dfs(graph[a][i]);
ddb         sorted.push_front(a);
cbb     }
d41
9b7     void dfst(int a)
f95     {
168         been[a] = true;
d28         scc[a] = ncomp;
9c2         for(int i = 0; i < tgraph[a].size(); i++)
c48             if(scc[tgraph[a][i]] == 0)
caa                 dfst(tgraph[a][i]);
cbb     }
214 };
Full file hash: 20fe5c
```

4.16 Transitive Closure

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
a27   Transitive Closure:
aef     Given a directed graph adjacency matrix, computes
    closure,
8d5     where closure[i][j] = 1 if there is a path from i to j
3fb     in the graph.
9e9     Closure is computed in O(N^3 / 64) due to bitset.
be8     Also supports adding an edge to the graph and
bf8     updating the closure accordingly in O(N^2 / 64).
d41
ca2   Constraints:
a5b     Vertices are labeled from 0 to MAXN - 1 (inclusive).
d41
faf   Performance:
45d     Solves something that should be 1000*300^3 = 27 * 10^9
061     in 0.6 s (which is consistent with the approximation N
    ^3 / 64
8c7     since dividing by 64 we get 4 * 10^8).
c4c */
d41
097 template<int MAXN>
125 struct transitive_closure
f95 {
680   vector<bitset<MAXN>> closure;
d41
4fc   template<class T>
5a6   transitive_closure(T adj_matrix) : closure(MAXN)
f95   {
889     for (int i = 0; i < MAXN; i++)
4f1       for (int j = 0; j < MAXN; j++)
69c         closure[i][j] = adj_matrix[i][j];
d41
889     for (int i = 0; i < MAXN; i++)
4f1       for (int j = 0; j < MAXN; j++)
459         if (closure[j][i])
cbb           closure[j] |= closure[i];
cbb   }
d41
a7a   void add_edge(int a, int b)
f95   {
5de     if (closure[a][b])
505       return;
d41
ec4     closure[a].set(b);
841     closure[a] |= closure[b];
d41
889     for (int i = 0; i < MAXN; i++)
d0e       if (closure[i][a])
245         closure[i] |= closure[a];
cbb   }
214 };
d41
Full file hash: 1aed34
```

4.17 Tree Isomorphism

```
5d1 #include "../../contest/header.hpp"
d41 /*
ca5     [DEFINITION]
429         AHU-Algorithm to check if trees are isomorphic.
d41
ff5     [COMPLEXITY]
a0c         O(NlgN) // Map of strings argument + comparison-
    based sort
d41
bd7     [USAGE]
fd7         Call get_roots function to retrieve the pairs of
    centers for
af1         each tree (if the tree has just one center the pair
     will show
6cd         it twice).
de3         Call canonical function for each tree beginning
    from each
aa6         possible center (two at most).
9b5         A tree is isomorphic to another iff they share one
    canonical
cdf         value.
d41
e7f     [RESET]
```

```
f28         If the problem has several test cases, don't forget
      to reset
a65         the global vars 'label' and 'map_labels'
c4c */
d41
70e int label;
4ec map<vector<int>, int> map_labels;
d41
a46 pii get_roots(vector<vector<int>> &graph)
f95 {
26a     queue<int> q;
dce     vector<int> vis(sz(graph));
275     vector<int> degree(sz(graph));
d41
28e     for (int i = 0; i < sz(graph); i++)
f95     {
f78         if (sz(graph[i]) == 1)
3f2             q.push(i);
902         degree[i] = sz(graph[i]);
cbb     }
d41
5c4     int last = 0;
ee6     while (!q.empty())
f95     {
e4a         int u = q.front();
833         q.pop();
d41
497         if (vis[u]) continue;
150         vis[u] = 1;
d41
2fb         last = u;
d41
e13         for (int v : graph[u])
f95         {
e9b             if (degree[v] == 1)
f95             {
021                 return {u, v};
cbb             }
c2d             if (!vis[v])
f95             {
a43                 degree[u]--;
7a9                 degree[v]--;
d41
e9b                 if (degree[v] == 1)
2a1                     q.push(v);
cbb             }
cbb         }
d41
cbb     }
d41
a90     return {last, last};
cbb }
d41
c44 int canonical(int u, int p, vector<vi> &graph)
f95 {
086     vi children_labels;
e13     for (int v : graph[u])
f95     {
f6b         if (v != p)
d4e             children_labels.push_back(canonical(v, u, graph
     ));
cbb     }
d41
760     sort(all(children_labels));
08d     if (map_labels.count(children_labels) == 0)
d04         map_labels[children_labels] = label++;
58d     return map_labels[children_labels];
cbb }
Full file hash: 037355
```

5 Misc

5.1 Bit tricks

```
d41d8c
d41d8c  // Returns one plus the index of the least significant
     1-bit of x, or
d41d8c  // if x is zero, returns zero.
6b21ec  __builtin_ffs(x)
d41d8c
d41d8c  // Returns the number of leading 0-bits in x, starting
     at the most
d41d8c  // significant bit position. If x is 0, the result is
     undefined.
fe8701  __builtin_clz(x)
d41d8c
d41d8c  // Returns the number of trailing 0-bits in x, starting
      at the least
d41d8c  // significant bit position. If x is 0, the result is
     undefined.
219b56  __builtin_ctz(x)
d41d8c
d41d8c  // Returns the number of 1-bits in x.
cffc98  __builtin_popcount(x)
d41d8c
d41d8c  // For long long versions append ll (e.g.
     __builtin_popcountll)
d41d8c
d41d8c  // Least significant bit in x.
c9de0b  x & -x
d41d8c
d41d8c  // Iterate on non-empty submasks of a bitmask.
f255f0  for (int submask = mask; submask > 0; submask = (mask &
      (submask - 1)))
d41d8c
d41d8c  // Iterate on non-zero bits of a bitset.
8ae1a7  for (int j = btset._Find_next(0); j < MAXV; j = btset.
     _Find_next(j))
Full file hash: 2f3798
```

5.2 DP Optimization - Binary Search

```
d41 // https://codeforces.com/contest/321/problem/E
d41
5d1 #include "../../contest/header.hpp"
d41
d41 /*
4e8   Binary Search Optimization for DP:
000     Optimizes dp of the form (or similar)
7dd       dp[i][j] = min_{k < i}(dp[k][j-1] + c(k + 1, i)).
95e     The classical case is a partitioning dp, where k
     determines
b5b     the break point for the next partition.
8c2     In this case, i is the number of elements to partition
     and j
5cc     is the number of partitions allowed.
d41
f24     Let opt[i][j] be the values of k which minimize the
     function.
7e9     (in case of tie, choose the smallest)
765     To apply this optimization, you need opt[i][j] <= opt[i
     +1][j].
ef8     That means the when you add an extra element (i + 1),
     your
05f     partitioning choice will not be to include more
     elements
cb7     than before (e.g. will no go from choosing [k, i] to
218     [k-1, i+1]).
242     This is usually intuitive by the problem details.
d41
4d8     Time goes from O(n^2m) to O(nm log n).
d41
513     To apply try to write the dp in the format above and
     verify if
499     the property holds.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
349 #define MAXN 4123
fc3 #define MAXM 812
d41
14e int n, m;
159 int u[MAXN][MAXN];
2bb int tab[MAXN][MAXM];
d41
65a inline int c(int i, int j)
f95 {
229   return (u[j][j] - u[j][i - 1] - u[i - 1][j] + u[i - 1][i
     - 1]) / 2;
cbb }
d41
d41 // This is responsible for computing tab[l...r][j], knowing
      that
d41 // opt[l...r][j] is in range [low_opt...high_opt]
30d void compute(int j, int l, int r, int low_opt, int high_opt
     )
f95 {
c30   int mid = (l + r) / 2, opt = -1; // mid is equivalent to
     i in the
d41                     //original dp.
d41
722   tab[mid][j] = inf;
0e2   for (int k = low_opt; k <= high_opt && k < mid; k++)
6f6     if (tab[k][j - 1] + c(k + 1, mid) < tab[mid][j])
f95     {
451       tab[mid][j] = tab[k][j - 1] + c(k + 1, mid);
613       opt = k;
cbb     }
d41
d41   // New bounds on opt for other pending computation.
42c   if (l <= mid - 1)
c7d     compute(j, l, mid - 1, low_opt, opt);
8b4   if (mid + 1 <= r)
8aa     compute(j, mid + 1, r, opt, high_opt);
cbb }
d41
13a int main(void)
f95 {
d69   scanf("%d %d", &n, &m);
535   for (int i = 1; i <= n; i++)
947     for (int j = 1; j <= n; j++)
f95     {
433       getchar();
512       u[i][j] = getchar() - '0';
cbb     }
d41
535   for (int i = 1; i <= n; i++)
947     for (int j = 1; j <= n; j++)
a10       u[i][j] += u[i - 1][j] + u[i][j - 1] - u[i - 1][j -
     1];
d41
535   for (int i = 1; i <= n; i++)
5c5     tab[i][0] = inf;
d41
d41   // Original dp
d41   // for (int i = 1; i <= n; i++)
d41   //   for (int j = 1; j <= m; j++)
d41   //   {
```

```
d41   //    tab[i][j] = inf;
d41   //    for (int k = 0; k < i; k++)
d41   //      tab[i][j] = min(tab[i][j], tab[k][j-1] + c(k + 1,
    i);
d41   //  }
d41
2e2   for (int j = 1; j <= m; j++)
fda     compute(j, 1, n, 0, n - 1);
d41
721   cout << tab[n][m] << endl;
cbb }
Full file hash: f2bb43
```

5.3 DP Optimization - CHT

```
d41 // https://codeforces.com/contest/319/problem/C
d41
ad6 #include "../../data_structures/line_container/
    line_container.cpp"
d41
d41 /*
082   Convex Hull Trick for DP:
5cb     Transforms dp of the form (or similar)
d90       dp[i] = min_{j < i}(dp[j] + b[j] * a[i]).
bf0     Time goes from O(n^2) to O(n log n), if using online
    line
cdb     container, or O(n) if lines are inserted in order of
    slope and
0e6     queried in order of x.
d41
62e     To apply try to find a way to write the factor inside
ea0     minimization as a linear function of a value related to
     i.
c2d     Everything else related to j will become constant.
c4c */
d41
69a #define MAXN 112345
d41
a58 int a[MAXN];
c4b int b[MAXN];
d41
f80 ll tab[MAXN];
d41
13a int main(void)
f95 {
1a8   int n;
f4c   scanf("%d", &n);
830   for (int i = 0; i < n; i++)
937     scanf("%d", &a[i]);
830   for (int i = 0; i < n; i++)
264     scanf("%d", &b[i]);
d41
a44   tab[0] = 0;
79a   line_container l;
c01   l.add(-b[0], -tab[0]);
d41
aa4   for (int i = 1; i < n; i++)
f95   {
23b     tab[i] = -l.query(a[i]);
8fd     l.add(-b[i], -tab[i]);
cbb   }
d41
d41   // Original DP O(n^2).
d41   // for (int i = 1; i < n; i++)
d41   // {
d41   //   tab[i] = inf;
d41   //   for (int j = 0; j < i; j++)
d41   //     tab[i] = min(tab[i], tab[j] + a[i] * b[j]);
d41   // }
d41
```

```
cf6   cout << tab[n - 1] << endl;
cbb }
Full file hash: 722d84
```

5.4 DP Optimization - Knuth

```
d41 // https://www.spoj.com/problems/BRKSTRNG/
d41
5d1 #include "../../contest/header.hpp"
d41
d41 /*
c97   Knuth Optimization for DP:
000     Optimizes dp of the form (or similar)
06c       dp[i][j] =
572         min_{i <= k <= j}(dp[i][k-1] + dp[k+1][j] + c(i, j)
    ).
a05     The classical case is building a optimal binary tree,
    where k
fd0     determines the root.
d41
90e     Let opt[i][j] be the value of k which minimizes the
    function.
7e9     (in case of tie, choose the smallest)
d68     To apply this optimization, you need
e46       opt[i][j - 1] <= opt[i][j] <= opt[i+1][j].
7c3     That means the when you remove an element form the left
a29     (i + 1), you won't choose a breaking point more to the
    left
f8b     than before.
287     Also, when you remove an element from the right (j - 1)
    , you
bfb     won't choose a breking point more to the right than
    before.
242     This is usually intuitive by the problem details.
d41
cbb     Time goes from O(n^3) to O(n^2).
d41
513     To apply try to write the dp in the format above and
    verify if
499     the property holds.
f76     Be careful with edge cases for opt.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
dbf #define MAXN 1123
d41
c4b int b[MAXN];
1ee ll tab[MAXN][MAXN];
38a int opt[MAXN][MAXN];
ef8 int l, n;
d41
5a7 int c(int i, int j)
f95 {
33e   return b[j + 1] - b[i - 1];
cbb }
d41
13a int main(void)
f95 {
57a   while (scanf("%d %d", &l, &n) != EOF)
f95   {
535     for (int i = 1; i <= n; i++)
264       scanf("%d", &b[i]);
665     b[n + 1] = l;
00d     b[0] = 0;
d41
da4     for (int i = 1; i <= n + 1; i++)
d6b       tab[i][i - 1] = 0, opt[i][i - 1] = i;
d41
586     for (int i = n; i > 0; i--)
5d4       for (int j = i; j <= n; j++)
f95       {
639         tab[i][j] = infll;
823         for (int k = max(i, opt[i][j - 1]); k <= j && k <=
    opt[i + 1][j]; k++)
9e9           if (tab[i][k - 1] + tab[k + 1][j] + c(i, j) < tab
    [i][j])
f95           {
680             tab[i][j] = tab[i][k - 1] + tab[k + 1][j] + c(i
    , j);
14d             opt[i][j] = k;
cbb           }
cbb       }
d41
ea7     printf("%lld\n", tab[1][n]);
cbb   }
cbb }
Full file hash: 17b7c8
```

5.5 MOs

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
5c7   Mo's Algorithm:
882     Solve Q interval queries on a sequence of N values
    offline
826     in O(N * sqrt(Q) * max(insertion time, removal time)).
d41
b95   Usage:
adc     Queries are defined by closed intervals
9b7       [l, r] (1 <= l <= r <= n).
3a9     add(i) must add i-th element to your data structure
802       (1 <= i <= n).
c8d     remove(i) must remove the i-th element (1 <= i <= n).
751     output(id) should answer query with given id using
    current
27a     state.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
806 struct query {
425   int l, r, id;
214 };
d41
044 template<class F>
e5b void mos(int n, vector<query> q, const F &add, const F &
    remove, const F &output)
f95 {
243   int bsize = 1 + n / sqrt(sz(q));
c33   sort(q.begin(), q.end(), [&](const query &lhs, const
    query &rhs) {
6b7         if (lhs.l / bsize != rhs.l / bsize)
4d6           return lhs.l < rhs.l;
5d9         if ((lhs.l / bsize) & 1)
ee7           return (lhs.r > rhs.r);
20f         return (lhs.r < rhs.r);
c0c   });
d41
00a   int l = 1, r = 0; // int l = 0, r = -1; (if indices
    starts at 0)
a0a   for (int i = 0; i < sz(q); i++)
f95   {
dc4     while (l > q[i].l)
194       add(--l);
583     while (r < q[i].r)
3dc       add(++r);
4fd     while (l < q[i].l)
87e       remove(l++);
```

```
5dc     while (r > q[i].r)
8b6       remove(r--);
d41
2d8     output(q[i].id);
cbb   }
cbb }
Full file hash: 0cbc87
```

5.6 MOs - Tree (Edge Query)

```
a34 #include "../../graph/lca/lca.cpp"
d41
d41 /*
741   Refer to the vertex queries option for more info.
aef   This version is different since it is made to handle queries on
bab   the cost of the edges in a path.
d41
a76   To do that, transfer the cost to the vertex down in the rooted
db6   tree and use this.
d41
89d   Remember to use a valid value for the root (even if it never will
c00   be in a query).
08e   Also remember some queries will be empty.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
fa0 struct query
f95 {
6ac   int l, r, id, lc;
214 };
d41
044 template <class F>
23d void mos_tree(int root, int n, vector<pii> pq, vector<int> graph[], const F &add, const F &remove, const F &output)
f95 {
ba2   int a, b;
32e   lca_preprocess lca(root, n, graph);
d41
ad3   vector<int> st(n + 1, 0), en(n + 1, 0), v(2 * n + 3, 0), cnt(n + 1, 0), s;
04c   int id = 0;
45f   s.push_back(root);
365   while (!s.empty())
f95   {
2ec     a = s.back();
342     s.pop_back();
d41
f4f     if (st[a])
2e4       v[en[a] = ++id] = a;
295     else
f95     {
bab       v[st[a] = ++id] = a;
bcc       s.push_back(a);
142       for (int i = 0; i < sz(graph[a]); i++)
9d7         if (graph[a][i] != lca.p[0][a])
bbd           s.push_back(graph[a][i]);
cbb     }
cbb   }
d41
400   vector<query> q;
d41
160   for (int i = 0; i < sz(pq); i++)
f95   {
5ca     tie(a, b) = pq[i];
d41
a91     if (st[a] > st[b])
257       swap(a, b);
3f0     int y = lca.lca(a, b);
84a     if (a == y)
173       q.push_back({st[a], st[b], i, st[y]});
295     else
e65       q.push_back({en[a], st[b], i, -1});
cbb   }
d41
7dd   int bsize = 1 + (2 * n) / sqrt(sz(q));
c33   sort(q.begin(), q.end(), [&](const query &lhs, const query &rhs) {
6b7     if (lhs.l / bsize != rhs.l / bsize)
712       return (lhs.l / bsize < rhs.l / bsize);
1e4     return lhs.r < rhs.r;
c0c   });
d41
568   auto consider = [&](int i) {
e8e     cnt[v[i]]++;
930     if (cnt[v[i]] % 2 == 1)
9a7       add(v[i]);
295     else
8e6       remove(v[i]);
214   };
d41
00a   int l = 1, r = 0;
a0a   for (int i = 0; i < sz(q); i++)
f95   {
dc4     while (l > q[i].l)
47b       consider(--l);
583     while (r < q[i].r)
04a       consider(++r);
4fd     while (l < q[i].l)
b33       consider(l++);
5dc     while (r > q[i].r)
b1b       consider(r--);
d41
4e8     if (q[i].lc != -1) // Remove LCA weight if necessary.
1df       consider(q[i].lc);
d41
2d8     output(q[i].id);
d41
4e8     if (q[i].lc != -1)
1df       consider(q[i].lc);
cbb   }
cbb }
Full file hash: e7d7ff
```

5.7 MOs - Tree (Vertex Query)

```
a34 #include "../../graph/lca/lca.cpp"
d41
d41 /*
31b   Mo's Algorithm on trees:
343     Solve Q path queries in a tree of N vertices
826     in O(N * sqrt(Q) * max(insertion time, removal time)).
9f3     Queries should be on values associated to the tree vertices.
d41
b95   Usage:
281     Pass to the function a tree root, the number of vertices (n),
632     a list of queries (pq) with both ends of each path, the graph
6f2     and functions add, remove and output, such that:
b08       add(i) must add the vertex labeled i to your data
a9f       structure (1 <= i <= n).
377       remove(i) must remove the vertex labeled i (1 <= i <= n).
c02       output(i) should answer query pq[i] using current state.
d41
aba     This will guarantee that when answering the i-th query, only
4f0     the vertices on the desired path are currently in your data
9f3     structure.
d41
c73   Runs in 1s for 10^5 vertices and queries on CF.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
fa0 struct query
f95 {
6ac   int l, r, id, lc;
214 };
d41
044 template <class F>
23d void mos_tree(int root, int n, vector<pii> pq, vector<int> graph[], const F &add, const F &remove, const F &output)
f95 {
ba2   int a, b;
32e   lca_preprocess lca(root, n, graph);
d41
ad3   vector<int> st(n + 1, 0), en(n + 1, 0), v(2 * n + 3, 0), cnt(n + 1, 0), s;
04c   int id = 0;
45f   s.push_back(root);
365   while (!s.empty()) // dfs pre-pos ordering.
f95   {
2ec     a = s.back();
342     s.pop_back();
d41
f4f     if (st[a])
2e4       v[en[a] = ++id] = a;
295     else
f95     {
bab       v[st[a] = ++id] = a;
bcc       s.push_back(a);
142       for (int i = 0; i < sz(graph[a]); i++)
9d7         if (graph[a][i] != lca.p[0][a])
bbd           s.push_back(graph[a][i]);
cbb     }
cbb   }
d41
400   vector<query> q;
d41
160   for (int i = 0; i < sz(pq); i++)
f95   {
5ca     tie(a, b) = pq[i];
d41
a91     if (st[a] > st[b])
257       swap(a, b);
3f0     int y = lca.lca(a, b);
84a     if (a == y)
97b       q.push_back({st[a], st[b], i, -1});
295     else
6a0       q.push_back({en[a], st[b], i, st[y]});
d41       // For queries of this type, the lca must be separately
d41       // added.
cbb   }
d41
7dd   int bsize = 1 + (2 * n) / sqrt(sz(q));
c33   sort(q.begin(), q.end(), [&](const query &lhs, const query &rhs) {
6b7     if (lhs.l / bsize != rhs.l / bsize)
712       return (lhs.l / bsize < rhs.l / bsize);
1e4     return lhs.r < rhs.r;
c0c   });
```

```
d41
d41   // Vertices inserted twice are removed.
568   auto consider = [&](int i) {
e8e     cnt[v[i]]++;
930     if (cnt[v[i]] % 2 == 1)
9a7       add(v[i]);
295     else
8e6       remove(v[i]);
214   };
d41
00a   int l = 1, r = 0;
a0a   for (int i = 0; i < sz(q); i++)
f95   {
dc4     while (l > q[i].l)
47b       consider(--l);
583     while (r < q[i].r)
04a       consider(++r);
4fd     while (l < q[i].l)
b33       consider(l++);
5dc     while (r > q[i].r)
b1b       consider(r--);
d41
4e8     if (q[i].lc != -1)
1df       consider(q[i].lc);
d41
2d8     output(q[i].id);
d41
4e8     if (q[i].lc != -1)
1df       consider(q[i].lc);
cbb   }
cbb }
Full file hash: b5d4a0
```

5.8 MOs - Hilbert

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
bd0   MO's using Hilbert Curve to sort the queries.
91d     O(N * sqrt(Q) * max(insertion time, removal time)).
d41
97c     Applicability:
223     When Q is significantly less than N, it works much
     faster than
f49   the classical version.
d41
b95     Usage:
02f     Same as the classical version, but use the query's
     hilbertorder
6fc   as comparator
c4c */
d41
547 constexpr int logn = 20;
a31 constexpr int maxn = 1 << logn;
10e ll hilbertorder(int x, int y)
f95 {
b72   ll d = 0;
cd0   for (int s = 1 << (logn - 1); s; s >>= 1)
f95   {
0e1     bool rx = x & s, ry = y & s;
9dc     d = d << 2 | rx * 3 ^ static_cast<int>(ry);
6b4     if (!ry)
f95     {
5f9       if (rx)
f95       {
e0b         x = maxn - x;
617         y = maxn - y;
cbb       }
9dd       swap(x, y);
cbb     }
cbb   }
be2   return d;
cbb }
d41
806 struct query {
425   int l, r, id;
1e3     ll ord() const
f95     {
825         return hilbertorder(l, r);
cbb     }
214 };
d41
044 template<class F>
e5b void mos(int n, vector<query> q, const F &add, const F &
     remove, const F &output)
f95 {
243   int bsize = 1 + n / sqrt(sz(q));
c33   sort(q.begin(), q.end(), [&](const query &lhs, const
     query &rhs) {
7df         return lhs.ord() < rhs.ord();
c0c   });
d41
00a   int l = 1, r = 0; // int l = 0, r = -1; (if indices
     starts at 0)
a0a   for (int i = 0; i < sz(q); i++)
f95   {
dc4     while (l > q[i].l)
194       add(--l);
583     while (r < q[i].r)
3dc       add(++r);
4fd     while (l < q[i].l)
87e       remove(l++);
5dc     while (r > q[i].r)
8b6       remove(r--);
d41
2d8     output(q[i].id);
cbb   }
cbb }
Full file hash: 4195b8
```

5.9 Ternary Search (continuous)

```
d41 /*
0a2   Ternary Search:
550     Finds x such that f(x) is minimum in range [bot, top]
     in
387     O(lg((top - bot) / eps)).
6f0     Value is correct within the specified precision eps.
d41
ca2   Constraints:
d37     f(x) is strictly decreasing for some interval [bot, x1
     ],
28c     constant in an interval [x1, x2]
564     and strictly increasing in a interval [x2, top]. x1 <=
     x2 are
e4f     arbitrary values where [x1, x2] is a plateau of optimal
862     solutions.
d41
b95   Usage:
5b6     Call the function passing a lambda expression or
     function f.
33c     If there are multiple possible solutions, assume that
     an
662     arbitrary one in the plateau is returned.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
398 template <typename F>
ca6 double ternary_search(const F &f, double bot = -1e9, double
      top = 1e9, double eps = 1e-9)
f95 {
14d   while (top - bot > eps)
f95   {
8e3     double x1 = (0.55*bot + 0.45*top); // (2*bot + top) / 3
     is
d41                       // more stable, but slower.
3f8     double x2 = (0.45*bot + 0.55*top);
948     if (f(x1) > f(x2))
443       bot = x1;
295     else
16b       top = x2;
cbb   }
d41
05e   return (bot + top) / 2;
cbb }
Full file hash: 082811
```

5.10 Ternary Search (discrete)

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
0a2   Ternary Search:
5d5     Finds the smallest x in range [bot, top] such that f(x)
     is
792     maximum in O(lg(top - bot)).
d41
ca2   Constraints:
de7     f(x) is strictly increasing for some interval [bot, x1
     ],
ffa     constant in an interval [x1, x2] and strictly
     decreasing in a
bc6     interval [x2, top]. x1 <= x2 are arbitrary values where
a05     [x1, x2] is a plateau of optimal solutions.
d41
b95   Usage:
5b6     Call the function passing a lambda expression or
     function f.
d41
399   Source: modified from https://github.com/
650   kth-competitive-programming/kactl/blob/master/content/
36d   various/TernarySearch.h
c4c */
d41
398 template <typename F>
2ba int ternary_search(const F &f, int bot, int top)
f95 {
03c   while (top - bot >= 5)
f95   {
5ad     int mid = (bot + top) / 2;
2ef     if (f(mid) < f(mid + 1))
6a2       bot = mid;
295     else
2b8       top = mid + 1;
cbb   }
d41
c6f   for (int i = bot + 1; i <= top; i++)
773     if (f(i) > f(bot))
8a7       bot = i;
d41
6ff   return bot;
cbb }
Full file hash: 5b5ba5
```

6 Combinatorial

6.1 Binomial - Pascal's Triangle

```
d41
be6 #include "../../../contest/header.hpp"
d41 /*
8c1   [DESCRIPTION]
798    Pre-computing all binomial coefficient (% MOD) up to
1ae      C(MAXN, MAXN) into a matrix using pascal's triangle.
d41
ff5   [COMPLEXITY]
544    O(n^2) to Pre-computing
6d7      O(1) to lookup
c4c  */
d41
fd8 #define MAXN 3123
39b #define MOD 1000000007
f93 long long C[MAXN][MAXN];
d41
51e void init_ncr()
f95 {
700     C[0][0] = 1;
a04     for (int n = 1; n < MAXN; ++n) {
608         C[n][0] = C[n][n] = 1;
fe8         for (int k = 1; k < n; ++k)
064             C[n][k] = (C[n - 1][k - 1] + C[n - 1][k]) % MOD
    ;
cbb     }
cbb }
Full file hash: 88d086
```

6.2 Binomial - Lucas' Theorem

```
be6 #include "../../../contest/header.hpp"
b92 #include "../../../number_theory/mod_inverse/mod_inverse.
    cpp"
d41 /*
8c1   [DESCRIPTION]
392    Lucas' theorem to calculate C(N, M) % P where N, M be
492    non-negative integers and P a prime.
d41
9af      Write N and M in the base P:
102    Np= n_kp^k +...+ n_1p + n_0 and Mp = m_kp^k +...+ m_1p +
     m_0.
0e8      Then C(N, M) === Prod(C(n_i, m_i)) % P
d41
470   [CONSEQUENCE]
da4    A binomial coefficient C(N, M) is divisible by a prime P
     iff
fea    there is an index i where
4cf    Np[i] < Mp[i] which leads to C(Nb[i], Mb[i]) = 0.
14d    Hence, C(N, M) % P = 0
d41
bd7   [USAGE]
ed7    Pre-compute all factorials (mod P) up to the P prime
    chosen.
904    all the function choseModP.
982    You can also pre-compute all factorials' modular
    inverses to
fd6    boost the performance.
d41
ff5   [COMPLEXITY]
c4d    O(log_p(N) * mod_inverse())
c4c  */
d41
2f5 ll chooseModP(ll n, ll m, int p, vector<ll> &fact)
f95 {
386   ll c = 1;
bce   while (n || m)
f95   {
5ec     ll a = n % p, b = m % p;
545     if (a < b)
bb3       return 0;
9f3     c = c * fact[a] % p * mod_inverse<ll>(fact[b], p) % p *
     mod_inverse<ll>(fact[a - b], p) % p;
f4a     n /= p;
b49     m /= p;
cbb   }
807   return c;
Full file hash: b91f81
```

6.3 Grundy

```
d41
5d1 #include "../../contest/header.hpp"
d41
d41 /*
e10   Mex for Grundy Number O(N):
aaf     To calculate the Grundy Number for a set of states,
    first
760     set terminal states' Grundy Number (zero if no move is
    a
da2     loss condition).
392     Then, for each state, find the MEX for the Grundy
    Numbers of
7b0     reacheable states (i.e. the lowest Grundy Number not
586     present).
d1d     This will be the current state's Grundy Number.
d95     If you have many parallel games, you can find the
    equivalent
54f     Grundy number by doing XOR of individual Grundy Numbers
    .
777     0 equals losing position, any other value is a winning
c72     position.
d41
ca2   Constraints:
c91     The game must be symmetrical (same moves are available
    for
4b3     both players); have perfect information (no hidden or
838     random stuff);
04a     be finite (no loops).
d41
b95   Usage:
90c     For each state, save the Grundy Number of all reacheble
6be     states in a vector v, and pass as argument to mex().
d41
d41
2c0   Source: my head
c4c */
d41
b78 int mex(vector<int> v){
d41   //Place every value in the position with same index (if
    possible
d41   //and it's not already there)
f14   for(int i = 0; i < v.size(); i++){
bd3     while(v[i] < v.size() && v[v[i]] != v[i])
dda       swap(v[v[i]], v[i]);
cbb   }
d41   //Verify the first missing number
c6e   for(int i = 0; i < v.size(); i++)
fce     if(v[i] != i)
d9a       return i;
d41
5f4   return v.size();
cbb }
Full file hash: 6fe471
```

6.4 Surreal Numbers

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
3fb   General Theory for Two Player Game
d41
431   Take a perfect information game involving two players,
5d5   Left and Right, where either one can start the game.
d41
1dd   This is a partial game in which the allowable moves
5d0   depend on which of the two players is currently moving
6b0   (e.g chess).
d41
c0d   There are four possible scenarios for an initial
    configuration:
81b     - Left wins (does not matter if first or second).
494     - Right wins
f36     - First player wins.
484     - Second player wins
d41
305   If a game has no "First player wins" configurations,
944   the configurations of the game can be mapped to real
295   numbers s(i) of the form a/2^b such that.
585     - s(i) > 0 -> Left wins
fc5     - s(i) < 0 -> Right wins
b10     - s(i) = 0 -> second player wins
d41
450   The union of two states i,j is mapped to s(i) + s(j)
d41
222   More formally:
d62   A game is a position in a contest between two players,
    Left and
f2a   Right.
500   Each player has a set of games called options to choose
    from in
b9d   turn.
2b0   Games are written {L|R} where L is the set of Left's
    options and
cf5   R is the set of Right's options.
d41
532   At the start there are no games at all, so the empty set
    is the
6a6   only set of options we can provide to the players.
1fa   This defines the game {|}, which is called 0. We consider
     a
212   player who must play a turn but has no options to have
    lost the
8e4   game (so in game 0 second player wins). Given this game 0
     there
59f   are now two possible sets of options, the empty set and
    the set
c31   whose only element is zero.
e06   The game {0|} is called 1, and the game {|0} is called
    -1.
d6f   In game 1, if Right goes first, Left wins. And if Left
    goes
765   first, he will choose game 0 to be next and win (because
    Right
4c2   will have no moves).
1cd   So game 1 is of the type Left wins, as expected.
6e4   The game {0|0} is called * (star), and is the first game
    we find
d31   that is not a number (in this game, first player wins).
d41
422   All numbers are positive, negative, or zero, and we say
    that a
079   game is positive if Left will win, negative if Right will
     win,
53c   or zero if the second player will win. Games that are not
```

```
     numbers
bde   have a fourth possibility: they may be fuzzy, meaning
    that the
4f5   first player will win. * is a fuzzy game.[4]
c4c */
d41
13a int main(void)
f95 {
b3f   cin.sync_with_stdio(0);
b95   cin.tie(0);
1a8   int n;
ba5   while (cin >> n)
f95   {
55c     ll mult = (1ll << 40);
d41     // This will store the surreal number for each game
    times
d41     // 2^40 (just to avoid doubles).
75f     vector<ll> s(n);
fb8     vector<int> val(n);
d41
830     for (int i = 0; i < n; i++)
f95     {
905       string a;
964       cin >> a;
e4c       ll x = mult;
0f6       bool change = false;
784       for (int j = 0; j < sz(a); j++)
f95       {
194         if (a[j] != a[0]) // After first different, start
d41                   // changing x.
981           change = true;
d41
b92         if (change)
4fe           x /= 2;
d41
525         if (a[j] == 'B')
910           s[i] += x;
295         else
6b1           s[i] -= x;
cbb       }
d41
7e7       val[i] = sz(a);
cbb     }
d41
d41     // Now we have s[i] for each game.
d41     // If we join two games i,j we get a game x with
d41     // s(x) = s[i] + s[j] and val(x) = val[i] + val[i].
d41     // So to find a fair game with s[x] = 0 and maximum val
d41     // We need to find a subset with zero sum and maximum
    val.
d41
d41     // Here onwards we just solve this problem with meet in
d41     // the middle.
d41
60d     unordered_map<ll, int> tab;
904     int mid = (n + 1) / 2;
d09     for (int i = 0; i < (1 << mid); i++)
f95     {
00d       ll x = 0;
39d       int y = 0;
7a6       for (int j = 0; j < mid; j++)
8aa         if (i & (1 << j))
f95         {
43e           x += s[j];
ab0           y += val[j];
cbb         }
d41
0f0       tab[x] = max(tab[x], y);
cbb     }
d41
1a4     int ans = 0;
d41
611     for (int i = 0; i < (1 << (n - mid)); i++)
f95     {
00d       ll x = 0;
39d       int y = 0;
6cc       for (int j = 0; j < (n - mid); j++)
8aa         if (i & (1 << j))
f95         {
b7e           x += s[mid + j];
7d4           y += val[mid + j];
cbb         }
d41
f85       auto it = tab.find(-x);
e1c       if (it != tab.end())
f95       {
fb7         ans = max(ans, y + it->second);
cbb       }
cbb     }
d41
886     cout << ans << endl;
cbb   }
cbb }
Full file hash: 767d32
```

7 Number Theory

7.1 General Chinese Remainder Theorem

```
5d1 #include "../../contest/header.hpp"
771 #include "../euclid/euclid.cpp"
d41
d41 /*
8c1     [DESCRIPTION]
055     Returns a number x % lcm(m,n), such that:
096         x === a (mod m)
fde         x === b (mod n)
d41
9c7     returns -1 if there is no solution
d41
ff5     [COMPLEXITY]
9a3     log(n)
d41
c1f     [CONSTRAINTS]
832         LCM(m, n) should fit in a long long variable.
c4c */
d41
5d1 ll crt(ll a, ll m, ll b, ll n)
f95 {
e6a     if (n > m)
134         swap(a, b), swap(m, n);
3e2     ll x, y, g = gcd<ll>(m, n, x, y);
f08     if ((a - b) % g != 0)
daa         return -1;
ef8     x = (b - a) % n * x % n / g * m + a;
16c     return x < 0 ? x + m * n / g : x;
cbb }
Full file hash: 261c61
```

7.2 General Chinese Remainder Theorem - System

```
5d1 #include "../../contest/header.hpp"
5c6 #include "crt.cpp"
d41 /*
8c1     [DESCRIPTION]
4c1     Returns a integer a number x, such that:
773         x === a[0] (mod m[0])
726         x === a[1] (mod m[1])
2f4         ...
f8e         x === a[n - 1] (mod m[n - 1])
d41
06b     The m[] set does not need to be only of coprimes,
    because it uses
e63     the generalized version of CRT.
d41
bd7     [USAGE]
8a1     Just pass the arrays as shown above in the description
    and their
902     size.
deb     It's 0-indexed, but its trivial to change it to 1-
    indexed.
d41
8a9     [RESULT]
336     The function returns x % LCM(m[0], m[1], ..., m[n -
    1]) if a
2d9     answer exists. Otherwise it returns -1.
d41
ff5     [COMPLEXITY]
a28     O( n * log(LCM(m)) )
d41
c1f     [CONSTRAINTS]
6a0         LCM(m[0], m[1], ..., m[n - 1]) should fit in a long
    long
65b         variable.
e2e         The values of a[] can be arbitrary, because they
    are
806         normalized inside the function
d41
f55     source: https://codeforces.com/blog/entry/61290
d41
c4c */
d41
815 ll crt_system(ll a[], ll m[], int n)
f95 {
d41     // normalize
830     for (int i = 0; i < n; i++)
e05         a[i] = (a[i] % m[i] + m[i]) % m[i];
d41
a78     ll ans = a[0];
20b     ll lcm = m[0];
aa4     for (int i = 1; i < n; i++)
f95     {
173         ans = crt(ans, lcm, a[i], m[i]);
787         if (ans == -1)
daa             return -1;
0be         ll x, y;
e2e         ll d = gcd<ll>(lcm, m[i], x, y);
930         lcm = lcm * m[i] / d;
cbb     }
ba7     return ans;
Full file hash: dd8682
```

7.3 Euclid

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
4b5   Extended Euclidean Algorithm:
71f     Returns the gcd of a and b.
49d     Also finds numbers x and y for which a * x + b * y =
    gcd(a, b)
f41     (not unique).
cd0     All pairs can be represented in the form
c42       (x + k * b / gcd, y - k * a / gcd)
```

```
76e     for k an arbitrary integer.
829     If there are several such x and y, the function returns
    the
e39     pair for which |x| + |y| is minimal.
232     If there are several x and y satisfying the minimal
    criteria,
2ed     it outputs the pair for which X <= Y.
d41
399   Source: modified from https://cp-algorithms.com/algebra/
df5               extended-euclid-algorithm.html
d41
b95   Usage:
647     For non-extendend version, c++ has __gcd and __lcm.
d41
ca2   Constraints:
30a     Produces correct results for negative integers as well.
c4c */
d41
4fc template<class T>
946 T gcd(T a, T b, T &x, T &y)
f95 {
fcb   if (b == 0)
f95   {
483     x = 1;
01d     y = 0;
3f5     return a;
cbb   }
d41
328   T x1, y1;
254   T d = gcd(b, a % b, x1, y1);
711   x = y1;
a2a   y = x1 - y1 * (a / b);
be2   return d;
cbb }
Full file hash: 0c35ae
```

7.4 Factorization (Pollard rho)

```
d41 /*
d00   Description:
b75     Pollard-rho randomized factorization algorithm. Returns
    prime
937   factors of a number, in arbitrary order (e.g. 2299 ->
fbf                         {11, 19, 11}).
d41
421   Time:
018     O(n^1/4) gcd calls, less for numbers with small factors
    .
d41
1d1   Source: https://github.com/kth-competitive-programming/
c4c  */
d41
5d1 #include "../../contest/header.hpp"
09d #include "../primality_test/millerRabin.cpp"
d41
97d ull pollard(ull n)
f95 {
4de   auto f = [n](ull x) { return (mod_mul(x, x, n) + 1) % n;
    };
6b3   if (!(n & 1))
18b     return 2;
8c3   for (ull i = 2;; i++)
f95   {
e17     ull x = i, y = f(x), p;
332     while ((p = __gcd(n + y - x, n)) == 1)
b78       x = f(x), y = f(f(y));
940     if (p != n)
74e       return p;
cbb   }
cbb }
d41
2de vector<ull> factorize(ull n)
f95 {
e7f   if (n == 1)
21d     return {};
121   if (isPrime(n))
48e     return {n};
bc6   ull x = pollard(n);
b3b   auto l = factorize(x), r = factorize(n / x);
7af   l.insert(l.end(), all(r));
792   return l;
cbb }
Full file hash: 66d5a6
```

7.5 Modular Inverse

```
771 #include "../euclid/euclid.cpp"
d41
d41 /*
18a   Modular Inverse:
76e     Returns an integer x such that (a * x) % m == 1.
715     The modular inverse exists if and only if a and m are
8dc     relatively prime.
ff1     Modular inverse is also equal to a^(phi(m) - 1) % m.
eb3     In particular, if m is prime a^(-1) == a^(m-2), which
    might be
6bc     faster to code.
d41
399   Source: modified from https://cp-algorithms.com/algebra/
f41               module-inverse.html
c4c */
d41
4fc template<class T>
b26 T mod_inverse(T a, T m)
f95 {
645   T x, y;
655   assert(gcd(a, m, x, y) == 1); // Or return something, if
    gcd is
d41                 // not 1 the inverse doesn't exist.
08f   return (x % m + m) % m;
cbb }
Full file hash: 7efa11
```

7.6 Large modular mult/pow

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
d00     Description:
b31     Calculate a * b mod c (or a^b mod c)
002     for 0 <= a, b < c < 2^63.
421     Time:
666     mod_mul: O (1)
19c     mod_pow: (log b)
1d1   Source: https://github.com/kth-competitive-programming/
c4c */
d41
119 ull mod_mul(ull a, ull b, ull M)
f95 {
053   ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
964   return ret + M * (ret < 0) - M * (ret >= (ll)M);
cbb }
b40 ull mod_pow(ull b, ull e, ull mod)
f95 {
c1a   ull ans = 1;
4d1   for (; e; b = mod_mul(b, b, mod), e /= 2)
654     if (e & 1)
69f       ans = mod_mul(ans, b, mod);
ba7   return ans;
Full file hash: ebdfbd
```

7.7 Modular Arithmetic

```
90f #include "../mod_inverse/mod_inverse.cpp"
d41
d41 /*
d0c   Modular Arithmetic:
7bc     Struct wrapper on to of modular arithmetics.
d41
399   Source: modified from https://github.com/
650   kth-competitive-programming/kactl/blob/master/content/
591   number-theory/ModularArithmetic.h
c4c */
d41
31e template <ll mod>
072 struct mod_num
f95 {
4ad   ll x;
de8   explicit mod_num(ll x = 0) : x(x % mod) {}
f76   mod_num operator+(mod_num b) { return mod_num(x + b.x); }
b1d   mod_num operator-(mod_num b) { return mod_num(x - b.x +
    mod); }
b2f   mod_num operator*(mod_num b) { return mod_num(x * b.x); }
09d   mod_num operator/(mod_num b) { return mod_num(x *
    mod_inverse(b.x, mod)); }
583   mod_num operator^(ll e)
f95   {
972     mod_num ans(1);
6d7     mod_num b = *this;
25d     for (; e; b = b * b, e /= 2)
654       if (e & 1)
bfb         ans = ans * b;
ba7     return ans;
cbb   }
d41
6dc   void operator+=(mod_num b) { x = (x + b.x) % mod; }
214 };
d41
31e template <ll mod>
58e ostream &operator<<(ostream &os, mod_num<mod> x)
f95 {
55e   return os << x.x;
cbb }
Full file hash: f151a0
```

7.8 Phi

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
bf2   Euler's totient function (PHI):
fab     Euler's totient function, also known as phi-function
    PHI(n),
f44     counts the number of integers between 1 and n inclusive
    , which
8d1     are coprime to n. Two numbers are coprime if their
    greatest
d6a     common divisor equals 1 (1 is considered to be coprime
    to any
83f     number).
d41
d41
399   Source: modified from https://cp-algorithms.com/algebra/
356               phi-function.html
e1f     and https://github.com/kth-competitive-programming/
    kactl/blob/
a57     master/content/number-theory/phiFunction.h
d41
```

```
b95   Usage:
a1b     Some useful properties:
9f5     - If p is a prime number, PHI(p)=p-1.
d4d     - If a and b are relatively prime, PHI(ab)=PHI(a)*PHI(b
    ).
343     - In general, for not coprime a and b,
045         PHI(ab)=PHI(a)*PHI(b)*d/PHI(d), with d=gcd(a,b)
    holds.
417     - PHI(PHI(m)) <= m / 2
25b     - Euler's theorem: a^PHI(m) === 1 (mod m), for a and m
    coprime
bff     - For a and m coprime: a^n === a^(n % PHI(m)) (mod m)
279     - For arbitrary x,m and n >= log_2(m):
131       x^n === x^(PHI(m)+[n % PHI(m)]) (mod m)
e1e     The one above allows computing modular exponentiation
    for
6e5     really large exponents.
565     - If d is a divisor of n, then there are phi(n/d)
    numbers
837       i <= n for which gcd(i,n)=d
137     - sum_{d|n} phi(d) = n
c22     - sum_{1 <= k <= n, gcd(k,n)=1} k = n * phi(n) / 2, for
     n > 1
c4c */
d41
d41 // Use this one for few values of phi.
b5f int phi(int n)
f95 {
efa   int result = n;
83f   for (int i = 2; i * i <= n; i++)
f95   {
775     if (n % i == 0)
f95     {
49e       while (n % i == 0)
135         n /= i;
21c       result -= result / i;
cbb     }
cbb   }
f3d   if (n > 1)
e48     result -= result / n;
dc8   return result;
cbb }
d41
4fe namespace totient
f95 {
2d6 const int MAXV = 1000001; // Takes ~0.03 s for 10^6.
6e5 int phi[MAXV];
d41
b2a void init()
f95 {
948   for (int i = 0; i < MAXV; i++)
ed1     phi[i] = i & 1 ? i : i / 2;
9be   for (int i = 3; i < MAXV; i += 2)
a22     if (phi[i] == i)
8a4       for (int j = i; j < MAXV; j += i)
a9b         phi[j] -= phi[j] / i;
cbb }
cbb } // namespace totient
Full file hash: e79764
```

7.9 Primality Test

```
d41 /*
d00     Description:
37d         Deterministic Miller-Rabin primality test.
334         Guaranteed to work for numbers up to 2^64 (for
    larger
824         numbers, extend A randomly).
d41
421     Time:
671         7 * O(log b)
d41
1d1     Source: https://github.com/kth-competitive-programming
    /
c4c */
d41
5d1 #include "../../contest/header.hpp"
cfa #include "../mod_mul/mod_mul.cpp"
d41
91e bool isPrime(ull n)
f95 {
e00     if (n < 2 || n % 6 % 4 != 1)
d15         return n - 2 < 2;
43a     ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022},
c17         s = __builtin_ctzll(n - 1), d = n >> s;
56e     for (auto &a : A)
f95     { // ^ count trailing zeroes
8a8         ull p = mod_pow(a, d, n), i = s;
274         while (p != 1 && p != n - 1 && a % n && i--)
2cb             p = mod_mul(p, p, n);
5fd         if (p != n - 1 && i != s)
bb3             return 0;
cbb     }
6a5     return 1;
cbb }
Full file hash: df75c7
```

7.10 Sieve

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
d5c   Sieve of Eratosthenes:
0a5     Finds all primes in interval [2, MAXP] in O(MAXP) time.
913     Also finds lp[i] for every i in [2, MAXP], such that lp
    [i] is
0aa     the minimum prime factor of i.
6db     Particularly useful for factorization.
d41
399   Source: modified from https://cp-algorithms.com/algebra/
874               prime-sieve-linear.html
d41
b95   Usage:
929     Set MAXP and call init.
852     Sieve for 10^7 should run in about 0.2 s.
c4c */
d41
2ca namespace sieve
f95 {
bac const int MAXP = 10000000; // Will find primes in interval
    [2, MAXP].
39b int lp[MAXP + 1]; // lp[i] is the minimum prime factor of i
    .
632 vector<int> p; // Ordered list of primes up to MAXP.
d41
b2a void init()
f95 {
008   for (int i = 2; i <= MAXP; i++)
f95   {
d4a     if (lp[i] == 0)
b6f       p.push_back(lp[i] = i);
d41
9d8     for (int j = 0; j < (int)p.size() && p[j] <= lp[i] && i
     * p[j] <= MAXP; j++)
fb7       lp[i * p[j]] = p[j];
cbb   }
cbb }
cbb } // namespace sieve
Full file hash: e9076d
```

8 Numerical

8.1 Big Int

```
5d1 #include "../../contest/header.hpp"
d41
d41 // This code is not meant to be written in icpc contests.
d41 // This is just here to fill a void for now.
d41 // Source: someone on CF
d41
d41 // NOTE:
d41 // This code contains various bug fixes compared to the
    original
d41 // version from
d41 // indy256 (github.com/indy256/codelibrary/blob/master/cpp/
    numbertheory/bigint-full.cpp),
d41 // including:
d41 // - Fix overflow bug in mul_karatsuba.
d41 // - Fix overflow bug in fft.
d41 // - Fix bug in initialization from long long.
d41 // - Optimized operators + - *.
d41 //
d41 // Tested:
d41 // - https://www.e-olymp.com/en/problems/266: Comparison
d41 // - https://www.e-olymp.com/en/problems/267: Subtraction
d41 // - https://www.e-olymp.com/en/problems/271:
    Multiplication
d41 // - https://www.e-olymp.com/en/problems/272:
    Multiplication
d41 // - https://www.e-olymp.com/en/problems/313: Addition
d41 // - https://www.e-olymp.com/en/problems/314: Addition/
    Subtraction
d41 // - https://www.e-olymp.com/en/problems/317:
    Multiplication (simple / karatsuba / fft)
d41 // - https://www.e-olymp.com/en/problems/1327:
    Multiplication
d41 // - https://www.e-olymp.com/en/problems/1328
d41 // - VOJ BIGNUM: Addition, Subtraction, Multiplication.
d41 // - SGU 111: sqrt
d41 // - SGU 193
d41 // - SPOJ MUL, VFMUL: Multiplication.
d41 // - SPOJ FDIV, VFDIV: Division.
d41
d73 const int BASE_DIGITS = 9;
82e const int BASE = 1000000000;
d41
6ac struct BigInt {
d65     int sign;
a9d     vector<int> a;
d41
d41     // -------------------- Constructors
    --------------------
d41     // Default constructor.
1ac     BigInt() : sign(1) {}
d41
d41     // Constructor from long long.
ccf     BigInt(long long v) {
324         *this = v;
cbb     }
235     BigInt& operator = (long long v) {
ce6         sign = 1;
ea2         if (v < 0) {
6a7             sign = -1;
6fa             v = -v;
cbb         }
228         a.clear();
fef         for (; v > 0; v = v / BASE)
c23             a.push_back(v % BASE);
357         return *this;
```

```
cbb     }
d41
d41     // Initialize from string.
c71     BigInt(const string& s) {
e65         read(s);
cbb     }
d41
d41     // -------------------- Input / Output
    --------------------
6c3     void read(const string& s) {
ce6         sign = 1;
228         a.clear();
bec         int pos = 0;
a68         while (pos < (int) s.size() && (s[pos] == '-' || s[
    pos] == '+')) {
dbe             if (s[pos] == '-')
2b8                 sign = -sign;
17d             ++pos;
cbb         }
795         for (int i = s.size() - 1; i >= pos; i -=
    BASE_DIGITS) {
c67             int x = 0;
d34             for (int j = max(pos, i - BASE_DIGITS + 1); j
    <= i; j++)
cfc                 x = x * 10 + s[j] - '0';
7c6             a.push_back(x);
cbb         }
0eb         trim();
cbb     }
bd2     friend istream& operator>>(istream &stream, BigInt &v)
    {
ac0         string s;
e0c         stream >> s;
c40         v.read(s);
a87         return stream;
cbb     }
d41
446     friend ostream& operator<<(ostream &stream, const
    BigInt &v) {
b5c         if (v.sign == -1 && !v.isZero())
27b             stream << '-';
4fd         stream << (v.a.empty() ? 0 : v.a.back());
fce         for (int i = (int) v.a.size() - 2; i >= 0; --i)
018             stream << setw(BASE_DIGITS) << setfill('0') <<
    v.a[i];
a87         return stream;
cbb     }
d41
d41     // -------------------- Comparison --------------------
701     bool operator<(const BigInt &v) const {
eb9         if (sign != v.sign)
603             return sign < v.sign;
a27         if (a.size() != v.a.size())
f7d             return a.size() * sign < v.a.size() * v.sign;
305         for (int i = ((int) a.size()) - 1; i >= 0; i--)
00d             if (a[i] != v.a[i])
244                 return a[i] * sign < v.a[i] * sign;
d1f         return false;
cbb     }
d41
426     bool operator>(const BigInt &v) const {
54b         return v < *this;
cbb     }
656     bool operator<=(const BigInt &v) const {
0fe         return !(v < *this);
cbb     }
605     bool operator>=(const BigInt &v) const {
d9c         return !(*this < v);
cbb     }
880     bool operator==(const BigInt &v) const {
7f4         return !(*this < v) && !(v < *this);
cbb     }
062     bool operator!=(const BigInt &v) const {
6c5         return *this < v || v < *this;
cbb     }
d41
d41     // Returns:
d41     // 0 if |x| == |y|
d41     // -1 if |x| < |y|
d41     // 1 if |x| > |y|
ce6     friend int __compare_abs(const BigInt& x, const BigInt&
     y) {
e78         if (x.a.size() != y.a.size()) {
c86             return x.a.size() < y.a.size() ? -1 : 1;
cbb         }
d41
a55         for (int i = ((int) x.a.size()) - 1; i >= 0; --i) {
a5b             if (x.a[i] != y.a[i]) {
b1e                 return x.a[i] < y.a[i] ? -1 : 1;
cbb             }
cbb         }
bb3         return 0;
cbb     }
d41
d41     // -------------------- Unary operator - and operators
    +- --------------------
1e3     BigInt operator-() const {
18b         BigInt res = *this;
b96         if (isZero()) return res;
d41
290         res.sign = -sign;
b50         return res;
cbb     }
d41
d41     // Note: sign ignored.
d60     void __internal_add(const BigInt& v) {
f72         if (a.size() < v.a.size()) {
2ce             a.resize(v.a.size(), 0);
cbb         }
1ad         for (int i = 0, carry = 0; i < (int) max(a.size(),
    v.a.size()) || carry; ++i) {
df4             if (i == (int) a.size()) a.push_back(0);
d41
85e             a[i] += carry + (i < (int) v.a.size() ? v.a[i]
    : 0);
49b             carry = a[i] >= BASE;
179             if (carry) a[i] -= BASE;
cbb         }
cbb     }
d41
d41     // Note: sign ignored.
8b4     void __internal_sub(const BigInt& v) {
65c         for (int i = 0, carry = 0; i < (int) v.a.size() ||
    carry; ++i) {
a14             a[i] -= carry + (i < (int) v.a.size() ? v.a[i]
    : 0);
e0b             carry = a[i] < 0;
da5             if (carry) a[i] += BASE;
cbb         }
0e3         this->trim();
cbb     }
d41
89f     BigInt operator += (const BigInt& v) {
8ea         if (sign == v.sign) {
570             __internal_add(v);
9d9         } else {
ae3             if (__compare_abs(*this, v) >= 0) {
e98                 __internal_sub(v);
9d9             } else {
dcc                 BigInt vv = v;
3c5                 swap(*this, vv);
fe0                 __internal_sub(vv);
cbb             }
cbb         }
357         return *this;
cbb     }
d41
6b1     BigInt operator -= (const BigInt& v) {
8ea         if (sign == v.sign) {
ae3             if (__compare_abs(*this, v) >= 0) {
e98                 __internal_sub(v);
9d9             } else {
dcc                 BigInt vv = v;
3c5                 swap(*this, vv);
fe0                 __internal_sub(vv);
0db                 this->sign = -this->sign;
cbb             }
9d9         } else {
570             __internal_add(v);
cbb         }
357         return *this;
cbb     }
d41
d41     // Optimize operators + and - according to
d41     // https://stackoverflow.com/questions/13166079/move-
    semantics-and-pass-by-rvalue-reference-in-overloaded-
    arithmetic
f1e     template< typename L, typename R >
81c         typename std::enable_if<
4ec             std::is_convertible<L, BigInt>::value &&
c0d             std::is_convertible<R, BigInt>::value &&
061             std::is_lvalue_reference<R&&>::value,
6b2             BigInt>::type friend operator + (L&& l, R&& r)
    {
46b         BigInt result(std::forward<L>(l));
fbe         result += r;
dc8         return result;
cbb     }
f1e     template< typename L, typename R >
81c         typename std::enable_if<
4ec             std::is_convertible<L, BigInt>::value &&
c0d             std::is_convertible<R, BigInt>::value &&
bcc             std::is_rvalue_reference<R&&>::value,
6b2             BigInt>::type friend operator + (L&& l, R&& r)
    {
5f0         BigInt result(std::move(r));
a5a         result += l;
dc8         return result;
cbb     }
d41
f1e     template< typename L, typename R >
81c         typename std::enable_if<
4ec             std::is_convertible<L, BigInt>::value &&
6ca             std::is_convertible<R, BigInt>::value,
161             BigInt>::type friend operator - (L&& l, R&& r)
    {
46b         BigInt result(std::forward<L>(l));
1d1         result -= r;
dc8         return result;
cbb     }
d41
d41     // -------------------- Operators * / %
    --------------------
a17     friend pair<BigInt, BigInt> divmod(const BigInt& a1,
    const BigInt& b1) {
872         assert(b1 > 0);  // divmod not well-defined for b <
     0.
d41
25f         long long norm = BASE / (b1.a.back() + 1);
7c4         BigInt a = a1.abs() * norm;
```

```
        BigInt b = b1.abs() * norm;
        BigInt q = 0, r = 0;
        q.a.resize(a.a.size());

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

        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        auto res = make_pair(q, r / norm);
        if (res.second < 0) res.second += b1;
        return res;
    }
    BigInt operator/(const BigInt &v) const {
        return divmod(*this, v).first;
    }

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

    void operator/=(int v) {
        assert(v > 0);  // operator / not well-defined for v <= 0.
        if (llabs(v) >= BASE) {
            *this /= BigInt(v);
            return ;
        }
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
            long long cur = a[i] + rem * (long long) BASE;
            a[i] = (int) (cur / v);
            rem = (int) (cur % v);
        }
        trim();
    }

    BigInt operator/(int v) const {
        assert(v > 0);  // operator / not well-defined for v <= 0.

        if (llabs(v) >= BASE) {
            return *this / BigInt(v);
        }
        BigInt res = *this;
        res /= v;
        return res;
    }
    void operator/=(const BigInt &v) {
        *this = *this / v;
    }

    long long operator%(long long v) const {
        assert(v > 0);  // operator / not well-defined for v <= 0.
        assert(v < BASE);
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)
            m = (a[i] + m * (long long) BASE) % v;
        return m * sign;
    }

    void operator*=(int v) {
        if (llabs(v) >= BASE) {
            *this *= BigInt(v);
            return ;
        }
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
            if (i == (int) a.size())
                a.push_back(0);
            long long cur = a[i] * (long long) v + carry;
            carry = (int) (cur / BASE);
            a[i] = (int) (cur % BASE);
            //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
            /*
             int val;
             __asm {
             lea esi, cur
             mov eax, [esi]
             mov edx, [esi+4]
             mov ecx, base
             div ecx
             mov carry, eax
             mov val, edx;
             }
             a[i] = val;
             */
        }
        trim();
    }

    BigInt operator*(int v) const {
        if (llabs(v) >= BASE) {
            return *this * BigInt(v);
        }
        BigInt res = *this;
        res *= v;
        return res;
    }

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

    void fft(vector<complex<double> > & a, bool invert) const {
        int n = (int) a.size();

        for (int i = 1, j = 0; i < n; ++i) {
            int bit = n >> 1;
            for (; j >= bit; bit >>= 1)
                j -= bit;
            j += bit;
            if (i < j)
                swap(a[i], a[j]);
        }

        for (int len = 2; len <= n; len <<= 1) {
            double ang = 2 * 3.1415926535897323846 / len * (invert ? -1 : 1);
            complex<double> wlen(cos(ang), sin(ang));
            for (int i = 0; i < n; i += len) {
                complex<double> w(1);
                for (int j = 0; j < len / 2; ++j) {
                    complex<double> u = a[i + j];
                    complex<double> v = a[i + j + len / 2] * w;
                    a[i + j] = u + v;
                    a[i + j + len / 2] = u - v;
                    w *= wlen;
                }
            }
        }
        if (invert)
            for (int i = 0; i < n; ++i)
                a[i] /= n;
    }

    void multiply_fft(const vector<int> &a, const vector<int> &b, vector<int> &res) const {
        vector<complex<double> > fa(a.begin(), a.end());
        vector<complex<double> > fb(b.begin(), b.end());
        int n = 1;
        while (n < (int) max(a.size(), b.size()))
            n <<= 1;
        n <<= 1;
        fa.resize(n);
        fb.resize(n);

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

        res.resize(n);
        long long carry = 0;
        for (int i = 0; i < n; ++i) {
            long long t = (long long) (fa[i].real() + 0.5) + carry;
            carry = t / 1000;
            res[i] = t % 1000;
        }
    }

    BigInt mul_simple(const BigInt &v) const {
```

```
02a         BigInt res;
325         res.sign = sign * v.sign;
4bc         res.a.resize(a.size() + v.a.size());
7a7         for (int i = 0; i < (int) a.size(); ++i)
b40             if (a[i])
761                 for (int j = 0, carry = 0; j < (int) v.a.
    size() || carry; ++j) {
df3                     long long cur = res.a[i + j] + (long
    long) a[i] * (j < (int) v.a.size() ? v.a[j] : 0) + carry;
98c                     carry = (int) (cur / BASE);
ff0                     res.a[i + j] = (int) (cur % BASE);
cbb                 }
d7e         res.trim();
b50         return res;
cbb     }
d41
ad1     typedef vector<long long> vll;
d41
4d4     static vll karatsubaMultiply(const vll &a, const vll &b
    ) {
94d         int n = a.size();
1fb         vll res(n + n);
44d         if (n <= 32) {
830             for (int i = 0; i < n; i++)
f90                 for (int j = 0; j < n; j++)
8dd                     res[i + j] += a[i] * b[j];
b50             return res;
cbb         }
d41
af0         int k = n >> 1;
f9f         vll a1(a.begin(), a.begin() + k);
72c         vll a2(a.begin() + k, a.end());
48e         vll b1(b.begin(), b.begin() + k);
88c         vll b2(b.begin() + k, b.end());
d41
03c         vll a1b1 = karatsubaMultiply(a1, b1);
e56         vll a2b2 = karatsubaMultiply(a2, b2);
d41
40d         for (int i = 0; i < k; i++)
c20             a2[i] += a1[i];
40d         for (int i = 0; i < k; i++)
b00             b2[i] += b1[i];
d41
6a2         vll r = karatsubaMultiply(a2, b2);
be9         for (int i = 0; i < (int) a1b1.size(); i++)
47f             r[i] -= a1b1[i];
cf0         for (int i = 0; i < (int) a2b2.size(); i++)
00a             r[i] -= a2b2[i];
d41
595         for (int i = 0; i < (int) r.size(); i++)
1bf             res[i + k] += r[i];
be9         for (int i = 0; i < (int) a1b1.size(); i++)
d6c             res[i] += a1b1[i];
cf0         for (int i = 0; i < (int) a2b2.size(); i++)
ab9             res[i + n] += a2b2[i];
b50         return res;
cbb     }
d41
287     BigInt mul_karatsuba(const BigInt &v) const {
48c         vector<int> a6 = convert_base(this->a, BASE_DIGITS,
     6);
f64         vector<int> b6 = convert_base(v.a, BASE_DIGITS, 6);
e1c         vll a(a6.begin(), a6.end());
5ed         vll b(b6.begin(), b6.end());
1a8         while (a.size() < b.size())
ddf             a.push_back(0);
0d1         while (b.size() < a.size())
c40             b.push_back(0);
634         while (a.size() & (a.size() - 1))
eed             a.push_back(0), b.push_back(0);
16b         vll c = karatsubaMultiply(a, b);
02a         BigInt res;
325         res.sign = sign * v.sign;
6e2         long long carry = 0;
7db         for (int i = 0; i < (int) c.size(); i++) {
dc9             long long cur = c[i] + carry;
cdf             res.a.push_back((int) (cur % 1000000));
735             carry = cur / 1000000;
cbb         }
7b1         res.a = convert_base(res.a, 6, BASE_DIGITS);
d7e         res.trim();
b50         return res;
cbb     }
d41
933     void operator*=(const BigInt &v) {
fa4         *this = *this * v;
cbb     }
244     BigInt operator*(const BigInt &v) const {
de6         if (a.size() * v.a.size() <= 1000111) return
    mul_simple(v);
fec         if (a.size() > 500111 || v.a.size() > 500111)
    return mul_fft(v);
a67         return mul_karatsuba(v);
cbb     }
d41
0f0     BigInt mul_fft(const BigInt& v) const {
02a         BigInt res;
325         res.sign = sign * v.sign;
d1a         multiply_fft(convert_base(a, BASE_DIGITS, 3),
    convert_base(v.a, BASE_DIGITS, 3), res.a);
74b         res.a = convert_base(res.a, 3, BASE_DIGITS);
d7e         res.trim();
b50         return res;
cbb     }
d41
d41     // -------------------- Misc --------------------
9f0     BigInt abs() const {
18b         BigInt res = *this;
3cc         res.sign *= res.sign;
b50         return res;
cbb     }
a0f     void trim() {
b03         while (!a.empty() && !a.back())
468             a.pop_back();
e28         if (a.empty())
ce6             sign = 1;
cbb     }
d41
88d     bool isZero() const {
5c0         return a.empty() || (a.size() == 1 && !a[0]);
cbb     }
d41
e7c     friend BigInt gcd(const BigInt &a, const BigInt &b) {
183         return b.isZero() ? a : gcd(b, a % b);
cbb     }
797     friend BigInt lcm(const BigInt &a, const BigInt &b) {
8b8         return a / gcd(a, b) * b;
cbb     }
d41
2f7     friend BigInt sqrt(const BigInt &a1) {
b25         BigInt a = a1;
53b         while (a.a.empty() || a.a.size() % 2 == 1)
8a6             a.a.push_back(0);
d41
0c5         int n = a.a.size();
d41
f91         int firstDigit = (int) sqrt((double) a.a[n - 1] *
    BASE + a.a[n - 2]);
3c7         int norm = BASE / (firstDigit + 1);
b65         a *= norm;
b65         a *= norm;
53b         while (a.a.empty() || a.a.size() % 2 == 1)
8a6             a.a.push_back(0);
d41
8a2         BigInt r = (long long) a.a[n - 1] * BASE + a.a[n -
    2];
4e5         firstDigit = (int) sqrt((double) a.a[n - 1] * BASE
    + a.a[n - 2]);
97c         int q = firstDigit;
02a         BigInt res;
d41
a10         for(int j = n / 2 - 1; j >= 0; j--) {
e63             for(; ; --q) {
592                 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);
60f                 if (r1 >= 0) {
011                     r = r1;
c2b                     break;
cbb                 }
cbb             }
d2c             res *= BASE;
f26             res += q;
d41
e79             if (j > 0) {
feb                 int d1 = res.a.size() + 2 < r.a.size() ? r.
    a[res.a.size() + 2] : 0;
baa                 int d2 = res.a.size() + 1 < r.a.size() ? r.
    a[res.a.size() + 1] : 0;
78b                 int d3 = res.a.size() < r.a.size() ? r.a[
    res.a.size()] : 0;
7d9                 q = ((long long) d1 * BASE * BASE + (long
    long) d2 * BASE + d3) / (firstDigit * 2);
cbb             }
cbb         }
d41
d7e         res.trim();
28a         return res / norm;
cbb     }
214 };
Full file hash: f1f35b
```

8.2 FFT

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
27a   FFT:
f7b     FFT allows multiplication of two polynomials in O(n log
     n).
420     This can also be used to multiply two long numbers
    faster.
c00     Other applications:
c35     - All possible sums of two arrays.
1da     - Dot product of vector a with every cyclic shift of
    vector b.
380     - Attaching two boolean stripes without two 1s next to
    each
b26       other.
52f     - String matching.
d41
b95   Usage:
178     long double is a lot slower. 3s with ld and 0.7 with
    double
93d     for 10^6 size vectors.
d41
1d1   Source: https://cp-algorithms.com/algebra/fft.html
c4c */
d41
99b using cd = complex<ld>;
```

```
c4f const ld PI = acos(-1.0L);
d41
9b5 void fft(vector<cd> &a, bool invert)
f95 {
6c3   int n = sz(a);
d41
d94   for (int i = 1, j = 0; i < n; i++)
f95   {
4af     int bit = n >> 1;
474     for (; j & bit; bit >>= 1)
53c       j ^= bit;
53c     j ^= bit;
d41
9dc     if (i < j)
332       swap(a[i], a[j]);
cbb   }
d41
2fe   for (int len = 2; len <= n; len <<= 1)
f95   {
c19     ld ang = 2 * PI / len * (invert ? -1 : 1);
808     cd wlen(cos(ang), sin(ang));
3dd     for (int i = 0; i < n; i += len)
f95     {
8c3       cd w(1);
559       for (int j = 0; j < len / 2; j++)
f95       {
cf0         cd u = a[i + j], v = a[i + j + len / 2] * w;
6c3         a[i + j] = u + v;
273         a[i + j + len / 2] = u - v;
3e4         w *= wlen;
cbb       }
cbb     }
cbb   }
d41
211   if (invert)
f95   {
0b5     for (cd &x : a)
b6d       x /= n;
cbb   }
cbb }
d41
d41 // Input a[0] + a[1]x + a[2]x^2 ...
d41 // Returns polynomial of size equal to the smallest power
    of two at
d41 // least as large as a.size() + b.size(). This can have
    some extra
d41 // zeros.
d41 // Use long double if using long long.
4fc template <class T>
a3a vector<T> multiply(vector<T> const &a, vector<T> const &b)
f95 {
6fa   vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end())
    ;
43e   int n = 1;
86a   while (n < sz(a) + sz(b))
c14     n <<= 1;
37a   fa.resize(n);
870   fb.resize(n);
d41
3a1   fft(fa, false);
c76   fft(fb, false);
830   for (int i = 0; i < n; i++)
940     fa[i] *= fb[i];
959   fft(fa, true);
d41
ebf   vector<T> result(n);
830   for (int i = 0; i < n; i++)
5b3     result[i] = (T)round(fa[i].real()); // Remember to
    remove
d41                         // rounding if working
d41                         // with floats.
dc8   return result;
cbb }
Full file hash: d7b825
```

8.3 Fraction

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
390   Fraction representation:
4d1     All operations run in O(gcd) = O(log).
d41
b95   Usage:
70e     Don't modify internal values, use constructor.
a5c     Some nice things about the constructor:
a0f       frac() = 0/1, frac(5) = 5/1.
d41
77e     Be careful that the numerator and denominator might
    overflow
eca     if lcm is too big.
d15     In those cases, you can always do frac<big_int>, but
    that will
162     be painful to code.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
4fc template <class T>
4cf struct frac
f95 {
e75   T a, b; // b can't be negative, very important.
d41
191   explicit frac(T a = 0, T b = 1) : a(a), b(b) { simpl(); }
d41
7d7   void simpl()
f95   {
8eb     T g = __gcd(abs(a), abs(b)) * sign(b); // Make b
    positive.
fe7     a /= g;
ee2     b /= g;
cbb   }
d41
d59   bool operator<(const frac &rhs) const
f95   {
5c6     return a * rhs.b < rhs.a * b;
cbb   }
d41
7eb   bool operator>(const frac &rhs) const
f95   {
2ab     return rhs < *this;
cbb   }
d41
d60   bool operator==(const frac &rhs) const // TODO: untested.
f95   {
77c     return !(*this < rhs) && !(rhs < *this);
cbb   }
d41
473   frac operator*(const frac &rhs) const
f95   {
f01     return frac(a * rhs.a, b * rhs.b);
cbb   }
d41
04b   frac operator+(const frac &rhs) const
f95   {
3ff     T m = (b * rhs.b) / __gcd(b, rhs.b);
24e     return frac(a * (m / b) + rhs.a * (m / rhs.b), m);
cbb   }
d41
c8c   frac operator-(void) const
f95   {
132     return frac(-a, b);
cbb   }
d41
de2   frac operator-(const frac &rhs) const
f95   {
111     return (*this) + (-rhs);
cbb   }
d41
d63   frac operator/(const frac &rhs) const
f95   {
f52     return (*this) * frac(rhs.b, rhs.a);
cbb   }
d41
9e0   friend ostream &operator<<(ostream &os, const frac &f)
f95   {
891     return os << f.a << "/" << f.b;
cbb   }
214 };
d41
Full file hash: c8862e
```

8.4 Integration

```
d41 /*
f64   Numerical Integration:
c14     Given a function f and an interval [a, b] estimates
    integral
1aa     of f(x) dx from a to b.
bfe     Error is in theory inversely proportional to n^4.
d41
b95   Usage:
be1     n, the number of intervals must be even.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
044 template <class F>
7d9 double simpsons(const F &f, int n /* even */, double a,
    double b)
f95 {
46a   double retv = f(a) + f(b);
d02   double h = (b - a) / n;
acf   for (int i = 1; i < n; i += 2)
900     retv += 4 * f(a + i * h);
1c3   for (int i = 2; i < n; i += 2)
6c1     retv += 2 * f(a + i * h);
d41
055   retv *= h / 3;
627   return retv;
cbb }
d41
d41 // Sample usage:
d41 // int main(void)
d41 // {
d41 //   printf("%.20lf\n", simpsons([](double x) { return pow(
    sin(M_PI * x / 2.0), 3.2);}, 2000, 0, 2));
d41 // }
Full file hash: caa0e5
```

8.5 linalg

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
f92   Vector and matrix operations:
687     Details are given in each function.
a2b     vec inherits from vector<T>, so there is a lot you can
    do
```

```
a0e     with it.
5ae     Also, mat inherits from vector<vec<T>>.
d41
3db   Author: Arthur Pratti Dadalto
d41
1ef   Source: some of it from https://github.com/
9d5               kth-competitive-programming/
3da                 kactl/blob/master/content/numerical/
9a6               MatrixInverse.h
c4c */
d41
4fc template <class T>
fe4 struct vec : vector<T>
f95 {
469   vec(int n) : vector<T>(n) {}
d41
d41   // c = a*x + b*y
e91   static void linear_comb(const vec &a, T x, const vec &b,
      T y, vec &c)
f95   {
8fe     for (int i = 0; i < sz(a); i++)
75e       c[i] = a[i] * x + b[i] * y;
cbb   }
d41
d41   // return a*x + b*y
250   static vec linear_comb(vec a, T x, const vec &b, T y)
f95   {
4fe     linear_comb(a, x, b, y, a);
3f5     return a;
cbb   }
214 };
d41
4fc template <class T>
dad struct mat : vector<vec<T>>
f95 {
d41   // Creates a zero-filled matrix of n rows and m columns.
2d2   mat(int n, int m) : vector<vec<T>>(n, vec<T>(m)) {}
d41
d41   // c = a * x + b * y
762   static void linear_comb(const mat &a, T x, const mat &b,
      T y, mat &c)
f95   {
8fe     for (int i = 0; i < sz(a); i++)
f47       for (int j = 0; j < sz(a[i]); j++)
4f8         c[i][j] = a[i][j] * x + b[i][j] * y;
cbb   }
d41
d41   // return a * x + b * y
08e   static mat linear_comb(mat a, T x, const mat &b, T y)
f95   {
4fe     linear_comb(a, x, b, y, a);
3f5     return a;
cbb   }
d41
13f   mat operator-(const mat &b) const { return linear_comb(*
      this, T(1), b, T(-1)); }
d41
013   mat operator+(const mat &b) const { return linear_comb(*
      this, T(1), b, T(1)); }
d41
93d   mat operator*(const T &x) { return linear_comb(*this, x,
      *this, T(0)); }
d41
d41   // Absolutely does not work for int.
72c   mat operator/(const T &x) const { return linear_comb(*
      this, T(1) / x, *this, T(0)); }
d41
d41   // Multiplication of NxR matrix and a RxM matrix.
d41   // TODO test me on non-square.
c36   mat operator*(mat b) const
f95   {
3f4     int n = (*this).size();
292     int m = b[0].size();
2fd     int r = (*this)[0].size();
a13     mat retv(n, m);
830     for (int i = 0; i < n; i++)
a75       for (int j = 0; j < m; j++)
608         for (int k = 0; k < r; k++)
7c3           retv[i][j] = retv[i][j] + (*this)[i][k] * b[k][j
      ];
d41
627     return retv;
cbb   }
d41
d41   // Returns inverse of matrix (assuming it is square and
d41   // non-singular).
d41   // Runs in O(n^3).
d41   // Absolutely does not work for int.
145   mat inverse() // TODO: test singular.
f95   {
d23     int n = sz(*this);
bca     mat a(n, 2 * n);  // A is Nx2N: X|I.
f7f     vector<int> col(n); // Will be using column pivoting,
d41                         // so need to remember original columns.
830     for (int i = 0; i < n; i++)
f95     {
f90       for (int j = 0; j < n; j++)
c1c         a[i][j] = (*this)[i][j];
34a       a[i][i + n] = T(1);
6dc       col[i] = i;
cbb     }
d41
830     for (int i = 0; i < n; i++)
f95     {
903       int r = i, c = i;
775       for (int j = i; j < n; j++)
90f         for (int k = i; k < n; k++)
f78           if (abs(a[j][k]) > abs(a[r][c]))
d4c             r = j, c = k;
d41
d41       // assert(abs(a[r][c]) > EPS); Uncomment to check
      singular
d41       // matrix
a2f       swap(a[i], a[r]);
d41
f90       for (int j = 0; j < n; j++)
c8c         swap(a[j][i], a[j][c]), swap(a[j][i + n], a[j][c +
      n]);
c1d       swap(col[i], col[c]);
d41
b70       vec<T>::linear_comb(a[i], T(1) / a[i][i], a[i], T(0),
       a[i]);
678       a[i][i] = T(1);
d41
197       for (int j = i + 1; j < n; j++)
370         vec<T>::linear_comb(a[j], T(1), a[i], -a[j][i], a[j
      ]);
cbb     }
d41
d41     // Right now A is:
d41     //
d41     //  1 * *
d41     //  0 1 *
d41     //  0 0 1
d41     //
d41     // Next we remove non-1s from right to left.
d41
917     for (int i = n - 1; i > 0; i--)
c79       for (int j = 0; j < i; j++)
370         vec<T>::linear_comb(a[j], T(1), a[i], -a[j][i], a[j
      ]);
d41
c70     mat retv(n, n);
830     for (int i = 0; i < n; i++)
f90       for (int j = 0; j < n; j++)
4eb         retv[col[i]][col[j]] = a[i][j + n];
627     return retv;
cbb   }
214 };
Full file hash: 2c7bde
```

8.6 NTT

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
79d   Number Theoretic Transform:
5dd     FFT allows multiplication of two polynomials in O(n log
       n)
ebd     where you need the coeficients modulo some specific
      prime.
d41
b95   Usage:
e2e     Can be used for convolutions modulo specific nice
      primes
5b2     of the form (b * 2^a + 1), where the convolution result
149     has size at most $2^a$.
f58     Inputs must be in [0, mod).
d41
1d1   Source: https://cp-algorithms.com/algebra/fft.html
c4c */
d41
b5e const ll mod = (119 << 23) + 1, root = 62; // = 998244353
d41 // For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 <<
       21
d41 // and 483 << 21 (same root). The last two are > 10^9.
d41
4c8 ll modpow(ll b, ll e)
f95 {
d54   ll ans = 1;
36e   for (; e; b = b * b % mod, e /= 2)
654     if (e & 1)
6a3       ans = ans * b % mod;
ba7   return ans;
cbb }
d41
192 typedef vector<ll> vl;
3f3 void ntt(vl &a, vl &rt, vl &rev, int n)
f95 {
830   for (int i = 0; i < n; i++)
b3f     if (i < rev[i])
1e6       swap(a[i], a[rev[i]]);
657   for (int k = 1; k < n; k *= 2)
1e5     for (int i = 0; i < n; i += 2 * k)
68f       for (int j = 0; j < k; j++)
f95       {
86e         ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i +
      j];
93d         a[i + j + k] = (z > ai ? ai - z + mod : ai - z);
589         ai += (ai + z >= mod ? z - mod : z);
cbb       }
cbb }
d41
92d vl conv(const vl &a, const vl &b)
f95 {
41f   if (a.empty() || b.empty())
21d     return {};
6b2   int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n =
       1 << B;
642   vl L(a), R(b), out(n), rt(n, 1), rev(n);
```

```
6b4   L.resize(n), R.resize(n);
d41
830   for (int i = 0; i < n; i++)
a17     rev[i] = (rev[i / 2] | (i & 1) << B) / 2;
d41
b2c   ll curL = mod / 2, inv = modpow(n, mod - 2);
414   for (int k = 2; k < n; k *= 2)
f95   {
893     ll z[] = {1, modpow(root, curL /= 2)};
1d3     for (int i = k; i < 2 * k; i++)
256       rt[i] = rt[i / 2] * z[i & 1] % mod;
cbb   }
d41
2fa   ntt(L, rt, rev, n);
89f   ntt(R, rt, rev, n);
830   for (int i = 0; i < n; i++)
1cb     out[-i & (n - 1)] = L[i] * R[i] % mod * inv % mod;
bc5   ntt(out, rt, rev, n);
c20   return {out.begin(), out.begin() + s};
cbb }
Full file hash: ebd7dd
```

8.7 Simplex

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
458   Simplex:
695     Optimizes a linear program of the form:
15b       maximize c*x, s.t. a*x <ops> b, x >= 0.
7b8     Each constraint can use a different operator from {<=
    >= ==}.
352     Not polynomial, but got AC 150 ms with 4000 constraints
     and
021     200 variables.
d41
b95   Usage:
6c3     Call run_simplex, with the number of constraints and
852     variables, a, b, ops and c (as specified above).
f28     Return value is ok if solution was found, unbounded if
cf1     objective value can be infinitely large
eb4     or infeasible if there is no solution given the
    constraints.
d41
38f     The value of each variable is returned in vector res.
90f     Objective function optimal value is also returned.
060     Sample usage is commented below.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
4fc template <class T>
fe4 struct vec : vector<T>
f95 {
469   vec(int n) : vector<T>(n) {}
d41
d41   // c = a*x + b*y
e91   static void linear_comb(const vec &a, T x, const vec &b,
    T y, vec &c)
f95   {
8fe     for (int i = 0; i < sz(a); i++)
75e       c[i] = a[i] * x + b[i] * y;
cbb   }
214 };
d41
4fc template <class T>
dad struct mat : vector<vec<T>>
f95 {
d41   // Creates a zero-filled matrix of n rows and m columns.
2d2   mat(int n, int m) : vector<vec<T>>(n, vec<T>(m)) {}
d41
d41   // Erase row O(n^2).
824     void erase_row(int i)
f95     {
7c9         this->erase(this->begin() + i);
cbb     }
d41
d41   // Erase column O(n^2).
1b2     void erase_col(int j)
f95     {
798         for (int i = 0; i < sz(*this); i++)
a77             (*this)[i].erase((*this)[i].begin() + j);
cbb     }
214 };
d41
d3f namespace simplex
f95 {
d41 // Any value within [-EPS, +EPS] will be considered equal
    to 0.
056 const double EPS = 1e-6;
d41
5e6 enum op { ge, le, eq };
d41
242 enum optimization_status { ok, unbouded, infeasible };
d41
4d9 int get_entering_var(mat<double> &tab)
f95 {
d41   // Get first non-artificial variable with negative
    objective
d41   // coeficient. If none, return -1. (could instead return
    most
d41   // negative, but that could cycle)
682   for (int i = 0; i < sz(tab[0]) - 1; i++)
72e     if (tab[0][i] < -EPS)
d9a       return i;
daa   return -1;
cbb }
d41
201 int get_exiting_var_row(mat<double> &tab, int entering_var)
f95 {
d41   // Get smallest value of val and first in case of tie. If
     none,
d41   // return -1.
fcb   int retv = -1;
621   double val = -1.0;
a07   for (int i = 1; i < sz(tab); i++)
f95   {
d41     // If strictly positive, it bounds the entering var.
dcd     if (tab[i][entering_var] > EPS)
f95     {
d41       // Entering var will be bounded by
d41       // tab[i][tab.size().second - 1] / tab[i][
    entering_var].
d41       // val could be slightly negative if
d41       //   tab[i][tab.size().second - 1] = -0.
393       if (val == -1.0 || tab[i][sz(tab[i]) - 1] / tab[i][
    entering_var] < val)
f95       {
78d         val = tab[i][sz(tab[i]) - 1] / tab[i][entering_var
    ];
52c         retv = i;
cbb       }
cbb     }
cbb   }
d41
627   return retv;
cbb }
d41
ed2 optimization_status solve_tab(mat<double> &tab, vector<int>
     &basic_var)
f95 {
d41   // artificial_count is the number of variables at the end
     we
d41   // should ignore.
a17   int entering_var;
6b7   while ((entering_var = get_entering_var(tab)) != -1)
f95   {
6c0     int exiting_var_row = get_exiting_var_row(tab,
    entering_var);
d41
d41     // If no exiting variable bounds the entering variable,
     the
d41     // objective is unbounded.
813     if (exiting_var_row == -1)
914       return optimization_status::unbouded;
d41
d41     // Set new basic var coeficient to 1.
89c     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]);
d41
d41     // Gaussian elimination of the other rows.
c7a     for (int i = 0; i < sz(tab); i++)
81c       if (i != exiting_var_row)
ed2         if (abs(tab[i][entering_var]) > EPS)
7ad           vec<double>::linear_comb(tab[i], 1.0, tab[
    exiting_var_row], -tab[i][entering_var], tab[i]);
d41
64d     basic_var[exiting_var_row] = entering_var;
cbb   }
d41
c52   return optimization_status::ok;
cbb }
d41
d41 // maximize c*x, s.t. a*x <ops> b. x >= 0.
f1a 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)
f95 {
334   for (int i = 0; i < num_constraints; i++)
5f9     if (ops[i] == op::ge)
f95     {
d41       // Beyond this point "ge" constraints won't exist.
444       vec<double>::linear_comb(a[i], -1, a[i], 0, a[i]); //
     a[i] *= -1;
250       b[i] *= -1;
1c3       ops[i] = op::le;
cbb     }
d41
026   int num_artificial_variables = 0;
371   int num_slack_variables = 0;
334   for (int i = 0; i < num_constraints; i++)
f95   {
0ec     if (ops[i] == op::le)
f95     {
37a       num_slack_variables++;
cbb     }
d41
359     if ((ops[i] == op::le && b[i] < -EPS) || ops[i] == op::
    eq)
f95     {
d41       // If we have rhs strictly negative in a inequality
    or an
d41       // equality constraint, we need an artificial val.
fc3       num_artificial_variables++;
cbb     }
cbb   }
d41
854   mat<double> tab(num_constraints + 1, num_vars +
    num_slack_variables + num_artificial_variables + 1);
```

```
9a9   vector<int> basic_var(num_constraints + 1);
775   vector<int> slack_cols, artificial_cols;
7f6   for (int i = num_vars; i < num_vars + num_slack_variables
      ; i++)
10c     slack_cols.push_back(i);
e0b   for (int i = num_vars + num_slack_variables; i < num_vars
       + num_slack_variables + num_artificial_variables; i++)
eaf     artificial_cols.push_back(i);
c70   int rhs_col = num_vars + num_slack_variables +
      num_artificial_variables;
d41
d41   // First objective will be to have artificial variables
      equal to 0
017   for (int i : artificial_cols)
b98     tab[0][i] = 1;
d41
9c4   for (int i = 0, k = 0, l = 0; i < num_constraints; i++)
f95   {
861     for (int j = 0; j < num_vars; j++)
e38       tab[i + 1][j] = a[i][j];
d41
0ec     if (ops[i] == op::le)
141       tab[i + 1][slack_cols[l++]] = 1;
d41
142     tab[i + 1][rhs_col] = b[i];
d41
359     if ((ops[i] == op::le && b[i] < -EPS) || ops[i] == op::
      eq)
f95     {
d41       // Basic var will be artificial
2a6       if (b[i] < -EPS)
009         vec<double>::linear_comb(tab[i + 1], -1, tab[i +
      1], 0, tab[i + 1]); // a[i] *= -1;
d41
86f       tab[i + 1][artificial_cols[k++]] = 1;
116       basic_var[i + 1] = artificial_cols[k - 1];
d41
06d       vec<double>::linear_comb(tab[0], 1.0, tab[i + 1],
      -1.0, tab[0]);
cbb     }
295     else // Basic var will be slack var.
f95     {
ae7       basic_var[i + 1] = slack_cols[l - 1];
cbb     }
cbb   }
d41
df8   assert(solve_tab(tab, basic_var) == optimization_status::
      ok);
d41
d41   // Best solution could not bring artificial variables to
      0
d41   // (objective max Z = sum(-xa)).
fe0   if (tab[0][sz(tab[0]) - 1] < -EPS)
94b     return optimization_status::infeasible;
d41
d41   // If we have an artificial variable on the base with xb
      = 0, we
d41   // need to remove it.
e64   for (int i = 1; i < sz(basic_var); i++)
077     if (basic_var[i] >= num_vars + num_slack_variables)
f95     {
d41       // Find non-artificial replacement.
e2f       for (int j = 0; j < sz(tab[i]) - 1 -
      num_artificial_variables; j++)
f95       {
d41         // If non-zero value in row, we can replace.
a88         if (j != basic_var[i] && abs(tab[i][j]) > EPS)
f95         {
d41           // Remove from the other rows.
b5f           vec<double>::linear_comb(tab[i], 1.0 / tab[i][j],
       tab[i], 0, tab[i]);
d41
443           for (int k = 0; k < sz(tab); k++)
635             if (k != i)
f95             {
e76               if (abs(tab[k][j]) > EPS)
4b6                 vec<double>::linear_comb(tab[k], 1, tab[i],
      -tab[k][j], tab[k]);
cbb             }
d41
d41           // Basic variable replacemente done, so proceed
      to
d41           // next basic_var.
7e0           basic_var[i] = j;
c2b           break;
cbb         }
cbb       }
cbb     }
d41
ca2   for (int i = sz(tab) - 1; i > 0; i--)
077     if (basic_var[i] >= num_vars + num_slack_variables)
f95     {
d41       // Could not replace basic var, so constraint is
      redundant
2cd       tab.erase_row(i);
fe1       basic_var.erase(basic_var.begin() + i);
cbb     }
d41
d41   // Remove artificial variable columns.
5c3   for (int i = sz(artificial_cols) - 1; i >= 0; i--)
9a2     tab.erase_col(artificial_cols[i]);
d41
131   for (int i = 0; i < sz(tab[0]); i++)
d26     tab[0][i] = 0;
f17   for (int i = 0; i < num_vars; i++)
942     tab[0][i] = -c[i];
d41
a07   for (int i = 1; i < sz(tab); i++)
b39     vec<double>::linear_comb(tab[0], 1, tab[i], -tab[0][
      basic_var[i]], tab[0]);
d41
54a   optimization_status status = solve_tab(tab, basic_var);
d41
b68   res = vec<double>(num_vars);
e64   for (int i = 1; i < sz(basic_var); i++)
047     if (basic_var[i] < num_vars)
81f       res[basic_var[i]] = tab[i][sz(tab[i]) - 1];
d41
a34   obj_val = tab[0][sz(tab[0]) - 1];
d41
62d   return status;
cbb }
cbb } // namespace simplex
d41
d41 /*
13a int main(void)
f95 {
14e   int n, m;
aa3   cin >> n >> m;
d41
37c   int num_constraints = m, num_vars = n;
d41
d41   // maximize c*x, s.t. a*x <ops> b. x >= 0.
262   mat<double> a(num_constraints, num_vars);
84d   vec<double> b(num_constraints);
01b   vec<simplex::op> ops(num_constraints);
dab   vec<double> c(num_vars);
40c   vec<double> res(num_vars);
d41
830   for (int i = 0; i < n; i++)
a73     cin >> c[i];
d41
94f   for (int i = 0; i < m; i++)
f95   {
7ba     int l, r, x;
159     cin >> l >> r >> x;
0df     for (int j = l - 1; j <= r - 1; j++)
a21       a[i][j] = 1;
df0     b[i] = x;
803     ops[i] = simplex::op::le;
cbb   }
d41
1af   double ans;
dd6   simplex::run_simplex(num_constraints, num_vars, a, ops, b
      , c, res, ans);
d41
530   cout << ((long long)(ans + 0.5)) << endl;
cbb }
c4c */
Full file hash: 46f321
```

9 String

9.1 Aho Corasick

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
305   Aho-Corasick: O(alpha_size * string_sum)
4e9     In general, multiple pattern string matching tree/
      automaton.
d41
a6f     Keep in mind that find_all can be O(N*sqrt(N)) if no
      duplicate
b16     patterns. (N is total string length)
d41
ca2   Constraints:
76a     chars in the string are all in the interval
662     [first, first + alpha_size - 1].
3da     This will not free some memory on object destruction.
390     Duplicate patterns are allowed, empty patterns are not.
d41
b95   Usage:
df3     Set alpha_size and the first char in the alphabet.
e98     Call constructor passing the list of pattern strings.
0f1     Use one of find, find_all ... to process a text or do
      your own
9d3     thing.
42a     To find the longest words that start at each position,
      reverse
ac5     all input.
343     Bottleneck in this code is memory allocation.
91a     For 10^6 total string size, memory usage can be up to
      300 Mb.
d41
b34     You can save time:
3cd       list_node, match_list, match_list_last are only
      needed to
d10       list all matches.
57e       atm automaton table can be cut to reduce memory usage
      .
018       The text processing stuff is also optional.
02e       Node memory can be one big array instead of vector.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
e7f struct aho_corasick
f95 {
```

```
  enum
  {
    alpha_size = 26,  // Number of chars in the alphabet.
    first = 'a'     // First char.
  };

  struct list_node    // Simple linked list node struct.
  {
    int id;
    list_node *next;
    explicit list_node(int id, list_node *next) : id(id), next(next) {}
  };

  struct node
  {
    int fail = -1;      // node failure link (aka suffix link).

    int nmatches = 0;   // Number of matches ending in this
                        // node.

    int next[alpha_size]; // Next node in trie for each letter.
                          // Replace with unordered_map or list
                          // if memory is tight.

    int atm[alpha_size];  // Optional: Automaton state
                          // transition table. Simpler text
                          // processing.

    // Pointer to first node in linked list of matches.
    // List ends with null pointer.
    list_node *match_list = nullptr;

    // Internal: pointer to last node in list of matches
    // (before bfs), or null if empty list.
    list_node *match_list_last = nullptr;

    // Start with all invalid transitions.
    node() { memset(next, -1, sizeof(next)); }
  };

  vector<node> nodes;

  aho_corasick(const vector<string> &pats)
  {
    nodes.emplace_back(); // Make root node 0.
    for (int i = 0; i < sz(pats); i++)
    {
      int cur = 0;  // Start from root.
      for (int j = 0; j < sz(pats[i]); j++)
      {
        int k = pats[i][j] - first;

        if (nodes[cur].next[k] <= 0)
        {
          // Make new node if needed.
          nodes[cur].next[k] = sz(nodes);
          nodes.emplace_back();
        }

        cur = nodes[cur].next[k];
      }

      // Add logic here if additional data is needed on matched
      // strings.
      nodes[cur].nmatches++;
      // Add string to node list of matches.
      nodes[cur].match_list = new list_node(i, nodes[cur].match_list);
      if (nodes[cur].nmatches == 1)
        nodes[cur].match_list_last = nodes[cur].match_list;
    }

    queue<int> q;
    // Define fail for first level.
    for (int i = 0; i < alpha_size; i++)
    {
      // Invalid transitions from 0 now become valid self
      // transitions.
      if (nodes[0].next[i] == -1)
        nodes[0].next[i] = 0;

      // Automaton state transition table.
      nodes[0].atm[i] = nodes[0].next[i];

      // Single letter nodes have fail = 0 and go in the queue.
      if (nodes[0].next[i] > 0)
      {
        q.push(nodes[0].next[i]);
        nodes[nodes[0].next[i]].fail = 0;
      }
    }

    while (!q.empty()) // Use bfs to compute fail for next level.
    {
      int cur = q.front();
      q.pop();

      for (int i = 0; i < alpha_size; i++)
        if (nodes[cur].next[i] > 0) // Don't use -1 and don't
                                    // use transition to root.
        {
          // Unrelated to code below, filling automaton.
          nodes[cur].atm[i] = nodes[cur].next[i];

          // Computing fail for next node and putting it in
          // the queue.
          int prox = nodes[cur].next[i];
          q.push(prox);

          int state = nodes[cur].fail;
          while (nodes[state].next[i] == -1)
            state = nodes[state].fail;

          nodes[prox].fail = nodes[state].next[i];

          // Add logic here if additional data is needed on
          // matched strings.
          nodes[prox].nmatches += nodes[nodes[prox].fail].nmatches;

          // Add in O(1) list from fail link to next node's
          // list.
          // Operation: a->b->null c->null to a->b->c->null.
          (nodes[prox].match_list_last ? nodes[prox].match_list_last->next : nodes[prox].match_list) = nodes[nodes[prox].fail].match_list;
        }
        else
        {
          nodes[cur].atm[i] = nodes[nodes[cur].fail].atm[i];
        }
    }
  }

  // Optional
  // Returns a vector retv such that, for each text position i:
  // retv[i] is the index of the largest pattern ending at position
  // i in the text.
  // If retv[i] == -1, no pattern ends at position i.
  vector<int> find(const string &text)
  {
    vector<int> retv(sz(text));
    int cur = 0;

    for (int i = 0; i < sz(text); i++)
    {
      cur = nodes[cur].atm[text[i] - first];
      retv[i] = (nodes[cur].match_list ? nodes[cur].match_list->id : -1);
    }

    return retv;
  }

  // Optional
  // Returns a vector retv such that, for each text position i:
  // retv[i] is the number of pattern matches ending at position i
  // in the text.
  vector<int> count(const string &text)
  {
    vector<int> retv(sz(text));
    int cur = 0;

    for (int i = 0; i < sz(text); i++)
    {
      cur = nodes[cur].atm[text[i] - first];
      retv[i] = nodes[cur].nmatches;
    }

    return retv;
  }

  // Optional
  // Returns a vector retv such that, for each text position i:
  // retv[i] is a list of indexes to the patterns ending at position
  // i in the text.
  // These lists will be sorted from largest to smallest pattern
  // length.
  // Keep in mind that find_all can be O(N*sqrt(N)) if no duplicate
  // patterns. (N is total string length)
  vector<vector<int>> find_all(const string &text)
  {
    vector<vector<int>> retv(sz(text));
    int cur = 0;

    for (int i = 0; i < sz(text); i++)
    {
      cur = nodes[cur].atm[text[i] - first];
      for (auto n = nodes[cur].match_list; n != nullptr; n = n->next)
        retv[i].push_back(n->id);
    }

```

```
627     return retv;
cbb   }
d41
d41   // Optional
d41   // Returns a vector retv such that:
d41   // retv is a list of indexes to the patterns ending at
    position
d41   // pos in the text.
d41   // This list will be sorted from largest to smallest
    pattern
d41   // length.
251   vector<int> find_all_at_pos(const string &text, int pos)
f95   {
aeb     vector<int> retv;
b3d     int cur = 0;
d41
774     for (int i = 0; i < sz(text); i++)
f95     {
13d       cur = nodes[cur].atm[text[i] - first];
d41
c57       if (i == pos)
d82         for (auto n = nodes[cur].match_list; n != nullptr;
    n = n->next)
1ad           retv.push_back(n->id);
cbb     }
d41
627     return retv;
cbb   }
214 };
Full file hash: 2ec64b
```

9.2 Hash

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
032   String hashing:
4d6     Get polynomial hash for any substring in O(1) after O(n
    )
76e     preprocessing.
d41
b95   Usage:
cac     Good values c = 137, mod = 10^9 + 7.
922     If necessary to check too many pairs of hashes, use two
c7c     different hashes.
d41
107     If hashing something other than english characters:
eb7       - Don't have elements with value 0.
0e7       - Use c > max element value.
c4c */
d41
164 struct hash_interval
f95 {
880   ll c, mod;
dcf   vector<ll> h, p;
d41   hash_interval(const string &s, ll c, ll mod) : c(c), mod(
    mod), h(sz(s) + 1), p(sz(s) + 1)
f95   {
c4b     p[0] = 1;
cdc     h[0] = 0;
11e     for (int i = 0; i < sz(s); i++)
f95     {
909       h[i + 1] = (c * h[i] + s[i]) % mod;
e69       p[i + 1] = (c * p[i]) % mod;
cbb     }
cbb   }
d41
d41   // Returns hash of interval s[a ... b] (where 0 <= a <= b
    < sz(s))
d12   ll get(int a, int b)
f95   {
2c6     return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod)
     % mod;
cbb   }
214 };
Full file hash: b9525a
```

9.3 KMP

```
5d1 #include "../../contest/header.hpp"
d41
d41 /*
8de   Prefix Function and KMP:
e45     Computes prefix function for a given string in O(n).
16b     String matching in O(n + m).
37f     No need to be strings, you can use vector<int> since
    the
e68     algorithms don't depend on the alphabet size, they only
f21     perform equality comparisons.
b5e     Usage is explained in each function.
d41
3db   Author: Arthur Pratti Dadalto
c4c */
d41
d41 // Returns the prefix function for the given string.
d41 // pi[i] for 0 <= i <= s.size() (s.size() + 1 elements).
d41 // pi[i] considers the prefix of string s having size i.
d41 // pi[i] is the size of its (the prefix's) largest proper
    prefix
d41 // which is also a suffix.
d41 // For "aabaaab", pi is is {0,0,1,0,1,2,2,3}
4fc template <class T>
8fa vector<int> prefix_function(T s)
f95 {
d2c   vector<int> pi(s.size() + 1, 0);
a94   for (int i = 2; i <= s.size(); i++)
f95   {
3f8     int j = pi[i - 1]; // j is the size of the candidate
    prefix
d41               // to expand.
d41
4b3     while (j > 0 && s[j] != s[i - 1]) // While we still
    have a
d41                       // candidate prefix and it
d41                       // can't be expanded.
d41
187       j = pi[j];  // Go to the next candidate prefix.
d41
d41     // If candidate prefix can be expanded, do it.
    Otherwise,
d41     // there is no prefix that is also a suffix.
f98     pi[i] = s[j] == s[i - 1] ? j + 1 : 0;
cbb   }
d41
81d   return pi;
cbb }
d41
d41 // Returns a sorted list of all positions in the text
    string where
d41 // begins an ocurrence of the key string.
d41 // e.g. kmp("aabaaab", "aab") returns {0, 4}.
4fc template <class T>
15b vector<int> kmp(T text, T key)
f95 {
aeb   vector<int> retv;
7fa   vector<int> pi = prefix_function(key);
d41
d41   // There is no need to have the entire text in memory,
    you could
d41   // do this char by char.
5d9   for (int i = 0, match = 0; i < text.size(); i++)
f95   {
d41     // match stores the size of the prefix of the key which
    is a
d41     // suffix of the current processed text.
9d9     while (match > 0 && text[i] != key[match])
7eb       match = pi[match];
db8     if (text[i] == key[match])
24b       match++;
d41
dd8     if (match == key.size())
f95     {
7b8       retv.push_back(i - match + 1);
7eb       match = pi[match]; // To avoid access to key[key.size
    ()]
d41                   // in next iteration.
cbb     }
cbb   }
d41
627   return retv;
cbb }
Full file hash: 415801
```

9.4 Suffix Array

```
d41
5d1 #include "../../contest/header.hpp"
d41
d41 /*
1f7   Suffix array:
47c     Build suffix array and LCP array in O((n + lim) log n)
    using
9ff     O(n + lim) memory, where lim is the alphabet size.
d41
fbd     sa[i] is the starting index of the suffix which is i-th
    in the
ec3     sorted suffix array.
765     The returned vector is of size s.size()+1,
15c     and sa[0] == s.size(). The '\0' char at the end is
    considered
c45     part of the string, so sa[0] = "\0", the prefix
    starting at
af1     index s.size().
d41
878     The lcp array contains longest common prefixes for
e7b     neighbouring strings in the suffix array:
e41       lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0.
d41
81e   Example:
981     Computing the LCP and the SA of "GATAGACA"
d33       i sa[i] lcp[i]  suffix
fd7       0 8   0   ""
cac       1 7   0   "A"
430       2 5   1   "ACA"
d30       3 3   1   "AGACA"
c89       4 1   1   "ATAGACA"
1a0       5 6   0   "CA"
b1b       6 4   0   "GACA"
299       7 0   2   "GATAGACA"
08e       8 2   0   "TAGACA"
d41
b95   Usage:
6a6     Important: the input string must not contain any zero
    values.
95c     Must use C++11 or above.
87e     You can use this for strings of integers, just change
    the
421     alphabet size.
d41
```

```
  Source: https://github.com/kth-competitive-programming/kactl/blob/
  master/content/strings/SuffixTree.h
*/

struct suffix_array
{
  vector<int> sa, lcp;
  suffix_array(const string &s, int lim = 256) // or basic_string<int> for integer strings.
  {
    int n = sz(s) + 1, k = 0, a, b;
    vector<int> x(s.begin(), s.end() + 1), y(n), ws(max(n, lim)), rank(n);
    sa = lcp = y;
    iota(sa.begin(), sa.end(), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p)
    {
      p = j;
      iota(y.begin(), y.end(), n - j);
      for (int i = 0; i < n; i++)
        if (sa[i] >= j)
          y[p++] = sa[i] - j;
      fill(ws.begin(), ws.end(), 0);
      for (int i = 0; i < n; i++)
        ws[x[i]]++;
      for (int i = 1; i < lim; i++)
        ws[i] += ws[i - 1];
      for (int i = n; i--;)
        sa[--ws[x[y[i]]]] = y[i];
      swap(x, y);
      p = 1;
      x[sa[0]] = 0;

      for (int i = 1; i < n; i++)
      {
        a = sa[i - 1];
        b = sa[i];
        x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
      }
    }

    for (int i = 1; i < n; i++)
      rank[sa[i]] = i;

    for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
      for (k &&k--, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; k++)
        ;
  }
};
Full file hash: 87092f
```

9.5 Suffix Tree

```
/*
  Suffix Tree:
        A compressed trie is a trie where all reduntant nodes are
        eliminated by allowing edges to hold substrings.
        A Suffix Tree is a Trie containig all the suffixes of a
        certain string S.
        Using a dummy character in the end of S gurantees that all
        suffixes end in leafs, and vice-vers.
        This code builds the Suffix Tree in O(|S|*lg|Alph|) time and
        O(|S|) memory (where |Alph| is the size of the alphabet).

  Example:
    A dfs through the Suffix Tree of "banana$" looks like:
        Begin in root
        Enter through "$"
        Leave through "$"
        Enter through "na"
        Enter through "$"
        Leave through "$"
        Enter through "na$"
        Leave through "na$"
        Enter through "banana$"
        Leave through "banana$"
        Enter through "a"
        Enter through "$"
        Leave through "$"
        Enter through "na"
        Enter through "$"
        Leave through "$
        Enter through "na$"
        Leave through "na$"
        Leave through "na"
        Leave through "a"

  Usage:
    Create an object Suffix Tree st passing the string as
        argument, and optionally the dummy character as second
        argument.
        "verify_substring(P)" checks in O(|P|) if P is a substring of
        S.


  Author: Augusto Damschi Bernardi
    Based on: https://bcc.ime.usp.br/tccs/2016/yancouto/tcc.pdf
*/
#include "../../contest/header.hpp"

struct node{
    // Each node keeps information about the edge arriving into it.

    // Keeps left and right index of edge's substring in S.
    // (may not be the same occurrence one, see "aba$");
    int left, right;

    // *parent points to parent node
    // *suffix points to node corresponding to [left+1...right]
    // (by the end of the process exists for every node other than
    // the root)
    node *parent, *suffix;

    //next_node[c] points to the kid of current node whose edge
    // begins with character c (only one by character).
    map<char, node*> next_node;

    node(int _left, int _right, node *_parent):
    left(_left), right(_right), parent(_parent){}

    ~node(){
        for(auto child : next_node)
            delete child.second;
    }

    //Lenght of current edge
    int len(){
        return right - left + 1;
    }

    //Convinient way to find kid
    node* next(char c){
        if(next_node.count(c))
            return next_node[c];
        return NULL;
    }
};

struct suffix_tree{
    node *root;
    char dummy;
    string s;

    suffix_tree(string _s, char _dummy = '$')
    {
    s = _s;
    dummy = _dummy;
        s += dummy;
        root = new node(0, -1, NULL);
        // In the beginning of iteration i,j, node cur_node in
        // [left...cur_dist] represents [i...j-1]
        // need_suffix points to node that doesn't have a suffix yet
        // (at most one at a time, for at most one iteration of i)
        node *cur_node = root;
        int cur_dist = -1, i = 0;

        //Invariants:
        //  *At the beginning of step i,j, s[i...j-1] and all of it's
        // suffixes are inserted in the suffix trie
        //  *At the beggining of step i,j, cur_node[cur_dist] is the
        // end point representing s[i...j-1]
        //  *At any increment of i, at most one node doesn't have
        // "suffix" field defined, and it's stored in "need_suffix"
        for(int j = 0; j < (int)s.size(); j++){
            char c = s[j];
            node *need_suffix = NULL;
            while(i <= j){
                // Inserts s[i...j]

                // Case 1: s[i...j] already exists in the suffix tree
                // If it's in the next node, move to it
                if(cur_dist == cur_node->len() - 1 and cur_node->next(c) != NULL){
                    cur_node = cur_node->next(c);
                    cur_dist = -1;
                }
                // If now we have to take one more char from the
                // edge, take it
                if(cur_dist < cur_node->len() - 1 and get_char(cur_node, cur_dist + 1) == c){
                    cur_dist += 1;
                    break;
                }
```

```
d41                 // Case 2: s[i...j-1] ends in a node
716                 if(cur_dist == cur_node->len() - 1){
b95                     cur_node->next_node[c] = new node(j, s.
    size() - 1, cur_node);
d41                     // Puts cur_node in s[i_1...j-1]
1e8                     if(cur_node != root){
55a                         cur_node = cur_node->suffix;
bb6                         cur_dist = cur_node->len() - 1;
cbb                     }
cbb                 }
d41
d41                 // Caso 3: s[i...j-1] ends in an edge
4e6                 else{
d41                     // Creates a new node and splits the
    edge
593                     node *mid = new node(cur_node->left,
    cur_node->left + cur_dist, cur_node->parent);
9e7                     cur_node->parent->next_node[get_char(
    mid, 0)] = mid;
d79                     mid->next_node[get_char(cur_node,
    cur_dist + 1)] = cur_node;
49e                     cur_node->parent = mid;
e8c                     cur_node->left += cur_dist + 1;
2f1                     mid->next_node[s[j]] = new node(j, s.
    size() - 1, mid);
d41                     // Sets any missing suffix link
07a                     if(need_suffix != NULL)
b9c                         need_suffix->suffix = mid;
d41                     // Tries to find the suffix link for "
    mid"
37a                     cur_node = mid->parent;
6a5                     int g;
1e8                     if(cur_node != root){
55a                         cur_node = cur_node->suffix;
a07                         g = j - cur_dist - 1;
cbb                     }
295                     else
26a                         g = i + 1;
d41                     // Initially cur_node points to node
d41                     // [i+1 ... g-1]
7b2                     while(g < j and g + cur_node->next(s[g
    ])->len() <= j){
a62                         cur_node= cur_node->next(s[g]);
976                         g += cur_node->len();
cbb                     }
d41                     // Case where suffix link was found
9ad                     if(g == j){
3fe                         need_suffix = NULL;
63a                         mid->suffix = cur_node;
bb6                         cur_dist = cur_node->len() - 1;
cbb                     }
d41                     // Case where suffix link doesnt exists
     yet
4e6                     else{
4d6                         need_suffix = mid;
a62                         cur_node = cur_node->next(s[g]);
9b5                         cur_dist = j - g - 1;
cbb                     }
cbb                 }
b2c                 i += 1;
cbb             }
cbb         }
cbb     }
d41
1b4     ~suffix_tree(){
cd7         delete root;
cbb     }
d41
837     char get_char(node *cur, int ind){
49b         return s[cur->left + ind];
cbb     }
d41
d41     //Optional
18f     bool verify_substring(string sub){
f85         node *cur_node = root;
116         int cur_dist = -1;
d41
2bf         for(char c : sub){
38c             if(cur_dist < cur_node->len() - 1){
d70                 if(get_char(cur_node, cur_dist + 1) != c)
d1f                     return false;
0eb                 cur_dist++;
cbb             }
4e6             else{
8cd                 if(cur_node->next(c) == NULL)
d1f                     return false;
566                 cur_node = cur_node->next(c);
6c7                 cur_dist = 0;
cbb             }
cbb         }
d41
8a6         return true;
cbb     }
d41
d41     //Onlu for debbuging
b29     void print(node* cur_node = NULL){
684     if(cur_node == NULL)
7b3       cur_node = root;
a53     printf("node %d %d [", cur_node->left, cur_node->right)
    ;
1d7     for(int i = cur_node->left; i <= cur_node->right; i++)
e6b       printf("%c", s[i]);
edd     printf("] entra\n");
d41
289         for(auto el : cur_node->next_node){
316             print(el.second);
cbb         }
d41
a53         printf("node %d %d [", cur_node->left, cur_node->
    right);
1d7     for(int i = cur_node->left; i <= cur_node->right; i++)
e6b       printf("%c", s[i]);
b9a     printf("] sai\n");
cbb   }
214 };
Full file hash: 622629
```