

Contents

1	Contest	4
1.1	Header	4
1.2	Sample Debug	6
1.3	Hash Code (Line)	7
1.4	Hash Code (File)	7
1.5	Diff Script	7
1.6	Submit Script	7
2	Data Structures	8
2.1	2D Segment Tree	8
2.2	BIT	12
2.3	BIT2D	13
2.4	Dynamic Segment Tree	14
2.5	Linear Container	16
2.6	Merge Sort Tree	18
2.7	Min Queue	20
2.8	Persistent Segment Tree	21
2.9	Segment Tree	23
2.10	Lazy Segment Tree	27
2.11	Key Treap	31
2.12	Sequential Treap	34
2.13	Union Find	37
3	Geometry	38
3.1	2D	38
3.2	3D	47
3.3	Convex Polygon and Circle Intersection	49
3.4	Closest Pair of points	52
3.5	Convex Hull – Sweep Line	54
3.6	Convex Hull – Graham Scan	56
3.7	Min Enclosing Circle (randomized)	58
3.8	Min Enclosing Circle (ternary search)	59
3.9	Rotating Calipers – Antipodal	61
3.10	Rotating Calipers – Convex Polygon Bouding Box	62
3.11	Rotating Calipers – Convex Polygon Diameter	63
3.12	Rotating Calipers – Convex Polygon Width	64
4	Graph	65
4.1	2-Sat	65
4.2	Biconnected Components	68
4.3	Bipartite Matching (Hopcroft Karp)	72
4.4	Bridges/Articulation Points	74
4.5	Centroid Decomposition	76
4.6	Euler Tour	78
4.7	Max Flow (Dinic)	82
4.8	Max Flow (Dinic w/ Scaling)	85
4.9	Min Cost Max Flow	88
4.10	Gomory Hu (Min cut)	91

4.11	Heavy-Light Decomposition	93
4.12	Heavy-Light Decomposition (Dadalto)	95
4.13	LCA	98
4.14	Min-Cut Global	100
4.15	Strongly Connected Components	102
4.16	Transitive Closure	104
4.17	Tree Isomorphism	106
5	Misc	108
5.1	Bit tricks	108
5.2	DP Optimization - Binary Search	109
5.3	DP Optimization - CHT	111
5.4	DP Optimization - Knuth	113
5.5	MOs	115
5.6	MOs - Tree (Edge Query)	117
5.7	MOs - Tree (Vertex Query)	120
5.8	MOs - Hilbert	123
5.9	Ternary Search (continuous)	125
5.10	Ternary Search (discrete)	126
6	Combinatorial	127
6.1	Binomial - Pascal's Triangle	127
6.2	Binomial - Lucas' Theorem	128
6.3	Grundy	129
6.4	Surreal Numbers	130
7	Number Theory	133
7.1	General Chinese Remainder Theorem	133
7.2	General Chinese Remainder Theorem - System	134
7.3	Euclid	136
7.4	Factorization (Pollard rho)	137
7.5	Modular Inverse	138
7.6	Large modular mult/pow	139
7.7	Modular Arithmetic	140
7.8	Phi	141
7.9	Primality Test	143
7.10	Sieve	144
8	Numerical	145
8.1	Big Int	145
8.2	FFT	158
8.3	Fraction	160
8.4	Integration	162
8.5	linalg	163
8.6	NTT	167
8.7	Simplex	169

9	String	176
9.1	Aho Corasick	176
9.2	Hash	182
9.3	KMP	183
9.4	Suffix Array	185
9.5	Suffix Tree	187

1 Contest

1.1 Header

```
#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

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

```

5d1131  #include "../contest/header.hpp"
d41d8c
d41d8c  /*
fc6e4f    2D Segment Tree:
924136    2D point update and 2D range query in  $O(\log(n) \cdot \log(m))$  per
035979    operation,
da2eb6    where  $n$  is the number of rows and  $m$  is the number of columns.
d41d8c
65b4df    Uses  $O(\text{Num\_Updates} \cdot \log(n) \cdot \log(m))$  memory.
d41d8c
765bf7    Given as an example using gcd.
d41d8c
b95cae    Usage:
5b40a6    See main and comments.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
d41d8c  // Number of rows and columns (inner nodes need to know this).
14e0a7  int n, m;
d41d8c
fc0cf7  struct nodes
f95b70  {
34833c    ll val;
af0912    nodes *left, *right;
2fd56f    nodes() : val(0), left(NULL), right(NULL) {}
d41d8c
d41d8c  // Update leaf tree.
7acf5a  void update(int l, int r, int a, ll x)
f95b70  {
bd3398    if (l == r)
c43fe0        val = x;
2954e9    else
f95b70        {
ae007b        int mid = (l + r) / 2;
a49729        if (a <= mid)
ff3239            (left ? left : (left = new nodes()))->update(l, mid, a, x);
2954e9        else
ee984e            (right ? right : (right = new nodes()))->update(mid + 1, r, a
, x);
d41d8c
6a85db        val = __gcd(left ? left->val : 0, right ? right->val : 0);
cbb184    }
cbb184  }
d41d8c
d41d8c  // Update non-leaf tree by joining the two child trees along the

```



```

d41d8c // modified path.
fe4c66 void updateb(int l, int r, int a, ll x, nodes *o, nodes *p)
f95b70 {
bd3398     if (l == r)
8eb30c         val = __gcd(o ? o->val : 0, p ? p->val : 0);
2954e9     else
f95b70     {
ae007b         int mid = (l + r) / 2;
a49729         if (a <= mid)
0db51d             (left ? left : (left = new nodes()))->updateb(l, mid, a, x, o
? o->left : NULL, p ? p->left : NULL);
2954e9         else
58b90e             (right ? right : (right = new nodes()))->updateb(mid + 1, r,
a, x, o ? o->right : NULL, p ? p->right : NULL);
d41d8c         val = __gcd(o ? o->val : 0, p ? p->val : 0);
cbb184     }
cbb184 }
d41d8c
1586df ll get(int l, int r, int a, int b)
f95b70 {
47234b     if (l == a && r == b)
d943f4         return val;
2954e9     else
f95b70     {
ae007b         int mid = (l + r) / 2;
f890f2         if (b <= mid)
ac57ce             return left ? left->get(l, mid, a, b) : 0;
a54f0c         else if (a > mid)
1c7837             return right ? right->get(mid + 1, r, a, b) : 0;
2954e9         else
61c9d1             return __gcd(left ? left->get(l, mid, a, mid) : 0, right ?
right->get(mid + 1, r, mid + 1, b) : 0);
cbb184     }
cbb184 }
2145c1 };
d41d8c
7fee06 struct nodef
f95b70 {
848f8b     nodes *val;
16ebe5     nodef *left, *right;
da624e     nodef() : left(NULL), right(NULL) { val = new nodes(); }
d41d8c
0ab567 void update(int l, int r, int a, int b, ll x)
f95b70 {
bd3398     if (l == r)
b7e6e3         val->update(0, m - 1, b, x);
2954e9     else
f95b70     {
ae007b         int mid = (l + r) / 2;

```

```

a49729         if (a <= mid)
e5a0ea             (left ? left : (left = new nodef()))->update(l, mid, a, b, x)
;
2954e9         else
25fae0             (right ? right : (right = new nodef()))->update(mid + 1, r, a
, b, x);
d41d8c
c63cc8         val->updateb(0, m - 1, b, x, left ? left->val : NULL, right ?
right->val : NULL);
cbb184     }
cbb184     }
d41d8c
0bc50b     ll get(int l, int r, int a, int b, int c, int d)
f95b70     {
47234b         if (l == a && r == b)
2b5ee1             return val->get(0, m - 1, c, d);
2954e9         else
f95b70         {
ae007b             int mid = (l + r) / 2;
f890f2             if (b <= mid)
4668df                 return left ? left->get(l, mid, a, b, c, d) : 0;
a54f0c             else if (a > mid)
d6a980                 return right ? right->get(mid + 1, r, a, b, c, d) : 0;
2954e9             else
90c1d1                 return __gcd(left ? left->get(l, mid, a, mid, c, d) : 0,
right ? right->get(mid + 1, r, mid + 1, b, c, d) : 0);
cbb184         }
cbb184     }
2145c1 };
d41d8c
13a4b1 int main(void)
f95b70 {
8dee13     ll a;
66f1d8     nodef *root = new nodef();
0dc26f     int q, tp, x, y, z, w;
cdcf55     scanf("%d %d %d", &n, &m, &q);
a953ae     while (q--)
f95b70     {
64d8be         scanf("%d", &tp);
abc772         if (tp == 1)
f95b70         {
425c33             scanf("%d %d %lld", &x, &y, &a);
dbf0ee             root->update(0, n - 1, x, y, a);
cbb184         }
2954e9         else
f95b70         {
bb2481             scanf("%d %d %d %d", &x, &y, &z, &w);
c0075a             printf("%lld\n", root->get(0, n - 1, x, z, y, w));
cbb184         }
cbb184     }

```

cbb184 }

Full file hash: 1f17b0

2.2 BIT

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
0e8577     BIT: element update, range sum query and sum lower_bound in
4a7c08     O(log(N)).
2716d1     Represents an array of elements in range [1, N].
c4c9bd */
d41d8c
4fce64 template <class T>
2d55ba struct bit
f95b70 {
b9a249     int n, LOGN;
2262a2     vector<T> val;
695052     bit(int _n) : n(_n), LOGN(log2(n + 1)), val(_n + 1, 0) {}
d41d8c
d41d8c     // val[pos] += x
b29da0 void update(int pos, T x)
f95b70 {
259a9f     for (int i = pos; i <= n; i += -i & i)
ac55c8         val[i] += x;
cbb184 }
d41d8c
d41d8c     // sum of range [1, pos]
8a835d T query(int pos)
f95b70 {
56622d     T retv = 0;
ac430c     for (int i = pos; i > 0; i -= -i & i)
106953         retv += val[i];
6272cf     return retv;
cbb184 }
d41d8c
d41d8c     // min pos such that sum of [1, pos] >= sum, or n + 1 if none
d41d8c     // exists.
79d23b int lower_bound(T x)
f95b70 {
501ce1     T sum = 0;
bec7a6     int pos = 0;
d41d8c
51d707     for (int i = LOGN; i >= 0; i--)
0328f7         if (pos + (1 << i) <= n && sum + val[pos + (1 << i)] < x)
420193             sum += val[pos += (1 << i)];
d41d8c
7e21de     return pos + 1; // pos will have position of largest value
d41d8c         // less than x.
cbb184 }
2145c1 };
Full file hash: 6acfcc

```

2.3 BIT2D

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
21fcf0     BIT: element update, range sum query in  $O(\log(n) * \log(m))$ .
849273     This can also be generalized for 3d.
a6cfe6     Represents a matrix of elements in range [1 ... n][1 ... m].
c4c9bd */
d41d8c
4fce64 template <class T>
f6f3a7 struct bit2d
f95b70 {
14e0a7     int n, m;
f7ea55     vector<vector<T>> val;
9c8214     bit2d(int _n, int _m) : n(_n), m(_m), val(_n + 1, vector<T>(_m + 1,
    0)) {}
d41d8c
d41d8c     // val[i][j] += x
4460cb     void update(int r, int c, T x)
f95b70     {
9e45d9         for (int i = r; i <= n; i += -i & i)
13d333             for (int j = c; j <= m; j += -j & j)
ff237f                 val[i][j] += x;
cbb184     }
d41d8c
d41d8c     // sum of positions (1 ... r, 1 ... c)
450f85     T query(int r, int c)
f95b70     {
56622d         T retv = 0;
bc7409         for (int i = r; i > 0; i -= -i & i)
d53722             for (int j = c; j > 0; j -= -j & j)
86df71                 retv += val[i][j];
6272cf         return retv;
cbb184     }
d41d8c
d41d8c     // sum of positions (ri ... rf, ci ... cf). (1 <= ri <= rf <= n)
d41d8c     // and (1 <= ci <= cf <= m). TODO: test me.
bdc664     T query_rect(int ri, int ci, int rf, int cf)
f95b70     {
6072bc         return query(rf, cf) - query(rf, ci - 1) - query(ri - 1, cf) +
    query(ri - 1, ci - 1);
cbb184     }
2145c1 };
Full file hash: a4a33c

```

2.4 Dynamic Segment Tree

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
699080     Segment tree with dynamic memory allocation and arbitrary
4685fc     interval.
91eb69     Every operation is  $O(\log(r-l))$ 
6449d0     Uses  $O(\min(r-l, n \cdot \log(r-l)))$  memory, where  $n$  is the number of
1ceddc     insertions.
d41d8c
ca2095     Constraints:
3dcfba     Segment tree range  $[l, r]$  must be such that  $0 \leq l \leq r$ .
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
4fce64 template<class T>
e4accb struct node
f95b70 {
f48ea0     T val;
af32d9     node *left, *right;
d41d8c
995125     T get(int l, int r, int a, int b)
f95b70     {
47234b         if (l == a && r == b)
d943f4             return val;
814ad2         int mid = (l + 0ll + r) / 2;
f890f2         if (b <= mid)
ac57ce             return left ? left->get(l, mid, a, b) : 0;
a54f0c         else if (a > mid)
1c7837             return right ? right->get(mid + 1, r, a, b) : 0;
2954e9         else
9b1cb1             return (left ? left->get(l, mid, a, mid) : 0) + (right ? right
->get(mid + 1, r, mid + 1, b) : 0);
cbb184     }
d41d8c
14d5ea     void update(int l, int r, int a, T x)
f95b70     {
bd3398         if (l == r)
c43fe0             val = x;
2954e9         else
f95b70         {
814ad2             int mid = (l + 0ll + r) / 2;
a49729             if (a <= mid)
1ec55a                 (left ? left : (left = new node()))->update(l, mid, a, x);
2954e9             else
92fe63                 (right ? right : (right = new node()))->update(mid + 1, r, a,
x);
d41d8c

```

```
dd51dd      val = (left ? left->val : 0) + (right ? right->val : 0);
cbb184      }
cbb184      }
2145c1    };
d41d8c
Full file hash: eef0d8
```

2.5 Linear Container

```

5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
20bead     Line Container (most common for convex hull trick). Amortized
2ffe37     O(log N) per operation.
d7b246     Container where you can add lines of the form  $kx+m$ , and query
2d624c     maximum values at points  $x$ .
dc45cd     Useful for dynamic programming.
d41d8c
1d1558     Source: https://github.com/kth-competitive-programming/kactl/
c12210     blob/master/content/contest/template.cpp
c4c9bd */
d41d8c
3fe318 struct line
f95b70 {
3e2604     mutable ll k, m, p;
889941     bool operator<(const line &o) const { return k < o.k; }
abfd1f     bool operator<(ll x) const { return p < x; }
2145c1 };
d41d8c
0c8ce5 struct line_container : multiset<line, less<>>
f95b70 {
d41d8c     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
f5e3e7     const ll inf = LLONG_MAX;
d41d8c
9608c5     ll div(ll a, ll b)
f95b70     { // floored division
353cf0         return a / b - ((a ^ b) < 0 && a % b);
cbb184     }
d41d8c
9c092f     bool isect(iterator x, iterator y)
f95b70     {
f959d1         if (y == end())
f95b70         {
09a75e             x->p = inf;
d1fe4d             return false;
cbb184         }
3cca77         if (x->k == y->k)
83e301             x->p = x->m > y->m ? inf : -inf;
2954e9         else
b4284e             x->p = div(y->m - x->m, x->k - y->k);
870ec6         return x->p >= y->p;
cbb184     }
d41d8c
928f4b     void add(ll k, ll m)
f95b70     {
116e6c         auto z = insert({k, m, 0}), y = z++, x = y;
2d9d80         while (isect(y, z))

```



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

2.6 Merge Sort Tree

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
abc8b2     Merge Sort Tree:
4196f6         Build segment tree where each node stores a sorted version
ea708f         of the underlying range.
dc8487         O(n log n) to build, O(log^2 n) each query (in this case).
2a5aa7         This example uses kth number in interval queries.
d41d8c
b95cae     Usage:
67c30a         In this example, instead of building the merge sort tree with
656847         the given vector (e.g. {1, 5, 2, 6, 3, 7, 4}), we sort a
d2e069         vector of indices by their value in the vector
6bf72e         (e.g. {0, 2, 4, 6, 1, 3, 5} for the vector above).
d41d8c
5cdd7f         This way, each node in the tree is responsible for a range of
1ffb1e         sorted elements in the vector and we can ask it how many of
d5585e         those have indices in the range [a, b].
c4c9bd */
d41d8c
4ee394 #define left(i) ((i) << 1)
56e5cf #define right(i) (((i) << 1) + 1)
d41d8c
05aea6 struct merge_sort_tree
f95b70 {
990cc8     vector<int> v; // original vector.
dc0a56     vector<vector<int>> val;
1beda1     vector<int> indices; // indices sorted by value in v.
d41d8c
1b6455     merge_sort_tree(vector<int> v) : v(v), val(4 * (sz(v) + 1))
f95b70     {
9ad08e         for (int i = 0; i < sz(v); i++)
f2b3cb             indices.push_back(i);
788c05         sort(all(indices), [&v](int i, int j) { return v[i] < v[j]; });
d41d8c
5ba763         build(1, 0, sz(v) - 1);
cbb184     }
d41d8c
b7317b     void build(int id, int l, int r)
f95b70     {
bd3398         if (l == r)
1b16b1             val[id].push_back(indices[l]);
2954e9         else
f95b70         {
ae007b             int mid = (l + r) / 2;
c7a0a2             build(left(id), l, mid), build(right(id), mid + 1, r);
0d740a             val[id] = vector<int>(r - l + 1);
2c5d2b             merge(all(val[left(id)]), all(val[right(id)]), val[id].begin())

```

```

;
cbb184    }
cbb184    }
d41d8c
d41d8c    // How many elements in this node have indices in the range [a, b]
a5ce6e    int count_interval(int id, int a, int b)
f95b70    {
a4c94b        return (int)(upper_bound(all(val[id]), b) - lower_bound(all(val[
    id]), a));
cbb184    }
d41d8c
beafee    int get(int id, int l, int r, int a, int b, int x)
f95b70    {
bd3398        if (l == r)
bc45ca            return v[val[id].back()];
ae007b        int mid = (l + r) / 2;
7c12bf        int lcount = count_interval(left(id), a, b);
87eaf6        if (lcount >= x)
7a388b            return get(left(id), l, mid, a, b, x);
2954e9        else
f29b51            return get(right(id), mid + 1, r, a, b, x - lcount);
cbb184    }
d41d8c
5c9a2a    int kth(int a, int b, int k)
f95b70    {
492d6c        return get(1, 0, sz(v) - 1, a, b, k);
cbb184    }
2145c1 };
Full file hash: 284e0c

```

2.7 Min Queue

```

5d1131 #include "../contest/header.hpp"
d41d8c /*
958401     max(min) queue with O(1) get_max(min).
d41d8c
f67dcb     Tips:
c53808         - Useful for sliding window 1D and 2D.
e41836         - For 2D problems, you will need to pre-compute another
c712f0           matrix, by making a row-wise traversal, and calculating the
c0a568           min/max value beginning in each cell. Then you just make a
dffe79           column-wise traverse as they were each an independent array.
c4c9bd */
d41d8c
8f0a66 struct max_queue
f95b70 {
84841a     queue<ll> q;
889d23     deque<ll> s;
d41d8c
dbb27b     int size()
f95b70     {
593f12         return (int)q.size();
cbb184     }
d41d8c
a1fe24     void push(ll val)
f95b70     {
d41d8c         // while (!s.empty() && s.back() > val) -> for a min_queue
1cb658         while (!s.empty() && s.back() < val) // for a max_queue
342ca4             s.pop_back();
fcc849         s.push_back(val);
d41d8c
380c99         q.push(val);
cbb184     }
d41d8c
d99fc4     void pop()
f95b70     {
7a8432         ll u = q.front();
833270         q.pop();
d41d8c
de7036         if (!s.empty() && s.front() == u)
784c93             s.pop_front();
cbb184     }
d41d8c
ba28bf     ll get_max()
f95b70     {
eccd4b         return s.front(); // same for min and max queue
cbb184     }
d41d8c
2145c1 };
Full file hash: 82549d

```

2.8 Persistent Segment Tree

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
a032aa     Persistent Segment Tree:
115cd2         Segment tree that stores all previous versions of itself.
91eb69         Every operation is  $O(\log(r-l))$ 
ad671a         Uses  $O(n \times \log(r-l))$  memory, where  $n$  is the number of updates.
d41d8c
b95cae     Usage:
4e368d         A new root is created for every persistent update (p_update)
db6014         and returned.
92aa65         Queries can be performed on any root as if it were a usual
3b27c5         segment tree.
61a20d         You should keep a list of roots. Something like:
072987             vector<node*> roots = {new node()};
e02688             roots.push_back(p_update(roots.back(), 0,
d75536                 2*MAXV, a[i] + MAXV, v + 1));
d41d8c
ca2095     Constraints:
3dcfba         Segment tree range  $[l, r]$  must be such that  $0 \leq l \leq r$ .
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
e4accb struct node
f95b70 {
97f03f     int val;
af32d9     node *left, *right;
d41d8c
1f6b0f     node(int x=0) : val(x), left(NULL), right(NULL) {}
2f77b9     node(node *l, node *r) : left(l), right(r) { val = (left ? left->
    val : 0) + (right ? right->val : 0); }
d41d8c
f219f1     int get(int l, int r, int a, int b)
f95b70     {
47234b         if (l == a && r == b)
d943f4             return val;
814ad2         int mid = (l + 0ll + r) / 2;
f890f2         if (b <= mid)
ac57ce             return left ? left->get(l, mid, a, b) : 0;
a54f0c         else if (a > mid)
1c7837             return right ? right->get(mid + 1, r, a, b) : 0;
2954e9         else
9b1cb1             return (left ? left->get(l, mid, a, mid) : 0) + (right ? right
    ->get(mid + 1, r, mid + 1, b) : 0);
cbb184     }
2145c1 };
d41d8c

```

```
63f202 node *p_update(node *prev, int l, int r, int a, int x)
f95b70 {
bd3398     if (l == r)
13478f         return new node(x);
d41d8c
814ad2     int mid = (l + 0ll + r) / 2;
a49729     if (a <= mid)
b73799         return new node(p_update(prev ? prev->left : NULL, l, mid, a, x),
prev ? prev->right : NULL);
2954e9     else
460332         return new node(p_update(prev ? prev->left : NULL, p_update(prev ? prev->
right : NULL, mid + 1, r, a, x));
cbb184 }
d41d8c
Full file hash: 707f69
```

2.9 Segment Tree

```

2b74fa #include <bits/stdc++.h>
ca417d using namespace std;
d41d8c
d41d8c /*
6f561b     Segment Tree with Lazy updates:
d8b1dc         Range update and range query in O(log(MAX_RANGE))
c329b0         Binary search on tree in O(log(MAX_RANGE))
3d7beb         Given as an example since it is not worth it to copy a
c1759f         generic tree during a contest.
d41d8c
e3c955     Solves: https://codeforces.com/contest/1179/problem/C
c4c9bd */
d41d8c
ab0dbf #define MAX_RANGE 1123456
d41d8c
fd87fe int val[4 * MAX_RANGE];
802d92 int delta[4 * MAX_RANGE];
d41d8c
4ee394 #define left(i) ((i) << 1)
56e5cf #define right(i) (((i) << 1) + 1)
d41d8c
0379af void prop(int id, int l, int r)
f95b70 {
cfd4b4     if (l != r)
f95b70     {
d41d8c         // Updates need to be numerically stackable (e.g. not valid
d41d8c         // to have a list of updates).
df541b         delta[left(id)] += delta[id];
966351         delta[right(id)] += delta[id];
cbb184     }
d41d8c
21c2c8     val[id] += delta[id]; // Node value needs to be obtainable without
d41d8c         // propagating all the way to root.
0a8860     delta[id] = 0;
cbb184 }
d41d8c
d41d8c // Sum x in all elements in range [a, b].
f2b4f2 void update(int id, int l, int r, int a, int b, int x)
f95b70 {
addc1f     if (a == l && b == r)
f95b70     {
d50197         delta[id] += x;
b62cfe         prop(id, l, r);
cbb184     }
2954e9     else
f95b70     {
b62cfe         prop(id, l, r);
ae007b         int mid = (l + r) / 2;

```

```

f890f2     if (b <= mid)
f95b70     {
6dbd37         update(left(id), l, mid, a, b, x);
384ec5         prop(right(id), mid + 1, r);
cbb184     }
a54f0c     else if (a > mid)
f95b70     {
859d13         update(right(id), mid + 1, r, a, b, x);
221ad0         prop(left(id), l, mid);
cbb184     }
2954e9     else
f95b70     {
fc79c7         update(left(id), l, mid, a, mid, x);
04c83e         update(right(id), mid + 1, r, mid + 1, b, x);
cbb184     }
d41d8c
caf644     val[id] = min(val[left(id)], val[right(id)]);
cbb184     }
cbb184 }
d41d8c
d41d8c // Get the minimum value in range [a, b].
9fed20 int get(int id, int l, int r, int a, int b)
f95b70 {
b62cfe     prop(id, l, r);
addc1f     if (a == l && b == r)
a0328b         return val[id];
2954e9     else
f95b70     {
ae007b         int mid = (l + r) / 2;
f890f2         if (b <= mid)
c55f80             return get(left(id), l, mid, a, b);
a54f0c         else if (a > mid)
26dd34             return get(right(id), mid + 1, r, a, b);
2954e9         else
5e3fad             return min(get(left(id), l, mid, a, mid), get(right(id), mid +
1, r, mid + 1, b));
cbb184     }
cbb184 }
d41d8c
d41d8c // Find index of rightmost element which is less than x. (works
d41d8c // because this is a seg of min)
0529b3 int bsearch(int id, int l, int r, int x)
f95b70 {
b62cfe     prop(id, l, r);
d41d8c
bd3398     if (l == r)
f7d2ed         return (val[id] < x) ? l : -1;
2954e9     else
f95b70     {
ae007b         int mid = (l + r) / 2;

```



```

221ad0     prop(left(id), l, mid);
384ec5     prop(right(id), mid + 1, r);
f01b35     if (val[right(id)] < x)
018a94         return bsearch(right(id), mid + 1, r, x);
2954e9     else
bad725         return bsearch(left(id), l, mid, x);
cbb184     }
cbb184 }
d41d8c
1037bf #define MAXN 312345
d41d8c
a58cd5 int a[MAXN];
c4b25f int b[MAXN];
d41d8c
13a4b1 int main(void)
f95b70 {
b067b3     int n, m, q, tp, x, y;
d69917     scanf("%d %d", &n, &m);
5359f3     for (int i = 1; i <= n; i++)
f95b70     {
9376f3         scanf("%d", &a[i]);
49e934         update(1, 1, 1000000, 1, a[i], -1);
cbb184     }
d41d8c
8eae24     for (int i = 1; i <= m; i++)
f95b70     {
264aeb         scanf("%d", &b[i]);
472fcc         update(1, 1, 1000000, 1, b[i], 1);
cbb184     }
d41d8c
4aaeab     scanf("%d", &q);
a953ae     while (q--)
f95b70     {
960099         scanf("%d %d %d", &tp, &x, &y);
abc772         if (tp == 1)
f95b70         {
996a9b             update(1, 1, 1000000, 1, a[x], 1);
e603e6             a[x] = y;
28cfa0             update(1, 1, 1000000, 1, a[x], -1);
cbb184         }
2954e9         else
f95b70         {
8dbabe             update(1, 1, 1000000, 1, b[x], -1);
0464a9             b[x] = y;
bc18aa             update(1, 1, 1000000, 1, b[x], 1);
cbb184         }
d41d8c
584906         int tmp = bsearch(1, 1, 1000000, 0);
d41d8c
d41d8c         // Test of get and bsearch. Make sure all to the right are

```

```
d41d8c    // non-negative.
5a5bec    if (tmp != 1000000)
5df0f6        assert(get(1, 1, 1000000, tmp == -1 ? 1 : (tmp + 1), 1000000)
    >= 0);
c3e568    if (tmp != -1)
1d95f2        assert(get(1, 1, 1000000, tmp, tmp) < 0);
d41d8c
b03a7a    printf("%d\n", tmp);
cbb184    }
cbb184    }
Full file hash: 90a905
```

2.10 Lazy Segment Tree

```

2b74fa #include <bits/stdc++.h>
ca417d using namespace std;
d41d8c
d41d8c /*
6f561b     Segment Tree with Lazy updates:
d8b1dc         Range update and range query in O(log(MAX_RANGE))
c329b0         Binary search on tree in O(log(MAX_RANGE))
3d7beb         Given as an example since it is not worth it to copy a
c1759f         generic tree during a contest.
d41d8c
e3c955     Solves: https://codeforces.com/contest/1179/problem/C
c4c9bd */
d41d8c
ab0dbf #define MAX_RANGE 1123456
d41d8c
fd87fe int val[4 * MAX_RANGE];
802d92 int delta[4 * MAX_RANGE];
d41d8c
4ee394 #define left(i) ((i) << 1)
56e5cf #define right(i) (((i) << 1) + 1)
d41d8c
0379af void prop(int id, int l, int r)
f95b70 {
cfd4b4     if (l != r)
f95b70     {
d41d8c         // Updates need to be numerically stackable (e.g. not valid
d41d8c         // to have a list of updates).
df541b         delta[left(id)] += delta[id];
966351         delta[right(id)] += delta[id];
cbb184     }
d41d8c
21c2c8     val[id] += delta[id]; // Node value needs to be obtainable without
d41d8c         // propagating all the way to root.
0a8860     delta[id] = 0;
cbb184 }
d41d8c
d41d8c // Sum x in all elements in range [a, b].
f2b4f2 void update(int id, int l, int r, int a, int b, int x)
f95b70 {
addc1f     if (a == l && b == r)
f95b70     {
d50197         delta[id] += x;
b62cfe         prop(id, l, r);
cbb184     }
2954e9     else
f95b70     {
b62cfe         prop(id, l, r);
ae007b         int mid = (l + r) / 2;

```

```

f890f2     if (b <= mid)
f95b70     {
6dbd37         update(left(id), l, mid, a, b, x);
384ec5         prop(right(id), mid + 1, r);
cbb184     }
a54f0c     else if (a > mid)
f95b70     {
859d13         update(right(id), mid + 1, r, a, b, x);
221ad0         prop(left(id), l, mid);
cbb184     }
2954e9     else
f95b70     {
fc79c7         update(left(id), l, mid, a, mid, x);
04c83e         update(right(id), mid + 1, r, mid + 1, b, x);
cbb184     }
d41d8c     val[id] = min(val[left(id)], val[right(id)]);
cbb184 }
cbb184 }
d41d8c
d41d8c // Get the minimum value in range [a, b].
9fed20 int get(int id, int l, int r, int a, int b)
f95b70 {
b62cfe     prop(id, l, r);
addc1f     if (a == l && b == r)
a0328b         return val[id];
2954e9     else
f95b70     {
ae007b         int mid = (l + r) / 2;
f890f2         if (b <= mid)
c55f80             return get(left(id), l, mid, a, b);
a54f0c         else if (a > mid)
26dd34             return get(right(id), mid + 1, r, a, b);
2954e9         else
5e3fad             return min(get(left(id), l, mid, a, mid), get(right(id), mid +
1, r, mid + 1, b));
cbb184     }
cbb184 }
d41d8c
d41d8c // Find index of rightmost element which is less than x. (works
d41d8c // because this is a seg of min)
0529b3 int bsearch(int id, int l, int r, int x)
f95b70 {
b62cfe     prop(id, l, r);
d41d8c
bd3398     if (l == r)
f7d2ed         return (val[id] < x) ? l : -1;
2954e9     else
f95b70     {
ae007b         int mid = (l + r) / 2;

```

```

221ad0     prop(left(id), l, mid);
384ec5     prop(right(id), mid + 1, r);
f01b35     if (val[right(id)] < x)
018a94         return bsearch(right(id), mid + 1, r, x);
2954e9     else
bad725         return bsearch(left(id), l, mid, x);
cbb184     }
cbb184 }
d41d8c
1037bf #define MAXN 312345
d41d8c
a58cd5 int a[MAXN];
c4b25f int b[MAXN];
d41d8c
13a4b1 int main(void)
f95b70 {
b067b3     int n, m, q, tp, x, y;
d69917     scanf("%d %d", &n, &m);
5359f3     for (int i = 1; i <= n; i++)
f95b70     {
9376f3         scanf("%d", &a[i]);
49e934         update(1, 1, 1000000, 1, a[i], -1);
cbb184     }
d41d8c
8eae24     for (int i = 1; i <= m; i++)
f95b70     {
264aeb         scanf("%d", &b[i]);
472fcc         update(1, 1, 1000000, 1, b[i], 1);
cbb184     }
d41d8c
4aaeab     scanf("%d", &q);
a953ae     while (q--)
f95b70     {
960099         scanf("%d %d %d", &tp, &x, &y);
abc772         if (tp == 1)
f95b70         {
996a9b             update(1, 1, 1000000, 1, a[x], 1);
e603e6             a[x] = y;
28cfa0             update(1, 1, 1000000, 1, a[x], -1);
cbb184         }
2954e9         else
f95b70         {
8dbabe             update(1, 1, 1000000, 1, b[x], -1);
0464a9             b[x] = y;
bc18aa             update(1, 1, 1000000, 1, b[x], 1);
cbb184         }
d41d8c
584906         int tmp = bsearch(1, 1, 1000000, 0);
d41d8c
d41d8c         // Test of get and bsearch. Make sure all to the right are

```

```
d41d8c    // non-negative.
5a5bec    if (tmp != 1000000)
5df0f6        assert(get(1, 1, 1000000, tmp == -1 ? 1 : (tmp + 1), 1000000)
    >= 0);
c3e568    if (tmp != -1)
1d95f2        assert(get(1, 1, 1000000, tmp, tmp) < 0);
d41d8c
b03a7a    printf("%d\n", tmp);
cbb184    }
cbb184    }
Full file hash: 90a905
```

2.11 Key Treap

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
1977a5     Treap:
0d1904         This treap implements something like a c++ set with additional
b36db0         operations: find the k-th element and count elements less than
2fe20c         a given value.
d41d8c
4c88cf     Time: O(log N) per operation.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
41c55a namespace treap
f95b70 {
e4accb struct node
f95b70 {
97f03f     int val; // node key.
ee1179     int p;   // node heap priority.
59afd1     int num; // node subtree size.
af32d9     node *left, *right;
d41d8c
71091e     node(int _val) : val(_val), p(rand()), num(1), left(NULL), right(
    NULL) {}
2145c1 };
d41d8c
48f3b4 int get_num(node *root)
f95b70 {
424a36     return (root == NULL) ? 0 : root->num;
cbb184 }
d41d8c
68f1eb void update_num(node *root)
f95b70 {
47a6f1     root->num = get_num(root->left) + get_num(root->right) + 1;
cbb184 }
d41d8c
afdba0 node *rotate_left(node *root)
f95b70 {
d25f1b     node *a = root;
a95379     node *b = root->right;
d41d8c
b51426     a->right = b->left;
e7e30a     b->left = a;
a5e0c3     update_num(a);
2b11db     update_num(b);
73f89f     return b;
cbb184 }
d41d8c

```

```

f17a34 node *rotate_right(node *root)
f95b70 {
d25f1b     node *a = root;
eb0328     node *b = root->left;
d41d8c
a09684     a->left = b->right;
7352c4     b->right = a;
a5e0c3     update_num(a);
2b11db     update_num(b);
73f89f     return b;
cbb184 }
d41d8c
d41d8c // Insert new node with key x in treap rooted at root if not already
d41d8c // there.
960bce node *insert(node *root, int x)
f95b70 {
0edbc9     if (root == NULL)
13478f         return new node(x);
6b2a0b     if (x > root->val)
34c9df         root->right = insert(root->right, x);
ba0dc8     else if (x < root->val)
12f5b5         root->left = insert(root->left, x);
d41d8c
622638     update_num(root);
d41d8c
4f4bcf     if (root->right && root->right->p > root->p)
04107a         root = rotate_left(root);
c93ea7     if (root->left && root->left->p > root->p)
3f3108         root = rotate_right(root);
e2fc54     return root;
cbb184 }
d41d8c
d41d8c // Remove node with key x in treap rooted at root if present.
d0ba77 node *remove(node *root, int x)
f95b70 {
0edbc9     if (root == NULL)
ea9b0a         return NULL;
6b2a0b     if (x > root->val)
fed39a         root->right = remove(root->right, x);
ba0dc8     else if (x < root->val)
6cf773         root->left = remove(root->left, x);
fb8e77     else if (root->left == NULL)
4de2d2         root = root->right;
a15580     else if (root->right == NULL)
2d4ff4         root = root->left;
386129     else if (root->left->p > root->right->p)
f95b70     {
3f3108         root = rotate_right(root);
fed39a         root->right = remove(root->right, x);
cbb184     }

```



```

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

```

2.12 Sequential Treap

```

5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
1977a5     Treap:
763e1e         A short self-balancing tree. It acts as a sequential container
88ce61         with log-time splits/joins, and is easy to augment with
2d7f2e         additional data.
d41d8c
4c88cf     Time:  $O(\log N)$  per operation.
d41d8c
ca2095     Constraints:
c1b810         Acts as a vector of size  $N$ , with positions in range  $[0, N-1]$ .
d41d8c
1d1558     Source: https://github.com/kth-competitive-programming/kactl/blob/
e1d533         master/content/data-structures/Treap.h
d41d8c
b95cae     Usage:
bfe8e6         To insert elements, create one node treaps.
396bfe         (e.g. treap::ins(root, new treap::node(x), i))
2abaf2         To augment with extra data you should mostly add stuff to the
24b033         recalc function. (e.g. to make it work like a seg tree)
03bb33         See applications for more usage examples.
c4c9bd */
d41d8c
41c55a namespace treap
f95b70 {
e4accb struct node
f95b70 {
8f5901     node *l = 0, *r = 0;
97f03f     int val;    // Any value associated with node.
ee1179     int p;    // Node heap priority.
c6aff2     int c = 1; // Node subtree size.
674490     node(int val) : val(val), p(rand()) {}
86d631     void recalc();
2145c1 };
d41d8c
853943 int cnt(node *n) { return n ? n->c : 0; }
9af082 void node::recalc() { c = cnt(l) + cnt(r) + 1; }
d41d8c
d41d8c // Apply function f on each tree node in order.
044d82 template <class F>
d5442c void each(node *n, F f)
f95b70 {
f63660     if (n)
f95b70     {
cbc351         each(n->l, f);
ed31a5         f(n->val);
f5ab50         each(n->r, f);

```

```

cbb184     }
cbb184     }
d41d8c
d41d8c     // Split treap rooted at n in two treaps containing positions [0, k)
d41d8c     // and [k,...)
de9c69     pair<node *, node *> split(node *n, int k)
f95b70     {
a020ba         if (!n)
e70a07             return {NULL, NULL};
9416bd         if (cnt(n->l) >= k) // "n->val >= k" for lower_bound(k)
f95b70         {
215a80             auto pa = split(n->l, k);
f3cfa7             n->l = pa.second;
2f09c0             n->recalc();
c05937             return {pa.first, n};
cbb184         }
2954e9         else
f95b70         {
7c23f0             auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
d37e77             n->r = pa.first;
2f09c0             n->recalc();
7af31a             return {n, pa.second};
cbb184         }
cbb184     }
d41d8c
d41d8c     // Merge treaps l and r keeping order (l first).
7f5419     node *merge(node *l, node *r)
f95b70     {
0c92a8         if (!l)
4c1f3c             return r;
6bf95d         if (!r)
792fd4             return l;
a0ade2         if (l->p > r->p)
f95b70         {
ed7b68             l->r = merge(l->r, r);
bf6a1f             l->recalc();
792fd4             return l;
cbb184         }
2954e9         else
f95b70         {
654f23             r->l = merge(l, r->l);
cda92d             r->recalc();
4c1f3c             return r;
cbb184         }
cbb184     }
d41d8c
d41d8c     // Insert treap rooted at n into position pos of treap rooted at t.
3fc637     node *ins(node *t, node *n, int pos)
f95b70     {
ca9a9f         auto pa = split(t, pos);

```

```

cc8215     return merge(merge(pa.first, n), pa.second);
cbb184 }
d41d8c
d41d8c // Remove node at position pos from treap rooted at t.
1e0b32 node *rem(node *t, int pos)
f95b70 {
abdf75     node *a, *b, *c;
cf9546     tie(a, b) = split(t, pos);
0052e9     tie(b, c) = split(b, 1);
d41d8c
625cf2     delete b;
a300e4     return merge(a, c);
cbb184 }
d41d8c
d41d8c // Example application: do a query in range [l, r].
0475c8 node *query(node *t, int l, int r)
f95b70 {
abdf75     node *a, *b, *c;
a8341d     tie(a, b) = split(t, l);
89f194     tie(b, c) = split(b, r - l + 1);
d41d8c
d41d8c     // printf("%lld\n", b->tab);
d41d8c
53aa0f     return merge(merge(a, b), c);
cbb184 }
d41d8c
d41d8c // Example application: move the range [l, r) to index k.
b51124 void move(node *&t, int l, int r, int k)
f95b70 {
abdf75     node *a, *b, *c;
a8341d     tie(a, b) = split(t, l);
e81a2b     tie(b, c) = split(b, r - l);
1527bb     if (k <= l)
eeb6c2         t = merge(ins(a, b, k), c);
2954e9     else
646d6a         t = merge(a, ins(c, b, k - r));
cbb184 }
cbb184 } // namespace treap
Full file hash: 02c35c

```

2.13 Union Find

```
5d1131 #include "../contest/header.hpp"
d41d8c
10cc9e struct union_find
f95b70 {
fb553b     vector<int> p, size;
aa0179     union_find(int n) : p(n), size(n, 1)
f95b70     {
9193a3         iota(p.begin(), p.end(), 0);
cbb184     }
d41d8c
7f9f53     int find(int a)
f95b70     {
0fc55f         return (p[a] == a) ? a : (p[a] = find(p[a]));
cbb184     }
d41d8c
d72862     void join(int a, int b)
f95b70     {
bca228         a = find(a);
b884aa         b = find(b);
ae993e         if (a == b)
505b97             return;
9cf8f0         if (size[a] < size[b])
2574c6             swap(a, b);
264436         p[b] = a;
60c97b         size[a] += size[b];
cbb184     }
2145c1 };
d41d8c
Full file hash: bb32ca
```

3 Geometry

3.1 2D

```

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

```

```

d41d8c
d41d8c // angle to x-axis in interval [-pi, pi]
9073ff double angle() const { return atan2(y, x); }
d41d8c
d41d8c // makes dist()==1 (unit vector).
6f5d42 point<double> unit() const { return *this / dist(); }
d41d8c
d41d8c // rotates +90 degrees around origin.
200c8f P perp() const { return P(-y, x); }
d41d8c
d41d8c // perpendicular unit vector.
567be8 point<double> normal() const { return perp().unit(); }
d41d8c
d41d8c // returns point rotated 'a' radians ccw around the origin.
82fcdd point<double> rotate(double a) const
f95b70 {
80d6a0     return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
cbb184 }
d41d8c
d41d8c // Returns projection of vector p on this vector.
f1ed33 point<double> proj(P p) const
f95b70 {
e0b502     double d = (double)dot(p);
38df6a     return point<double>((double)x * d, (double)y * d) / (double)
    dist2();
cbb184 }
d41d8c
d41d8c // Angle between the vectors in interval [-pi, pi]. Positive if p
d41d8c // is ccw from this.
8ad6e5 double angle(P p) const { return p.rotate(-angle()).angle(); }
2145c1 };
d41d8c
d41d8c // Solves the linear system {a * x + b * y = e
d41d8c //                               {c * x + d * y = f
d41d8c // Returns {1, {x, y}} if solution is unique, {0, {0,0}} if no
d41d8c // solution and {-1, {0,0}} if infinite solutions.
d41d8c // If using integer function type, this will give wrong answer if
d41d8c // answer is not integer.
d41d8c // TODO: test me with integer and non-integer.
4fce64 template <class T>
562c39 pair<int, point<T>> linear_solve2(T a, T b, T c, T d, T e, T f)
f95b70 {
468cb9     point<T> retv;
256940     T det = a * d - b * c;
d41d8c
57f40d     if (det == 0) // Maybe do EPS compare if using floating point.
f95b70     {
cdd981         if (b * f == d * e && a * f == c * e)
3d7337             return {-1, point<T>()};
37dde3         return {0, point<T>()};

```

```

cbb184     }
d41d8c
d41d8c     // In case solution needs to be integer, use something like the
d41d8c     // line below.
d41d8c     // assert((e * d - f * b) % det == 0 &&
d41d8c     //         (a * f - c * e) % det == 0);
d41d8c
848480     return {1, point<T>((e * d - f * b) / det, (a * f - c * e) / det)};
cbb184 }
d41d8c
d41d8c     // Represents line segments defined by two points.
4fce64     template <class T>
4b2ec6     struct segment
f95b70     {
5dcf91         typedef point<T> P;
efb78f         P pi, pf; // Initial and final points.
d41d8c
a76c62         explicit segment(P a = P(), P b = P()) : pi(a), pf(b) {}
d41d8c
d41d8c         // Distance from this segment to a given point.
d41d8c         // ***IMPORTANT*** DOES NOT WORK FOR LONG LONG IF X > 1000.
325177         double dist(P p)
f95b70         {
58fd41             if (pi == pf)
adefd2                 return (p - pi).dist();
96a4f0             auto d = (pf - pi).dist2();
ff5b41             auto t = min(d, max((T)0, (p - pi).dot(pf - pi)));
0b5de3             return ((p - pi) * d - (pf - pi) * t).dist() / (double)d;
cbb184         }
d41d8c
d41d8c         // Checks if given point belongs to segment. Use dist(p) <= EPS
d41d8c         // instead when using point<double>.
0e3dba         bool on_segment(P p)
f95b70         {
50f719             return p.cross(pi, pf) == 0 && (pi - p).dot(pf - p) <= 0;
cbb184         }
d41d8c
d41d8c         // If a unique intersection point between the line segments exists
d41d8c         // then it is returned.
d41d8c         // If no intersection point exists an empty vector is returned.
d41d8c         // If infinitely many exist a vector with 2 elements is returned,
d41d8c         // containing the endpoints of the common line segment.
d41d8c         // The wrong position will be returned if P is point<ll> and the
d41d8c         // intersection point does not have integer coordinates.
d41d8c         // However, no problem in using it to check if intersects or not
d41d8c         // in this case (size of vector will be correct).
d41d8c         // *** IMPORTANT *** Products of **three** coordinates are used in
d41d8c         // intermediate steps so watch out for overflow if using int or
d41d8c         // long long.
f3f800         vector<P> intersect(segment rhs)

```



```

f95b70 {
9b1730     auto oa = rhs.pi.cross(rhs.pf, pi), ob = rhs.pi.cross(rhs.pf, pf)
,
1d46ec         oc = pi.cross(pf, rhs.pi), od = pi.cross(pf, rhs.pf);
d41d8c
d41d8c     // Checks if intersection is single non-endpoint point.
288e4c     if (sign(oa) * sign(ob) < 0 && sign(oc) * sign(od) < 0)
655339         return {(pi * ob - pf * oa) / (ob - oa)};
d41d8c
4c122f     set<P> s;
0373dd     if (rhs.on_segment(pi))
f07e25         s.insert(pi);
6725fe     if (rhs.on_segment(pf))
3c93ab         s.insert(pf);
3ad8fc     if (on_segment(rhs.pi))
522b2f         s.insert(rhs.pi);
f425cd     if (on_segment(rhs.pf))
d1c5a5         s.insert(rhs.pf);
d2dd66     return vector<P>(s.begin(), s.end());
cbb184 }
2145c1 };
d41d8c
d41d8c // Represents a line by its equation in the form a * x + b * y = c.
d41d8c // Can be created from two points or directly from constants.
4fce64 template <class T>
3fe318 struct line
f95b70 {
5dcf91     typedef point<T> P;
52d831     T a, b, c; // line a * x + b * y = c
d41d8c
f4f0fd     explicit line(P p1, P p2) // TODO: test me.
f95b70     {
4c2f1e         assert(!(p1 == p2));
6a88e5         a = p2.y - p1.y;
82330e         b = p1.x - p2.x;
cfae8e         c = a * p1.x + b * p1.y;
d41d8c
d41d8c         // In case of int, it is useful to scale down by gcd (e.g to
d41d8c         // use in a set).
d41d8c         // Might be useful to normalize here.
cbb184     }
d41d8c
510551     explicit line(T _a, T _b, T _c) : a(_a), b(_b), c(_c) {}
d41d8c
d41d8c     // Distance from this line to a given point. TODO: test me.
325177     double dist(P p)
f95b70     {
8c04b9         return (double)abs(a * p.x + b * p.y - c) / sqrt((double)(a * a +
        b * b));
cbb184     }

```

```

d41d8c
d41d8c // Intersects this line with another given line. See linear_solve2
d41d8c // for usage. TODO: test me.
4a5d8e pair<int, P> intersect(line rhs)
f95b70 {
6c76dc     return linear_solve2(a, b, rhs.a, rhs.b, c, rhs.c);
cbb184 }
d41d8c
d41d8c // Normalize line to c >= 0, a*a + b*b == 1. Only use with double.
050345 line normalize()
f95b70 {
22b5e2     double d = P(a, b).dist() * (c < 0 ? -1 : 1);
7c9abe     return line(a / d, b / d, c / d);
cbb184 }
d41d8c
d41d8c // Reflects point in current line
4b6b49 P reflect(P p)
f95b70 {
5ded80     P res;
d41d8c
d25fdb     res.x = ((b * b - a * a) * p.x - 2 * a * b * p.y + 2 * a * c) / (
a * a + b * b);
464e63     res.y = ((a * a - b * b) * p.y - 2 * a * b * p.x + 2 * b * c) / (
a * a + b * b);
d41d8c
b5053e     return res;
cbb184 }
2145c1 };
d41d8c
d41d8c // Represents a circle by its center and radius. Mostly only works
d41d8c // with double.
4fce64 template <class T>
0b1113 struct circle
f95b70 {
5dcf91     typedef point<T> P;
1ab228     P center;
c3df30     T r;
d41d8c
d41d8c // Intersects circle with a given line. This does not work with
d41d8c // integer types.
d41d8c // If there is no intersection, returns 0 and retv is whatever.
d41d8c // If intersection is a single point, returns 1 and retv is a pair
d41d8c // of equal points.
d41d8c // If intersection is two points, return 2 and retv is the two
d41d8c // intersection points.
d41d8c // Assume points are given in no particular order. If you really
d41d8c // need it, should be leftmost first when looking from center of
d41d8c // the circle.
ec2c6b int intersect(line<T> l, pair<P, P> &retv)
f95b70 {

```

```

800175     l = l.normalize();
f543ca     l.c -= l.a * center.x + l.b * center.y; // Recenter so that we
d41d8c           // can consider circle
d41d8c           // center in origin.
18b956     P v(l.a, l.b);
cf8231     P p0 = v * l.c; // p0 is the point in the line closest to
d41d8c           // origin.
d41d8c
2d9566     if (p0.dist() > r + EPS) // No intersection.
bb30ba         return 0;
40b0e2     else if (p0.dist() > r - EPS) // dist in [r - EPS, r + EPS] ->
d41d8c           //single point intersection at
d41d8c           // p0.
f95b70     {
de0c90         retv = {p0, p0};
6a5530         return 1;
cbb184     }
d41d8c
85b09c     double d = sqrt(r * r - l.c * l.c); // d is distance from p0
d41d8c           // to the intersection
d41d8c           // points.
c4bf3f     retv = {center + p0 + v.normal() * d, center + p0 - v.normal() *
d};
18b932     return 2;
cbb184 }
d41d8c
d41d8c // Intersects circle with another circle. This does not work with
d41d8c // integer types.
d41d8c // This assumes the circles do not have the same center. Check
d41d8c // this case if needed, can have 0 or infinite intersection
d41d8c // points.
d41d8c // If there is no intersection, returns 0 and retv is whatever.
d41d8c // If intersection is a single point, returns 1 and retv is a pair
d41d8c // of equal points.
d41d8c // If intersection is two points, return 2 and retv is the two
d41d8c // intersection points.
d41d8c // Assume points are given in no particular order. If you really
d41d8c // need it, should be leftmost first when looking from center of
d41d8c // the rhs circle.
f2bab0     int intersect(circle rhs, pair<P, P> &retv)
f95b70     {
db42cd         rhs.center = rhs.center - center;
2adf3a         int num = rhs.intersect(line<T>(2 * rhs.center.x, 2 * rhs.center.
y, rhs.center.x * rhs.center.x + rhs.center.y * rhs.center.y + r * r - rhs
.r * rhs.r), retv);
2a6a69         retv.first = retv.first + center;
e34010         retv.second = retv.second + center;
fcc01b         return num;
cbb184     }
d41d8c

```

```

d41d8c // Returns a pair of the two points on the circle whose tangent
d41d8c // lines intersect p.
d41d8c // If p lies within the circle NaN-points are returned. P is
d41d8c // intended to be Point<double>.
d41d8c // The first point is the one to the right as seen from the point
d41d8c // p towards the circle.
163627 pair<P, P> tangents(P p)
f95b70 {
75ad6b     p = p - center;
28b73b     double k1 = r * r / p.dist2();
f84c08     double k2 = sqrt(k1 - k1 * k1);
a64b03     return {center + p * k1 + p.perp() * k2, center + p * k1 - p.perp
    () * k2};
cbb184 }
d41d8c
d41d8c // Finds all the outer tangent lines between current circle and
d41d8c // 'other'.
d41d8c // Returns the points in the current circle crossed by those
d41d8c // tangents in retV1, and in retV2 the points in the circle
d41d8c // 'other'.
d41d8c // First point of each pair is one line, and second point of each
d41d8c // pair is the other.
d41d8c // IMPORTANT: You have to verify if one circle is not strictly
d41d8c // inside the other.
d41d8c // IMPORTANT: Only use with double.
d41d8c // In the case that one circle is inside the other with one
d41d8c // tangent point p, first points equals to p, and second points
d41d8c // are out of the circles and in the tangent line;
26fa58 void outter_tangents(circle other, pair<P, P> &retV1, pair<P, P> &
    retV2)
f95b70 {
799d5d     T a1 = asin((other.r - r) / (center - other.center).dist());
8f98db     T a2 = -atan2(other.center.y - center.y, other.center.x - center.
    x);
57b952     T a3 = asin(1) - a2 + a1;
d41d8c
132c23     retV1.first = P(center.x + r * cos(a3), center.y + r * sin(a3));
68d140     retV2.first = P(other.center.x + other.r * cos(a3), other.center.
    y + other.r * sin(a3));
d41d8c
d41d8c // In the case there is one tangent point (and circles are
d41d8c // external),
d41d8c // sets second point in a way that the tangent line can be
d41d8c // found.
4480f8     if (abs((center - other.center).dist() + min(r, other.r) - max(r,
    other.r)) < EPS)
f95b70     {
260261         P vec = center - retV1.first;
6eaed4         retV1.second = retV2.second = retV1.first + vec.rotate(asin(1))
    ;

```

```

cbb184    }
d41d8c
2954e9    else
f95b70    {
7c21b0        line<double> l = line<double>(center, other.center);
6ffd15        retV1.second = l.reflect(retV1.first);
b78d4b        retV2.second = l.reflect(retV2.first);
cbb184    }
cbb184    }
d41d8c
d41d8c    // Finds all the inner tangent lines between current circle
d41d8c    // and 'other'.
d41d8c    // Returns the points in the current circle crossed by those
d41d8c    // tangents in retV1, and in retV2 the points in the circle
d41d8c    // 'other'.
d41d8c    // First point of each pair is one line, and second point of each
d41d8c    // pair is the other.
d41d8c    // IMPORTANT: You have to verify if one circle does not intersect
d41d8c    // the other in more than one point (verify centers distance vs
d41d8c    // r + other.r).
d41d8c    // IMPORTANT: Only use with double.
d41d8c    // In the case that the circles intersect in one point p and are
d41d8c    // exterior to one another, points returned as first will be p,
d41d8c    // and points returned as second
d41d8c    // will be points outside the circles in the tangent line)
f8a8b2    void inner_tangents(circle other, pair<P, P> &retV1, pair<P, P> &
    retV2)
f95b70    {
d41d8c        // Point where inner tangents cross (when they are the same
d41d8c        // line, it's the point in the segment between circle centers)
575bfd        P cp = (other.center * r + center * other.r) / (r + other.r);
d41d8c
d41d8c        //Finds points for current circle
1e9e91        double u = r / (center - cp).dist();
32d48a        double angle = acos(u);
d1ca34        P vec = cp - center;
10b38f        retV1 = {center + vec.rotate(angle) * u, center + vec.rotate(-
    angle) * u};
d41d8c
d41d8c        //find points for other circle
9db06e        u = other.r / (other.center - cp).dist();
32f4fb        angle = acos(u);
0ffac3        vec = cp - other.center;
8d1129        retV2 = {other.center + vec.rotate(angle) * u, other.center + vec
    .rotate(-angle) * u};
d41d8c
d41d8c        //In the case there is one tangent point (and circles are
d41d8c        // external), sets second point in a way that the tangent line
d41d8c        // can be found.
b4987a        if (abs(r + other.r - (center - other.center).dist()) < EPS)

```

```

8a290f         retV1.second = retV2.second = cp + vec.rotate(asin(1));
cbb184     }
2145c1 };
d41d8c
d41d8c // The circumcircle of a triangle is the circle intersecting all
d41d8c // three vertices.
d41d8c // Returns the unique circle going through points A, B and C (given
d41d8c // in no particular order).
d41d8c // This assumes that the triangle has non-zero area.
d41d8c // TODO: test specifically.
11308f circle<double> circumcircle(const point<double> &A, const point<
double> &B, const point<double> &C)
f95b70 {
b10dc9     circle<double> retv;
6d2418     point<double> a = C - B, b = C - A, c = B - A;
1d9440     retv.r = a.dist() * b.dist() * c.dist() / abs(c.cross(b)) / 2;
0d1695     retv.center = A + (b * c.dist2() - c * b.dist2()).perp() / b.cross(
c) / 2;
6272cf     return retv;
cbb184 }
d41d8c
d41d8c // Returns TWO TIMES the area of the SIMPLE (non self intersecting)
d41d8c // polygon defined in pol.
d41d8c // The area is NEGATIVE if the polygon is in CLOCKWISE.
4fce64 template <class T>
9459e8 T area_polygon2(vector<point<T>> pol)
f95b70 {
3b0433     T area = 0;
76fc34     for (int i = 0; i < (int)pol.size() - 1; i++)
861f64         area += pol[i].cross(pol[i + 1]);
d41d8c
f3d2cf     area += pol[pol.size() - 1].cross(pol[0]);
d41d8c
742c67     return area;
cbb184 }
d41d8c
4fce64 template <class T>
aa651f ostream &operator<<(ostream &os, point<T> p)
f95b70 {
d80d70     return os << "(" << p.x << ", " << p.y << ")";
cbb184 }
Full file hash: 075f92

```

3.2 3D

```

5d1131 #include "../..//contest/header.hpp"
d41d8c /**
630f69  * 3D geometry operations.
1bebdd  * Status: tested, except for phi and theta
08b0d9  * Source: https://github.com/kth-competitive-programming/kactl
c4c9bd  */
d41d8c
4fce64 template <class T>
23acba struct point3D
f95b70 {
9cd6e5     typedef point3D P;
d0efa8     typedef const P &R;
329f43     T x, y, z;
d41d8c
4772a6     explicit point3D(T x = 0, T y = 0, T z = 0) : x(x), y(y), z(z) {}
c832e7     bool operator<(R p) const
f95b70     {
4485b9         return tie(x, y, z) < tie(p.x, p.y, p.z);
cbb184     }
8f15b3     bool operator==(R p) const
f95b70     {
469446         return tie(x, y, z) == tie(p.x, p.y, p.z);
cbb184     }
9ae9df     P operator+(R p) const { return P(x + p.x, y + p.y, z + p.z); }
54a3fe     P operator-(R p) const { return P(x - p.x, y - p.y, z - p.z); }
743d88     P operator*(T d) const { return P(x * d, y * d, z * d); }
17b561     P operator/(T d) const { return P(x / d, y / d, z / d); }
e4910c     T dot(R p) const { return x * p.x + y * p.y + z * p.z; }
8d13b8     P cross(R p) const
f95b70     {
923b71         return P(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y *
p.x);
cbb184     }
b70a0d     T dist2() const { return x * x + y * y + z * z; }
18b7a8     double dist() const { return sqrt((double)dist2()); }
d41d8c     //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
3d643b     double phi() const { return atan2(y, x); }
d41d8c     //Zenith angle (latitude) to the z-axis in interval [0, pi]
0fa109     double theta() const { return atan2(sqrt(x * x + y * y), z); }
55ead7     P unit() const { return *this / (T)dist(); } //makes dist()=1
d41d8c     //returns unit vector normal to *this and p
685a90     P normal(P p) const { return cross(p).unit(); }
d41d8c     //returns point rotated 'angle' radians ccw around axis
37a003     P rotate(double angle, P axis) const
f95b70     {
1b5479         double s = sin(angle), c = cos(angle);
9892bc         P u = axis.unit();
6b77df         return u * dot(u) * (1 - c) + (*this) * c - cross(u) * s;

```

```
cbb184      }
2145c1  };
d41d8c
d41d8c  // Returns the shortest distance on the sphere with radius "radius"
d41d8c  // between the points with azimuthal angles (longitude) f1 and f2
d41d8c  // from x axis and zenith angles (latitude) t1 and t2
d41d8c  // from z axis. All angles measured in radians.
d41d8c  // The algorithm starts by converting the spherical coordinates
d41d8c  // to cartesian coordinates so if that is what you have you can
d41d8c  // use only the two last rows. dx*radius is then the difference
d41d8c  // between the two points in the x direction and d*radius is the
d41d8c  // total distance between the points.
56c998  double spherical_distance(double f1, double t1,
0fa376                                double f2, double t2, double radius)
f95b70  {
8254bf      double dx = cos(t2) * cos(f2) - cos(t1) * cos(f1);
85235a      double dy = cos(t2) * sin(f2) - cos(t1) * sin(f1);
e9545f      double dz = sin(t2) - sin(t1);
c093df      double d = sqrt(dx * dx + dy * dy + dz * dz);
1549ea      return radius * 2 * asin(d / 2);
Full file hash: 1fa9d8
```


3.3 Convex Polygon and Circle Intersection

```

d41d8c // https://codeforces.com/gym/101158/
d41d8c
2729c3 #include "../misc/ternary_search/ternary_search_continuous.cpp"
ad578e #include "../2d/2d.cpp"
d41d8c
e89126 #define point point<double>
76d647 #define line line<double>
0f3aa0 #define circle circle<double>
1ba0cb #define segment segment<double>
d41d8c
d41d8c // Returns the intersection area between a convex polygon and a
d41d8c // circle.
d41d8c // Only works if circle center is inside the polygon and
d41d8c // the points in p are given in counter-clockwise order.
d41d8c // Has some precision issues, so EPS value is very relevant.
ceabb9 double circle_convex_polygon_intersection(const vector<point> &p,
    circle c)
f95b70 {
989fa2     double retv = 0;
90f3f1     int n = sz(p);
83008c     for (int i = 0; i < n; i++)
f95b70     {
244579         line l(p[i], p[(i + 1) % n]);
7ae2e0         segment s(p[i], p[(i + 1) % n]);
cba687         pair<point,point> res;
d41d8c
b56540         vector<point> bd; // Boundary points (either in segment or
d41d8c // segment-circle intersection).
874a5b         bd.push_back(p[i]);
acfd94         bd.push_back(p[(i + 1) % n]);
d41d8c
1155a7         if (c.intersect(l, res) == 2)
f95b70         {
0a2dd9             if (s.dist(res.first) < EPS)
c5c7d1                 bd.push_back(res.first);
ad5b75             if (s.dist(res.second) < EPS)
98cab5                 bd.push_back(res.second);
cbb184         }
d41d8c
b28a6e         sort(bd.begin() + 1, bd.end(), [&bd] (point lhs, point rhs) {
d41d8c             return (lhs-bd[0]).dist2() < (rhs-bd[0]).dist2(); });
f0c5c7
f95b70         if (bd.size() == 2)
fca073         {
            if ((bd[0] - c.center).dist() < c.r + EPS && (bd[1] - c.center)
                .dist() < c.r + EPS) // Segment completely inside.
7870b2                 retv += c.center.cross(bd[0], bd[1]) / 2;
2954e9                 else // Segment completely outside.

```

```

8242d2         retv += c.r * c.r * (bd[0] - c.center).angle(bd[1] - c.center
    ) / 2;
cbb184     }
b92fba     else if (bd.size() == 3) // One point inside circle and one
d41d8c         // outside.
f95b70     {
c6011c         if ((bd[0] - c.center).dist() < c.r + EPS)
f95b70         {
d41d8c             // Point 0 is inside.
7870b2             retv += c.center.cross(bd[0], bd[1]) / 2;
874bd3             retv += c.r * c.r * (bd[1] - c.center).angle(bd[2] - c.center
    ) / 2;
cbb184         }
2954e9     else
f95b70     {
d41d8c         // Point 2 is inside
2c3166         retv += c.center.cross(bd[1], bd[2]) / 2;
8242d2         retv += c.r * c.r * (bd[0] - c.center).angle(bd[1] - c.center
    ) / 2;
cbb184     }
cbb184     }
2954e9     else
f95b70     {
8242d2         retv += c.r * c.r * (bd[0] - c.center).angle(bd[1] - c.center)
    / 2;
2c3166         retv += c.center.cross(bd[1], bd[2]) / 2;
85ee55         retv += c.r * c.r * (bd[2] - c.center).angle(bd[3] - c.center)
    / 2;
cbb184     }
cbb184     }
d41d8c     return retv;
cbb184 }
d41d8c
d41d8c // This finds the maximum intersection between convex polygon and
d41d8c // any circle of a given radius.
d41d8c // Has some precision issues, review before using.
b1f14a double max_circle_intersection(const vector<point> &p, double r)
f95b70 {
1841a9     circle retv;
0afcf5     retv.r = r;
d41d8c
0a37af     auto f1 = [&](double x) {
e86a01         auto f2 = [&](double y) {
065f73             return -circle_convex_polygon_intersection(p, {point(x, y), r})
    ;
2145c1         };
d41d8c
a5b13e         double bot = 1e18;
781615         double top = -1e18;

```

```

fe16ed     for (int i = 0; i < sz(p); i++)
f95b70     {
e2299e         segment s1(p[i], p[(i + 1) % sz(p)]);
36cff7         segment s2(point(x, -1e3), point(x, 1e3));
d41d8c
6c4c4e         auto inter = s2.intersect(s1);
6327ce         if (inter.size() > 0)
f95b70         {
4f47ca             for (point a : inter)
f95b70             {
cbacd5                 bot = min(bot, a.y);
3b0f8a                 top = max(top, a.y);
cbb184             }
cbb184         }
cbb184     }
d41d8c
04379b         retv.center.y = ternary_search(f2, bot, top, EPS);
50ac50         return f2(retv.center.y);
2145c1     };
d41d8c
fdd13e     double botx = 1e18;
f5849a     double topx = -1e18;
fe16ed     for (int i = 0; i < sz(p); i++)
f95b70     {
f3840a         botx = min(botx, p[i].x);
de514a         topx = max(topx, p[i].x);
cbb184     }
d41d8c
b055bb     retv.center.x = ternary_search(f1, botx, topx, EPS);
d41d8c
fdb3d6     return -f1(retv.center.x);
cbb184 }
d41d8c
13a4b1 int main(void)
f95b70 {
1a88fd     int n;
c1224c     double r;
a68515     cin >> n >> r;
cfb0d9     vector<point> p(n);
83008c     for (int i = 0; i < n; i++)
243162         cin >> p[i].x >> p[i].y;
d41d8c
79dca9     printf("%.20lf\n", max_circle_intersection(p, r));
cbb184 }
Full file hash: 50f2e3

```

3.4 Closest Pair of points

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
4355f8     Closest Pair of points  $O(n * \log n)$ :
bd72a8     Finds the closest pair of points from a set of given points.
d41d8c
b95cae     Usage:
96d0df     Call closest_pair with the array of points and the number.
bf2691     The function will modify the array.
0794bd     Then, get the squared distance from ans and the indexes
1a530e     of both points from idx.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
69abfb #define MAXN 112345
d41d8c
4befb0 struct point
f95b70 {
0be4e5     ll x, y;
53e65f     int id;
2145c1 };
d41d8c
828140 namespace closest_pair
f95b70 {
fef62f     point tmp[MAXN];
d41d8c
f5be94     ll ans = infll;
b733a0     int idx[2];
d41d8c
8ce1f9     void update(point i, point j)
f95b70     {
fd08d3         ll dist = (i.x - j.x) * (i.x - j.x) + (i.y - j.y) * (i.y - j.y);
5734c7         if (dist < ans)
f95b70         {
2341bb             ans = dist;
7f62e1             idx[0] = min(i.id, j.id);
a4f5cc             idx[1] = max(i.id, j.id);
cbb184         }
cbb184     }
d41d8c
5810d3     bool compx(point a, point b) { return a.x < b.x; }
d41d8c
71b8c2     bool compy(point a, point b) { return a.y < b.y; }
d41d8c
98ebec     void solve(point p[], int l, int r)
f95b70     {
c00ab0         if (r - l <= 3)

```

```

f95b70     {
245656     for (int i = l; i <= r; i++)
039087         for (int j = i + 1; j <= r; j++)
2af8f7             update(p[i], p[j]);
aa321e     sort(p + l, p + r + 1, compy);
505b97     return;
cbb184     }
d41d8c
ae007b     int mid = (l + r) / 2;
709b2b     ll xmid = p[mid].x;
d41d8c
b28835     solve(p, l, mid), solve(p, mid + 1, r);
6c2a65     merge(p + l, p + mid + 1, p + mid + 1, p + r + 1, tmp, compy);
0dc2a6     copy(tmp, tmp + r - l + 1, p + l);
d41d8c
1fcfd8     int sz = 0;
245656     for (int i = l; i <= r; i++)
7a807a         if ((p[i].x - xmid) * (p[i].x - xmid) < ans)
f95b70         {
ed1c37             for (int j = sz - 1; j >= 0 && (p[i].y - tmp[j].y) * (p[i].y -
                tmp[j].y) < ans; j--)
28f98a                 update(p[i], tmp[j]);
db9665                 tmp[sz++] = p[i];
cbb184         }
cbb184     }
d41d8c
d50be6     void closest_pair(point p[], int n)
f95b70     {
ac1f1a         ans = infl;
c55ecb         sort(p, p + n, compx);
f8fda2         solve(p, 0, n - 1);
cbb184     }
2145c1 }; // namespace closest_pair
d41d8c
Full file hash: 77bdaa

```

3.5 Convex Hull - Sweep Line

```

d41d8c
5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
d954ca     Convex hull:
db7203         Computes lower and upper convex hull for a set of points in
93e88e         O(n * log n).
91502f         Using lower and upper convex hull you can also check if a
ccea8         point belongs
6e26be         to the polygon in O(log n) with the point_in_ch function.
d41d8c
b95cae     Usage:
0fe9f0         Upper/lower hulls start at lowest x (tie broken by lowest y)
f33fe6         and end at highest x (tie broken by highest y).
f189fb         Points can be collinear, but convex hull will not contain
f9c0f7         collinear points.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
4befb0 struct point
f95b70 {
0be4e5     ll x, y;
d41d8c
e45b92     explicit point(ll x = 0, ll y = 0) : x(x), y(y) {}
d41d8c
76082e     ll cross(point p1, point p2) { return (p1.x - x) * (p2.y - y) - (p2
.x - x) * (p1.y - y); }
d41d8c
5bb1b0     bool operator<(const point &rhs) const { return tie(x, y) < tie(rhs
.x, rhs.y); }
2145c1 };
d41d8c
249af2 void convex_hull(vector<point> p, vector<point> &upper, vector<point>
&lower)
f95b70 {
905c3f     sort(p.begin(), p.end());
c796b3     auto build = [&p](vector<point> &ch, ll tp) {
2b34c4         ch.push_back(p[0]), ch.push_back(p[1]);
074b6d         for (int i = 2; i < sz(p); i++)
f95b70         {
d638b8             while (ch.size() >= 2 && tp * ch[sz(ch) - 2].cross(ch.back(), p
[i]) >= 0)
9d9654                 ch.pop_back();
3db638             ch.push_back(p[i]);
cbb184         }
2145c1     };
d41d8c

```

```

5ad0c4     build(upper, 1);
6be537     build(lower, -1);
cbb184 }
d41d8c
d41d8c // Optional.
d41d8c // Checks if point o is inside the convex hull area in O(log n).
d41d8c // Also returns true if point is on the convex hull perimeter.
c90a0f bool point_in_ch(point o, vector<point> &upper, vector<point> &lower)
f95b70 {
329d8c     if (o.x < upper[0].x || o.x > upper.back().x)
d1fe4d         return false;
d41d8c
e22e3a     auto check = [o](vector<point> &ch, ll tp) {
f37175         int i = lower_bound(ch.begin(), ch.end(), o, [](point a, point b)
        { return a.x < b.x; }) - ch.begin();
e660eb         if ((i != 0 && tp * ch[i - 1].cross(ch[i], o) > 0) ||
2d9038             (i + 1 < sz(ch) && tp * ch[i].cross(ch[i + 1], o) > 0) ||
f11c44             (i + 2 < sz(ch) && tp * ch[i + 1].cross(ch[i + 2], o) > 0))
d1fe4d             return false;
d41d8c
8a6c14         return true;
2145c1     };
d41d8c
65476a     return check(upper, 1) && check(lower, -1);
cbb184 }
Full file hash: b7da8b

```

3.6 Convex Hull – Graham Scan

```

ad578e #include "../2d/2d.cpp"
d41d8c
d41d8c /*
a556d8     Solution for convex hull problem (minimum polygon covering a set
6bccb0     of points) based on ordering points by angle.
3248ac     * Finds the subset of points in the convex hull in O(Nlog(N)).
5686c3     * This version works if you either want intermediary points in
900618     segments or not (see comments delimited by //)
01b744     * This version works when all points are collinear
65c116     * This version works for repeated points if you add a label to
327f1a     struct, and use this label in overloaded +, - and =.
c4c9bd */
d41d8c
d41d8c //Only uses 'struct point' form 2d.cpp. Apply following changes to
    use
d41d8c //with double:
d41d8c //Double version: bool operator<(P p) const { return fabs(x - p.x) <
d41d8c //                      EPS ? y < p.y :
d41d8c //                      x < p.x; }
d41d8c //Double version: bool operator==(P p) const { return fabs(x - p.x) <
d41d8c //                      EPS &&
d41d8c //                      fabs(y - p.y) < EPS; }
d41d8c
d41d8c /* Compara primeiro por angulo em relacao a origem e depois por
8108ff     distancia para a origem */
67a100 template <typename T>
f309c0 bool cmp(point<T> a, point<T> b)
f95b70 {
a9b570     if (a.cross(b) != 0)
c33606         return a.cross(b) > 0;
ba7b3a     return a.dist2() < b.dist2();
cbb184 }
d41d8c
67a100 template <typename T>
2fc305 vector<point<T>> CH(vector<point<T>> points)
f95b70 {
d41d8c     /* Encontra pivo (ponto extremos que com ctz faz parte do CH) */
95b799     point<T> pivot = points[0];
e409fb     for (auto p : points)
e01c07         pivot = min(pivot, p);
d41d8c
d41d8c     /* Desloca conjunto para pivo ficar na origem e ordena pontos pelo
e87d90     angulo e distancia do pivo */
9ac126     for (int i = 0; i < (int)points.size(); i++)
3010bd         points[i] = points[i] - pivot;
d41d8c
e2c4e0     sort(points.begin(), points.end(), cmp<ll>);
d41d8c

```



```
9ac126     for (int i = 0; i < (int)points.size(); i++)
eda5a9         points[i] = points[i] + pivot;
d41d8c
d41d8c     /* Ponto extra para fechar o poligono */
36b3da     points.push_back(points[0]);
d41d8c
620533     vector<point<T>> ch;
d41d8c
e409fb     for (auto p : points)
f95b70     {
d41d8c         /* Enquanto o proximo ponto gera uma curva para a direita,
e6d815         retira ultimo ponto atual */
d41d8c         /* Segunda comparaçãõ serve para caso especial de pontos
15307c         colineares quando se quer eliminar os intermediarios */
d41d8c         // Trocar terceira comparacao pra <= para descartar pontos do
d41d8c         // meio de arestas no ch
d41d8c         // Double: trocar terceira comparaçãõ por < EPS (descarta
d41d8c         // pontos em arestas) ou < -EPS (mantem pinto em aresta
29fcb4         while (ch.size() > 1 && !(p == ch[ch.size() - 2]) && ch[ch.size()
- 2].cross(ch[ch.size() - 1], p) < 0)
9d9654             ch.pop_back();
d2ebaf         ch.push_back(p);
cbb184     }
d41d8c
d41d8c     /*Elimina ponto extra*/
9d9654     ch.pop_back();
d41d8c
66cc3c     return ch;
cbb184 }
Full file hash: 19c056
```

3.7 Min Enclosing Circle (randomized)

```

ad578e  #include "../2d/2d.cpp"
d41d8c
d41d8c  /*
744027    Minimum Enclosing Circle:
00485e        Given a list of points, returns a circle of minimum radius
e9a6ea        such that all given points are within the circle.
eea06e        Runs in O(n) expected time (in practice 200 ms for 10^5
68234f        points).
d41d8c
ca2095    Constraints:
99b71a        Non-empty list of points.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
e89126  #define point point<double>
0f3aa0  #define circle circle<double>
d41d8c
41ee07  circle min_enclosing_circle(vector<point> p)
f95b70  {
b4da45      shuffle(p.begin(), p.end(), mt19937(time(0)));
2e09de      point o = p[0];
76160f      double r = 0, eps = 1 + 1e-8;
fe16ed      for (int i = 0; i < sz(p); i++)
197ee7          if ((o - p[i]).dist() > r * eps)
f95b70          {
ba37a5              o = p[i], r = 0;
c791cd              for (int j = 0; j < i; j++)
f5972f                  if ((o - p[j]).dist() > r * eps)
f95b70                  {
d2b545                      o = (p[i] + p[j]) / 2;
0657ce                      r = (o - p[i]).dist();
674051                      for (int k = 0; k < j; k++)
355d4d                          if ((o - p[k]).dist() > r * eps)
f95b70                          {
7fb807                              o = circumcircle(p[i], p[j], p[k]).center;
0657ce                              r = (o - p[i]).dist();
cbb184                          }
cbb184                  }
cbb184          }
d41d8c
645c1d      return {o, r};
cbb184  }
d41d8c
Full file hash: 5d3836

```

3.8 Min Enclosing Circle (ternary search)

```

ad578e  #include "../2d/2d.cpp"
2729c3  #include "../misc/ternary_search/ternary_search_continuous.cpp"
d41d8c
d41d8c  /*
744027    Minimum Enclosing Circle:
00485e        Given a list of points, returns a circle of minimum radius
e9a6ea        such that all given points are within the circle.
a29bff        Runs in  $O(n * \log^2((top - bot) / eps))$  (in practice 2.5s at
652727        best for  $10^5$  points).
d41d8c
ca2095    Constraints:
99b71a        Non-empty list of points.
d41d8c
b95cae    Usage:
63c3f2        The coordinates of the circle's center must be in the range
85bc5a        [bot, top].
bf45ba        eps specifies the precision of the result, but set it to a
9e11c6        higher value than necessary since the error in x affects the
2b116c        y value.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
e89126  #define point point<double>
0f3aa0  #define circle circle<double>
d41d8c
e1710f  circle min_enclosing_circle(const vector<point> &p, double bot = -1e9
, double top = 1e9, double eps = 1e-9)
f95b70  {
1841a9    circle retv;
d41d8c
0a37af    auto f1 = [&](double x) {
d9991d        auto f2 = [&](double y)
f95b70        {
996834            double r = 0;
fe16ed            for (int i = 0; i < sz(p); i++)
62adf4                r = max(r, (p[i].x - x)*(p[i].x - x) + (p[i].y - y)*(p[i].y -
y));
4c1f3c            return r;
2145c1        };
410f57        retv.center.y = ternary_search(f2, bot, top, eps);
50ac50        return f2(retv.center.y);
2145c1    };
d41d8c
596ad7    retv.center.x = ternary_search(f1, bot, top, eps);
3b2a60    retv.r = sqrt(f1(retv.center.x));
d41d8c
6272cf    return retv;

```

cbb184 }

Full file hash: 2acede

3.9 Rotating Calipers - Antipodal

```

b79ded #include "../graham_scan_convex_hull/graham_scan.cpp"
d41d8c
d41d8c /*
63c392     Antipodal pairs O(n):
159d83     Uses rotating calipers techniqe to find all antipodal pairs.
1c0e76     Returned list will be such the the entire polygon lies between
d26ffe     the line defined by (p[retv[i].first],
c138b3         p[(retv[i].first + 1) % n])
c53770     and a parallel line passing by p[retv[i].second].
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
d41d8c // p is a convex hull in ccw order with no duplicate or collinear
d41d8c // points. Might not work as expected for two points.
9bbc91 vector<pii> antipodal_pairs(const vector<point<ll>> &p)
f95b70 {
15925a     int j = 1, n = sz(p);
070406     vector<pii> retv;
83008c     for (int i = 0; i < n; i++)
f95b70     {
d41d8c         // While j + 1 is farther from segment {i, i+1} than j.
c68fef         while (p[i].cross(p[(i + 1) % n], p[(j + 1) % n]) > p[i].cross(p[
            [(i + 1) % n], p[j]))
600403             j = (j + 1) % n;
d41d8c
d89902         retv.push_back({i, j});
d41d8c
d41d8c         // If j + 1 is at the same distance as j, both pairs are
d41d8c         // antipodal.
24f7fb         if (p[i].cross(p[(i + 1) % n], p[(j + 1) % n]) == p[i].cross(p[(i
            + 1) % n], p[j]))
9b972b             retv.push_back({i, (j + 1) % n});
cbb184     }
d41d8c
6272cf     return retv;
cbb184 }
Full file hash: 8bebf6

```

3.10 Rotating Calipers - Convex Polygon Bouding Box

```

b79ded #include "../graham_scan_convex_hull/graham_scan.cpp"
d41d8c
d41d8c /*
685ef6     Bounding Box O(n):
0d0223         Finds the smallest perimeter for a rotated rectangle
ce1da6         that covers the entire given convex polygon.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
d41d8c // p is a convex hull in ccw order with no duplicate or
d41d8c // collinear points. Might not work as expected for two points.
6b75d8 double min_bounding_box_perimeter(const vector<point<ll>> &p)
f95b70 {
15925a     int j = 1, n = sz(p);
3bf848     int k = 1, l = 1;
d41d8c
49dac9     double ans = 1e18;
83008c     for (int i = 0; i < n; i++)
f95b70     {
d41d8c         // While j + 1 is farther from segment {i, i+1} than j.
c68fef         while (p[i].cross(p[(i + 1) % n], p[(j + 1) % n]) > p[i].cross(p
[(i + 1) % n], p[j]))
600403             j = (j + 1) % n;
d41d8c
1473fe         if (i == 0)
e37f78             l = j;
d41d8c
b6dda7         while ((p[(i + 1) % n] - p[i]).dot(p[(k + 1) % n] - p[k]) > 0)
399fa2             k = (k + 1) % n;
d41d8c
88148d         while ((p[(i + 1) % n] - p[i]).dot(p[(l + 1) % n] - p[l]) < 0)
f24778             l = (l + 1) % n;
d41d8c
6c5e35         line<ll> ln(p[i], p[(i + 1) % n]);
94ca0c         ans = min(ans, 2 * ln.dist(p[j]) +
71655f             2 * (p[(i + 1) % n] - p[i]).proj(p[k] - p[i]).dist() +
ad9971             2 * (p[(i + 1) % n] - p[i]).proj(p[l] - p[i]).dist());
cbb184     }
d41d8c
ba75d2     return ans;
cbb184 }
Full file hash: 3eb318

```

3.11 Rotating Calipers - Convex Polygon Diameter

```

21e975 #include "antipodal_pairs.cpp"
d41d8c
d41d8c /*
31acb4     Polygon Diameter O(n):
c98af8         Gets the largest distance for a pair of points
f25925         in a convex polygon.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
d41d8c // p is a convex hull in ccw order with no duplicate or
d41d8c // collinear points.
cf5142 double convex_polygon_diameter(const vector<point<ll>> &p)
f95b70 {
fc82f0     vector<pii> anti = antipodal_pairs(p);
d41d8c
989fa2     double retv = 0;
623203     for (pii a : anti)
f95b70     {
285483         if ((p[a.first] - p[a.second]).dist() > retv)
46faad             retv = (p[a.first] - p[a.second]).dist();
d41d8c
6bdb59         if ((p[(a.first + 1) % sz(p)] - p[a.second]).dist() > retv)
5f5564             retv = (p[(a.first + 1) % sz(p)] - p[a.second]).dist();
cbb184     }
d41d8c
6272cf     return retv;
cbb184 }
Full file hash: fd37ea

```

3.12 Rotating Calipers - Convex Polygon Width

```

21e975 #include "antipodal_pairs.cpp"
d41d8c
d41d8c /*
bd42f2     Polygon Width O(n):
effffb         Gets the smallest width for a "tunnel" by which you can
b9a211         slide the convex polygon.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
d41d8c // p is a convex hull in ccw order with no duplicate or
d41d8c // collinear points.
c40976 double convex_polygon_width(const vector<point<ll>> &p)
f95b70 {
fc82f0     vector<pii> anti = antipodal_pairs(p);
d41d8c
80ba22     double retv = 1e18;
fe49b8     for (int i = 0; i < sz(anti); i++)
f95b70     {
7941e6         line<ll> l(p[anti[i].first], p[(anti[i].first + 1) % sz(p)]);
63e5f2         if (l.dist(p[anti[i].second]) < retv)
b79fff             retv = l.dist(p[anti[i].second]);
cbb184     }
d41d8c
6272cf     return retv;
cbb184 }
Full file hash: 353b1b

```


4 Graph

4.1 2-Sat

```

d41d8c
5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
e433a5     2-SAT O(N + E):
55f7c7         Calculates a valid assignment to boolean variables a, b, c,...
60b5f0         to a 2-SAT problem, so that an expression of the type (a || b)
4ec317         && (!a || c) && (d || !b) && ...
be11b4         becomes true, or reports that it is unsatisfiable.
d41d8c
ca2095     Constraints:
340a2c         Variables are labeled form 0 to n-1.
d41d8c
b95cae     Usage:
ba3cd0         Negated variables are represented by bit-inversions (~x).
d41d8c
0642c0     Usage sample:
1f57f5         two_sat ts(number of boolean variables);
2298ae         ts.either(0, ~3); // Var 0 is true or var 3 is false
25ac45         ts.set_true(2); // Var 2 is true
3bc16d         ts.set_true(~0); // Var 0 is false
433a2b         ts.at_most_one({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
7428e8         ts.solve(); // Returns true iff it is solvable
031b2f         ts.values[0..N-1] holds the assigned values to the vars
d41d8c
1d1558     Source: https://github.com/kth-competitive-programming/kactl/blob/
4ca6c0         master/content/graph/2sat.h
c4c9bd */
d41d8c
48c8d6 struct two_sat
f95b70 {
1a88fd     int n;
3098d4     vector<vector<int>> graph;
21fa4d     vector<int> values; // 0 = false, 1 = true
d41d8c
16cbf5     two_sat(int n = 0) : n(n), graph(2 * n) {}
d41d8c
d41d8c     // a || b.
c34a10 void either(int a, int b)
f95b70 {
80f104     a = max(2 * a, -1 - 2 * a);
b0d997     b = max(2 * b, -1 - 2 * b);
c5f3f1     graph[a].push_back(b ^ 1);
87da37     graph[b].push_back(a ^ 1);
cbb184 }
d41d8c

```

```

d41d8c // x == true.
ac44e0 void set_true(int x) { either(x, x); } // (optional)
d41d8c
6bdbf3 int add_var() // (optional)
f95b70 {
b3bd55     graph.emplace_back();
b3bd55     graph.emplace_back();
695b27     return n++;
cbb184 }
d41d8c
d41d8c // Zero or one of variables in the list must be true.
d41d8c // This will create auxiliary variables.
485ee6 void at_most_one(const vector<int> &li) // (optional)
f95b70 {
3e592b     if (sz(li) <= 1)
505b97         return;
da9e57     int cur = ~li[0];
0b7bba     for (int i = 2; i < sz(li); i++)
f95b70     {
78687b         int next = add_var();
909c7f         either(cur, ~li[i]);
86e0e2         either(cur, next);
d1aaa0         either(~li[i], next);
072102         cur = ~next;
cbb184     }
d41d8c
ed7d2a     either(cur, ~li[1]);
cbb184 }
d41d8c
71b50d vector<int> val, comp, z;
da47f9 int time = 0;
d41d8c
9a6bd8 int dfs(int i)
f95b70 {
9dee5e     int low = (val[i] = ++time), x;
c5b648     z.push_back(i);
a17795     for (int e : graph[i])
7c7a5e         if (!comp[e])
0e86c7             low = min(low, val[e] ? val[e] : dfs(e));
28412d     if (low == val[i])
f95b70     {
d4579b         do
f95b70         {
792d73             x = z.back();
a04857             z.pop_back();
7ccb4             comp[x] = low;
14283e             if (values[x >> 1] == -1)
3784ec                 values[x >> 1] = x & 1;
fb5137         } while (x != i);
cbb184     }

```

```
d41d8c
3e1f0f     return val[i] = low;
cbb184    }
d41d8c
d41d8c     // Returns true if solution exists and values[0..n-1] holds the
d41d8c     // assigned values to the vars.
fcde20    bool solve()
f95b70    {
bf1691        values.assign(n, -1);
c5b951        val.assign(2 * n, 0);
e200b2        comp = val;
3df60a        for (int i = 0; i < 2 * n; i++)
f894a1            if (!comp[i])
1e5da3                dfs(i);
83008c        for (int i = 0; i < n; i++)
17e04c            if (comp[2 * i] == comp[2 * i + 1])
d1fe4d                return false;
8a6c14        return true;
cbb184    }
2145c1 };
Full file hash: fff52a
```

4.2 Biconnected Components

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
df7805 Finding bridges, articulation points and biconnected components in
5bdbd1 O(V + E):
9cae86 A bridge is an edge whose removal splits the graph in two
81eac7 connected components.
b99b5d An articulation point is a vertex whose removal splits the
26af5b graph in two connected components.
d41d8c
db8f61 A biconnected component (or 2VCC) is a maximal subgraph where
081109 the removal of any vertex doesn't
4ee263 make the subgraph disconnected. In other words, it is a
ca6018 maximal 2-vertex-connected (2VC) subgraph.
d41d8c
d9b8f4 A 2-connected graph is a 2VC one, except that a---b is
3c1ace considered 2VC but not 2-connected.
d41d8c
3a585b Useful theorems:
d41d8c
f3f658 A 2-edge connected (2EC) graph is a graph without bridges.
7bdcc8 Any 2-connected graph is also 2EC.
d41d8c
6550a6 Let G be a graph on at least 2 vertices. The following
85875d propositions are equivalent:
81e567 * (i) G is 2-connected;
57674c * (ii) any two vertices are in a cycle; (a cycle can't
8044ed repeat vertices)
f8a586 * (iii) any two edges are in a cycle and degree(G) >= 2;
654422 * (iv) for any three vertices x,y et z, there is a
b4be10 (x,z)-path containing y.
81e274 Let G be a graph on at least 3 vertices. The following
85875d propositions are equivalent:
346e5e * (i) G is 2-edge-connected;
4883af * (ii) any edge is in a cycle;
523eda * (iii) any two edges are in a tour and degree >= 1;
96edf8 * (iv) any two vertices are in a tour
cf2f8a (a tour can repeat vertices)
d41d8c
c57c4a If G is 2-connected and not bipartite, all vertices belong to
b96bcc some odd cycle. And any two vertices are in a odd cycle (not
e7ef1d really proven).
d41d8c
bcc5c9 If G is 2-edge-connected (proof by AC):
1a2e35 For any two vertices x, y and one edge e, there is a
509be9 (x, y)-walk containing e without repeating edges.
d41d8c
727afd A graph admits a strongly connected orientation if and only

```

```

56e3d3         if it is 2EC.
002aec         A strong orientation of a given bridgeless undirected graph
09302d         may be found in linear time by performing a depth first search
fd5c96         of the graph, orienting all edges in the depth first search
29b020         tree away from the tree root, and orienting all the remaining
21549f         edges (which must necessarily connect an ancestor and a
833be1         descendant in the depth first search tree) from the
edf1c0         descendant to the ancestor.
d41d8c
ca2095     Constraints:
b9aa54         ***undirected*** graph.
80b2d0         Vertices are labeled from 0 to n (inclusive).
1e2ce5         Graph is connected (but for unconnected just replace single
cdc50c         dfs call with a loop).
d41d8c
b95cae     Usage:
5b7348         Create the struct setting the starting vertex (a), the maximum
ed5ffc         vertex label (n),
5256c4         the graph adjacency list (graph) and a callback f to apply on
6f4b97         the biconnected components.
f8f25e         Afterwards, art[i] == true if i is an articulation point.
71cfdc         If the pair {a, i} is on the bridges list, then the edge
2dbf3a         {a, graph[a][i]} is a bridge.
7e9183         The callback must receive a vector of edges {a, b} that are
6f03eb         in the same biconnected component.
bbec2f         Remember that for a single vertex, the biconnected callback
e223cd         will not be called.
d41d8c
e152b4     Sample Usage:
0ec6ee         auto rdm = apb(1, n, graph, [&](vector<pii> v){
f4ecd5         set<int> s;
9ad08e         for (int i = 0; i < sz(v); i++)
f95b70         {
f19ef4             s.insert(v[i].first);
0858fa             s.insert(v[i].second);
cbb184         }
d41d8c
0fe299         ans = max(ans, sz(s));
c0c97e         });
c4c9bd     */
d41d8c
f117a6     struct apb
f95b70     {
9cf2b9         vector<int> *graph;
9cf143         vector<bool> art;
c9001f         vector<int> num /* dfs order of vertices starting at 1 */, low;
c83796         vector<pii> bridges;
91936b         vector<pii> st;
53e65f         int id;
d41d8c

```

```

044d82  template<class F>
09caad  apb(int a, int n, vector<int> graph[], const F &f) : graph(graph),
      art(n + 1, false), num(n + 1), low(n + 1)
f95b70  {
0f6720      id = 1;
ccac4e      dfs(a, a, f);
cbb184  }
d41d8c
044d82  template<class F>
dc584b  void dfs(int a, int p, const F &f)
f95b70  {
7be506      low[a] = num[a] = id++;
34863b      int comp = 0;
d41d8c
1429ef      for (int i = 0; i < sz(graph[a]); i++)
f95b70      {
b7a810          if (num[graph[a][i]] == 0)
f95b70          {
d40410              int si = sz(st);
f309f5              comp++;
8ece2e              st.push_back({a, graph[a][i]}); // Tree edge.
d41d8c
fc5941              dfs(graph[a][i], a, f);
085d64              low[a] = min(low[a], low[graph[a][i]]);
d41d8c
bb63a0              if (low[graph[a][i]] >= num[a])
f95b70              {
558f81                  if (a != 1)
016392                      art[a] = true;
d41d8c
b91456                      f(vector<pii>(st.begin() + si, st.end()));
901921                      st.resize(si);
cbb184              }
d41d8c
0e9ddb              if (low[graph[a][i]] > num[a])
b3cacb                  bridges.push_back({a, i});
cbb184          }
624580      else if (graph[a][i] != p && num[graph[a][i]] < num[a])
f95b70          {
d41d8c              // Back edge.
066898                  low[a] = min(low[a], num[graph[a][i]]);
8ece2e                  st.push_back({a, graph[a][i]});
cbb184              }
cbb184      }
d41d8c
85e3a2      if (a == p && comp > 1)
016392          art[a] = true;
cbb184      }
2145c1  };
d41d8c

```

Full file hash: 5cb0b8

4.3 Bipartite Matching (Hopcroft Karp)

```

2b74fa #include <bits/stdc++.h>
ca417d using namespace std;
d41d8c
d41d8c /*
ec23c9     Hopcroft-Karp:
eaeddf         Bipartite Matching O(sqrt(V)E)
d41d8c
ca2095     Constraints:
998cc9         Vertices are labeled from 1 to l + r (inclusive).
682ff0         DO NOT use vertex 0.
968b86         Vertices 1 to l belong to left partition.
a6a4c4         Vertices l + 1 to l + r belong to right partition.
d41d8c
b95cae     Usage:
d86132         Set MAXV if necessary.
70636b         Call init passing l and r.
0f3b71         Add edges to the graph from left side to right side.
5263f1         Call hopcroft to get the matching size.
661627         Then, each vertex v has its pair indicated in p[v] (or 0
259ac0         for not paired).
c4c9bd */
d41d8c
d41d8c namespace hopcroft
d41d8c {
f95b70
998014     const int inf = 0x3f3f3f3f;
ed5ed2     const int MAXV = 112345;
d41d8c
3098d4     vector<vector<int>> graph;
0a3d29     int d[MAXV], q[MAXV], p[MAXV], l, r;
d41d8c
4025e1     void init(int _l, int _r)
f95b70     {
0ebd66         l = _l, r = _r;
2213c3         graph = vector<vector<int>>(l + r + 1);
cbb184     }
d41d8c
6a1cf9     bool bfs()
f95b70     {
18753f         int qb = 0, qe = 0;
4f2bde         memset(d, 0x3f, sizeof(int) * (l + 1));
a89ba9         for (int i = 1; i <= l; i++)
8b3877             if (p[i] == 0)
248d2f                 d[i] = 0, q[qe++] = i;
d41d8c
2caa87         while (qb < qe)
f95b70         {
e8e8a0             int a = q[qb++];
0087d7             if (a == 0)

```



```

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

```

4.4 Bridges/Articulation Points

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
62784d     Finding bridges and articulation points in  $O(V + E)$ :
9cae86     A bridge is an edge whose removal splits the graph in two
81eac7     connected components.
b99b5d     An articulation point is a vertex whose removal splits the
26af5b     graph in two connected components.
d24e48     This can also be adapted to generate the biconnected
480550     components of a graph, since the articulation points split
ebc6fe     components.
d41d8c
d41d8c     Constraints:
ca2095     ***undirected*** graph.
b9aa54     Vertices are labeled from 0 to n (inclusive).
80b2d0     Graph is connected (otherwise it doesn't make sense).
d41d8c
b95cae     Usage:
5b7348     Create the struct setting the starting vertex (a), the maximum
1f1f42     vertex label (n) and the graph adjacency list (graph).
6ffc91     Afterwards, art[i] == true if i is an articulation point.
71cfdc     If the pair {a, i} is on the bridges list, then the edge
2dbf3a     {a, graph[a][i]} is a bridge.
c4c9bd */
d41d8c
f117a6 struct apb
f95b70 {
9cf2b9     vector<int> *graph;
9cf143     vector<bool> art;
c9001f     vector<int> num /* dfs order of vertices starting at 1 */, low;
c83796     vector<pii> bridges;
53e65f     int id;
d41d8c
4dc736     apb(int a, int n, vector<int> graph[]) : graph(graph), art(n + 1,
false), num(n + 1), low(n + 1)
f95b70     {
0f6720         id = 1;
bb407e         dfs(a, a);
cbb184     }
d41d8c
69c421     void dfs(int a, int p)
f95b70     {
7be506         low[a] = num[a] = id++;
34863b         int comp = 0;
d41d8c
c4fff3         for (int i = 0; i < graph[a].size(); i++)
f95b70         {

```

```
b7a810     if (num[graph[a][i]] == 0)
f95b70     {
f309f5         comp++;
783129         dfs(graph[a][i], a);
085d64         low[a] = min(low[a], low[graph[a][i]]);
d41d8c
b28b5f         if (a != 1 && low[graph[a][i]] >= num[a])
016392             art[a] = true;
d41d8c
0e9ddb         if (low[graph[a][i]] > num[a])
b3cacb             bridges.push_back({a, i});
cbb184     }
2cae7a     else if (graph[a][i] != p && num[graph[a][i]] < low[a])
ed0d8a         low[a] = num[graph[a][i]];
cbb184     }
d41d8c
85e3a2     if (a == p && comp > 1)
016392         art[a] = true;
cbb184     }
2145c1 };
d41d8c
Full file hash: 780b6d
```

4.5 Centroid Decomposition

```

5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
05773c     Centroid Decomposition:
6fc36f         Solve tree problems by divide and conquer splitting the tree
2b598f         repeatedly on centroid.
df452b         Centroid is the vertex with smallest <largest subtree>.
6f6d7b         O(n log n) if process is O(sz)
d41d8c
b95cae     Usage:
54abda         Call put_edge to initialize the tree edges.
f442d7         Then call decomp(i, n) for any vertex i in the tree, with n
e34f6d         being the number of vertices.
9a4f69         Function process will be called for a centroid <a> with
b282dc         subtree total size sz.
8d0f04         In process you can use:
14d894             graph[a][i] - graph adjacency list
2d278e             block[a] - true if you should ignore the vertex.
454645             sub_size[a][i] - subtree size for edge a -> graph[a][i]
3c9c0e             (considering only non-blocked parts).
d41d8c
37596d         if process can be O(sz + h * log) where h is subtree height it
574f9e         is a lot better constant than O(sz * log)
d41d8c
2e1f62     PRINT APPLICATION WITH THIS.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
69abfb #define MAXN 112345
d41d8c
f00da0 void process(int a, int sz);
d41d8c
47ba6e vector<int> graph[MAXN];
ff6c5e vector<int> sub_size[MAXN];
3992d5 bool block[MAXN];
d41d8c
73833f int dfs_centroid(int a, int p, int sz, int &centroid, int &val)
f95b70 {
9dcc34     int sum = 0, mx = 0, pid = -1;
1429ef     for (int i = 0; i < sz(graph[a]); i++)
7d7af1         if (graph[a][i] != p && !block[graph[a][i]])
f95b70             {
143ef8                 int x = dfs_centroid(graph[a][i], a, sz, centroid, val);
d41d8c
8a1a8f                 sub_size[a][i] = x;
e79613                 mx = max(x, mx);
8e032b                 sum += x;

```

```

cbb184     }
c86ade     else if (graph[a][i] == p && !block[graph[a][i]])
348edf         pidx = i;
d41d8c
d639e6     if (pidx != -1)
f95b70     {
76b8e6         sub_size[a][pidx] = sz - sum - 1;
299abe         mx = max(mx, sub_size[a][pidx]);
cbb184     }
d41d8c
53f952     if (mx < val)
c1ab59         val = mx, centroid = a;
d41d8c
5e2eb0     return sum + 1;
cbb184 }
d41d8c
4e2a11 void decomp(int a, int sz)
f95b70 {
7fe7e7     int val = inf;
65821c     dfs_centroid(a, a, sz, a, val);
d41d8c
4d8555     process(a, sz);
d41d8c
8b3e5f     block[a] = true;
1429ef     for (int i = 0; i < sz(graph[a]); i++)
5f14fe         if (!block[graph[a][i]])
e0cc02             decomp(graph[a][i], sub_size[a][i]);
cbb184 }
d41d8c
93961e void put_edge(int a, int b)
f95b70 {
8fa0be     graph[a].push_back(b);
cb6128     sub_size[a].push_back(0);
4c6197     graph[b].push_back(a);
3a136d     sub_size[b].push_back(0);
cbb184 }

```

Full file hash: e63204

4.6 Euler Tour

```

d41d8c  /*
ca5da8  [DEFINITION]
468171    a) Eulerian Path: visits every edge only once, but can repeat
4c1025    vertices.
6f1929    b) Eulerian Cycle: is a eulerian path that is a cycle
85966e    (start vertex == end vertex)
86f723    OBS: We disconsider vertices that have indegree==outdegree==0
4f4b8b    (we call them as useless vertices)
d41d8c
fadacf  [CONDITIONS]
058d84    [Undirected graph]
129577    [Path/Cycle]
d18b39    a) The number of vertices with odd degrees is 2(Eulerian
b33be4    Path) or 0(Eulerian cycle)
9d36d5    b) The graph of useful vertices (see OBS above) should be
06aa6f    connected
438031    If either of the above condition fails Euler Path/Cycle
ec3bfc    can't exist.
1f9012    [Directed graph]
e181bd    [Cycle]
b7de35    a) All vertices should have (indegree==outdegree)
32725a    b) The UNDIRECTED version of the graph of useful
b03f8d    vertices (see OBS above) should be connected
cf65ac    [Path]
0238b6    a) Equal to Cycle's conditions, but:
b47428    b) There should be a vertex in the graph which has
0deace    (indegree+1==outdegree)
7673bb    d) There should be a vertex in the graph which has
2cbd0a    (indegree==outdegree+1)
438031    If either of the above condition fails Euler Path/Cycle
ec3bfc    can't exist.
b2ea28    OBS: The "connected" condition it's not explicit tested by
0709f0    the algorithm because it's enough checking the size of
605ae1    the found path.
d41d8c
8d6c7a    [COMPLEXITY]  $O(V + E)$ 
d41d8c
bd7472    [USAGE]
83c9e2    You should initialize the following global variables
0df1f1    [vertices]
b8d6ce    * 0-indexed
3e0b9c    * It's fine to include useless vertices
d41d8c
bb96a6    [edges]
941e2c    * In undirected graphs be sure that you created just
b65d7b    one edge and u,v have this edge in the outs vector.
d41d8c
1c211c    [n] number of total vertices (including useless)

```

```

3ed989             [m] number of total edges
d41d8c
aa9cd2             You should call init() before call euler_tour(n_edges), the
d8b1ae             n_edges argument is how many edges you are expecting to
d08570             traverse in the euler_tour/walk.
d41d8c
4da678             !!WARNING!!: Never modify the graph after calling init(),
d1d6dc             that could invalidate the references.
d41d8c
e25003             [return]
7b39ec             An integer vector that represents the vertices' indexes
25f361             of the found cycle (when exists) or the found path
3fbbe7             (when exists). If none was found, an empty vector is
7241ee             returned.
b9c9c5             You can change the return value to be an integer vector
8e2b34             that represent the edges' indexes.
abd2c1             OBS: You can check if the returned value is a path by
415167             checking if ret.front() != ret.back()
d41d8c
ea195c             [reset]
83ffe8             If the problem has several testcases, don't forget to
dfadbe             reset global vars
d41d8c
c4c9bd             */
d41d8c
5d1131             #include "../contest/header.hpp"
d41d8c
b01817             namespace euler
f95b70             {
d41d8c
f6b0b0             #define MAXM 112345
69abfb             #define MAXN 112345
d41d8c
729806             struct edge
f95b70             {
dfd8dc                 int u, v, id;
2145c1             };
d41d8c
a593f0             struct vertice
f95b70             {
1b2b2a                 vi outs;                // edges indexes
85a791                 int in_degree = 0; // not used with undirected graphs
2145c1             };
d41d8c
14e0a7             int n, m;
b811d5             edge edges[MAXM];
34e61c             vertice vertices[MAXN];
de77f5             vi::iterator its[MAXN];
291f7b             bool used_edge[MAXM];
d41d8c

```

```

b2a56e void init()
f95b70 {
83008c     for (int i = 0; i < n; i++)
f95b70     {
654e3c         its[i] = vertices[i].outs.begin();
cbb184     }
cbb184 }
d41d8c
76b369 vi euler_tour(int n_edges, int src)
f95b70 {
dc14c4     vi ret_vertices;
d41d8c     //vi ret_edges;
a424cb     vector<pii> s = {{src, -1}};
365295     while (!s.empty())
f95b70     {
448825         int x = s.back().first;
ad4be3         int e = s.back().second;
2e108b         auto &it = its[x], end = vertices[x].outs.end();
d41d8c
f7b756         while (it != end && used_edge[*it])
0b04e0             ++it;
d41d8c
ddad26         if (it == end)
f95b70         {
1e3129             ret_vertices.push_back(x);
d41d8c             //ret_edges.push_back(e);
342ca4             s.pop_back();
cbb184         }
2954e9         else
f95b70         {
82e7a6             auto edge = edges[*it];
27ff09             int v = edge.u == x ? edge.v : edge.u;
af50b8             s.push_back({v, *it});
101ac4             used_edge[*it] = true;
cbb184         }
cbb184     }
e07428     if (sz(ret_vertices) != n_edges + 1)
316ff2         ret_vertices.clear(); // No Eulerian cycles/paths.
d41d8c     /*
0fa901 if (sz(ret_edges) != n_edges)
99f9fa     ret_edges.clear(); // No Eulerian cycles/paths.
c4c9bd */
d41d8c
d41d8c     // Check if is cycle ret_vertices.front() == ret_vertices.back()
d41d8c
87de2e     reverse(all(ret_vertices));
32540c     return ret_vertices;
d41d8c
d41d8c     /*
e1eaea     reverse(all(ret_edges));

```



```
95f206    return ret_edges;
c4c9bd    */
cbb184   }
d41d8c
cbb184   } // namespace euler
Full file hash: a07957
```

4.7 Max Flow (Dinic)

```

be64a6 #include "../.../contest/header.hpp"
d41d8c
d41d8c /*
908d2f     Dinic:
67cbe4         Max-flow  $O(V^2E)$ 
eaeddf         Bipartite Matching  $O(\sqrt{V}E)$ 
d41d8c
ca2095     Constraints:
80b2d0         Vertices are labeled from 0 to n (inclusive).
8f4ce8         Edge capacities must fit int (flow returned is long long).
d41d8c
b95cae     Usage:
d86132         Set MAXV if necessary.
148d9c         Call init passing n, the source and the sink.
2d6398         Add edges to the graph by calling put_edge(_undirected).
fe33ff         Call max_flow to get the total flow. Then, individual edge
df7b62         flows can be retrieved in the graph.
22c3c2         Note that flow will be negative in return edges.
c4c9bd */
d41d8c
82657b namespace dinic
f95b70 {
729806 struct edge
f95b70 {
bf6256     int dest, cap, re, flow;
2145c1 };
d41d8c
998014 const int inf = 0x3f3f3f3f;
8550b5 const int MAXV = 312345;
d41d8c
8a367b int n, s, t, d[MAXV], q[MAXV], next[MAXV];
d8f9f2 vector<vector<edge>> graph;
d41d8c
bc6f23 void init(int _n, int _s, int _t)
f95b70 {
c992e9     n = _n, s = _s, t = _t;
b72d19     graph = vector<vector<edge>>(n + 1);
cbb184 }
d41d8c
7c85eb void put_edge(int u, int v, int cap)
f95b70 {
506964     graph[u].push_back({v, cap, (int)graph[v].size(), 0});
68ec95     graph[v].push_back({u, 0, (int)graph[u].size() - 1, 0});
cbb184 }
d41d8c
d6a592 void put_edge_undirected(int u, int v, int cap)
f95b70 {
506964     graph[u].push_back({v, cap, (int)graph[v].size(), 0});

```

```

fce495     graph[v].push_back({u, cap, (int)graph[u].size() - 1, 0});
cbb184 }
d41d8c
6a1cf9 bool bfs()
f95b70 {
18753f     int qb = 0, qe = 0;
3c6658     q[qe++] = s;
98fde3     memset(d, 0x3f, sizeof(int) * (n + 1));
d66185     d[s] = 0;
2caa87     while (qb < qe)
f95b70     {
e8e8a0         int a = q[qb++];
c9a55a         if (a == t)
8a6c14             return true;
3352c6         for (int i = 0; i < (int)graph[a].size(); i++)
f95b70         {
10e42b             edge &e = graph[a][i];
d948dd             if (e.cap - e.flow > 0 && d[e.dest] == inf)
f4063b                 d[q[qe++]] = e.dest = d[a] + 1;
cbb184         }
cbb184     }
d41d8c
d1fe4d     return false;
cbb184 }
d41d8c
1a19d4 int dfs(int a, int flow)
f95b70 {
c9a55a     if (a == t)
99d2e8         return flow;
10647a     for (int &i = next[a]; i < (int)graph[a].size(); i++)
f95b70     {
10e42b         edge &e = graph[a][i];
c6fb85         if (d[a] + 1 == d[e.dest] && e.cap - e.flow > 0)
f95b70         {
5f308a             int x = dfs(e.dest, min(flow, e.cap - e.flow));
5f75db             if (x == 0)
5e2bd7                 continue;
7f9751             e.flow += x;
4c55a5             graph[e.dest][e.re].flow -= x;
ea5659             return x;
cbb184         }
cbb184     }
d41d8c
343737     d[a] = inf;
bb30ba     return 0;
cbb184 }
d41d8c
afa2f7 long long max_flow()
f95b70 {
f013d3     long long total_flow = 0;

```

```
d594a7     while (bfs())
f95b70     {
ba90c2         memset(next, 0, sizeof(int) * (n + 1));
60616b         while (int path_flow = dfs(s, inf))
a0d8d9             total_flow += path_flow;
cbb184     }
d41d8c
793f63     return total_flow;
cbb184 }
cbb184 } // namespace dinic
Full file hash: 574d3a
```

4.8 Max Flow (Dinic w/ Scaling)

```

be64a6 #include "../.../contest/header.hpp"
d41d8c
d41d8c /*
3678e9     Dinic with Scaling:
8ac057         Max-flow  $O(VE * \log(\text{MAX\_CAP}))$ , but usually slower than regular
952d89         Dinic.
d41d8c
ca2095     Constraints:
80b2d0         Vertices are labeled from 0 to n (inclusive).
8f4ce8         Edge capacities must fit int (flow returned is long long).
d41d8c
b95cae     Usage:
d86132         Set MAXV if necessary.
148d9c         Call init passing n, the source and the sink.
2d6398         Add edges to the graph by calling put_edge(_undirected).
fe33ff         Call max_flow to get the total flow. Then, individual edge
df7b62         flows can be retrieved in the graph.
22c3c2         Note that flow will be negative in return edges.
c4c9bd */
d41d8c
82657b namespace dinic
f95b70 {
729806 struct edge
f95b70 {
bf6256     int dest, cap, re, flow;
2145c1 };
d41d8c
998014 const int inf = 0x3f3f3f3f;
8550b5 const int MAXV = 312345;
d41d8c
19c361 int n, s, t, lim, d[MAXV], q[MAXV], next[MAXV];
d8f9f2 vector<vector<edge>> graph;
d41d8c
bc6f23 void init(int _n, int _s, int _t)
f95b70 {
c992e9     n = _n, s = _s, t = _t;
b72d19     graph = vector<vector<edge>>(n + 1);
cbb184 }
d41d8c
7c85eb void put_edge(int u, int v, int cap)
f95b70 {
506964     graph[u].push_back({v, cap, (int)graph[v].size(), 0});
68ec95     graph[v].push_back({u, 0, (int)graph[u].size() - 1, 0});
cbb184 }
d41d8c
d6a592 void put_edge_undirected(int u, int v, int cap)
f95b70 {
506964     graph[u].push_back({v, cap, (int)graph[v].size(), 0});

```

```

fce495     graph[v].push_back({u, cap, (int)graph[u].size() - 1, 0});
cbb184 }
d41d8c
6a1cf9 bool bfs()
f95b70 {
18753f     int qb = 0, qe = 0;
3c6658     q[qe++] = s;
98fde3     memset(d, 0x3f, sizeof(int) * (n + 1));
d66185     d[s] = 0;
2caa87     while (qb < qe)
f95b70     {
e8e8a0         int a = q[qb++];
c9a55a         if (a == t)
8a6c14             return true;
3352c6         for (int i = 0; i < (int)graph[a].size(); i++)
f95b70         {
10e42b             edge &e = graph[a][i];
21aca9             if (e.cap - e.flow >= lim && d[e.dest] == inf)
f4063b                 d[q[qe++] = e.dest] = d[a] + 1;
cbb184         }
cbb184     }
d41d8c
d1fe4d     return false;
cbb184 }
d41d8c
1a19d4 int dfs(int a, int flow)
f95b70 {
c9a55a     if (a == t)
99d2e8         return flow;
10647a     for (int &i = next[a]; i < (int)graph[a].size(); i++)
f95b70     {
10e42b         edge &e = graph[a][i];
cbf046         if (d[a] + 1 == d[e.dest] && e.cap - e.flow >= lim /* >= 1 ? */)
f95b70         {
5f308a             int x = dfs(e.dest, min(flow, e.cap - e.flow));
5f75db             if (x == 0)
5e2bd7                 continue;
7f9751             e.flow += x;
4c55a5             graph[e.dest][e.re].flow -= x;
ea5659             return x;
cbb184         }
cbb184     }
d41d8c
343737     d[a] = inf;
bb30ba     return 0;
cbb184 }
d41d8c
afa2f7 long long max_flow()
f95b70 {
f013d3     long long total_flow = 0;

```

```
aab413     for (lim = (1 << 30); lim >= 1; lim >>= 1)
d594a7     while (bfs())
f95b70     {
ba90c2         memset(next, 0, sizeof(int) * (n + 1));
60616b         while (int path_flow = dfs(s, inf))
a0d8d9             total_flow += path_flow;
cbb184     }
d41d8c
793f63     return total_flow;
cbb184 }
cbb184 } // namespace dinic
Full file hash: ac7da7
```

4.9 Min Cost Max Flow

```

2b74fa  #include <bits/stdc++.h>
ca417d  using namespace std;
d41d8c
d41d8c  /*
dfc480      Min-Cost Max-Flow:  $O(V^2E^2)$ 
078f0d      Finds the maximum flow of minimum cost.
d41d8c
ca2095      Constraints:
80b2d0      Vertices are labeled from 0 to n (inclusive).
b018cb      Edge cost and capacities must fit int (flow and cost
3b8d65      returned are long long).
75ef18      Edge Cost must be non-negative.
d41d8c
b95cae      Usage:
d86132      Set MAXV if necessary.
148d9c      Call init passing n, the source and the sink.
909583      Add edges to the graph by calling put_edge.
92655a      Call mincost_maxflow to get the total flow and its cost
f08286      (in this order).
772475      Individual edge flows can be retrieved in the graph.
22c3c2      Note that flow will be negative in return edges.
c4c9bd  */
d41d8c
ad1153  typedef long long ll;
d29b14  typedef pair<long long, long long> pll;
d41d8c
e3de19  namespace mcmf
f95b70  {
729806  struct edge
f95b70  {
60f183      int dest, cap, re, cost, flow;
2145c1  };
d41d8c
ed5ed2  const int MAXV = 112345;
6a4c6c  const ll inll = 0x3f3f3f3f3f3f3f3fLL;
998014  const int inf = 0x3f3f3f3f;
d41d8c
128c92  int n, s, t, p[MAXV], e_used[MAXV];
97e5bb  bool in_queue[MAXV];
c97378  ll d[MAXV];
d41d8c
d8f9f2  vector<vector<edge>> graph;
d41d8c
bc6f23  void init(int _n, int _s, int _t)
f95b70  {
c992e9      n = _n, s = _s, t = _t;
b72d19      graph = vector<vector<edge>>(n + 1);
cbb184  }

```



```

d41d8c
a4abfa void put_edge(int u, int v, int cap, int cost)
f95b70 {
bd3e8b     graph[u].push_back({v, cap, (int)graph[v].size(), cost, 0});
2b8b8d     graph[v].push_back({u, 0, (int)graph[u].size() - 1, -cost, 0});
cbb184 }
d41d8c
b34984 bool spfa()
f95b70 {
664c61     memset(in_queue, 0, sizeof(bool) * (n + 1));
9eef50     memset(d, 0x3f, sizeof(ll) * (n + 1));
26a528     queue<int> q;
d66185     d[s] = 0;
e2828b     p[s] = s;
08bec3     q.push(s);
ee6bdd     while (!q.empty())
f95b70     {
0930a5         int a = q.front();
833270         q.pop();
e7249b         in_queue[a] = false;
d41d8c
c4fff3         for (int i = 0; i < graph[a].size(); i++)
f95b70         {
10e42b             edge &e = graph[a][i];
6fa321             if (e.cap - e.flow > 0 && d[e.dest] > d[a] + e.cost)
f95b70             {
3bf598                 d[e.dest] = d[a] + e.cost;
6d6530                 p[e.dest] = a;
183d83                 e_used[e.dest] = i;
27788c                 if (!in_queue[e.dest])
b34293                     q.push(e.dest);
04f0f7                 in_queue[e.dest] = true;
cbb184             }
cbb184         }
cbb184     }
d41d8c
d1cd45     return d[t] < infll;
cbb184 }
d41d8c
99658d pll mincost_maxflow()
f95b70 {
f04b2a     pll retv = pll(0, 0);
d9383f     while (spfa())
f95b70     {
e98031         int x = inf;
c9b315         for (int i = t; p[i] != i; i = p[i])
d4a316             x = min(x, graph[p[i]][e_used[i]].cap - graph[p[i]][e_used[i]].
flow);
c9b315         for (int i = t; p[i] != i; i = p[i])
dc731d             graph[p[i]][e_used[i]].flow += x, graph[i][graph[p[i]][e_used[i]

```

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

4.10 Gomory Hu (Min cut)

```

5e41d6 #include "../flow/dinic/dinic.cpp"
d41d8c
d41d8c /*
ecc690     Gomory-Hu Tree construction  $O(V * \text{flow\_time})$  (so  $O(V^3E)$ , but not
72bf5a     really):
854631     The Gomory-Hu tree of an undirected graph with capacities is a
4faf3f     weighted
33f64b     tree that represents the minimum s-t cuts for all s-t pairs in
676894     the graph.
d41d8c
e0f147     The minimum cut cost between vertices s and t is the minimum
63a829     cost of an edge on the path from s to t in the Gomory-Hu tree.
d41d8c
ca2095     Constraints:
ea5b90     Vertices are labeled from 0 to n-1 (inclusive).
ecd5dd     Undirected graph.
d41d8c
b95cae     Usage:
44294c     Check Dinic usage.
a7923f     Create struct and call add edge for each edge in the graph.
53b540     Then, just call solve passing the number of vertices.
d41d8c
9949f6     The vector returned will have size n and for each  $i > 0$ ,
41f0b1     retv[i] is a pair (cost, parent) representing an edge
c6df4c     (i, parent) in the Gomory-Hu tree.
2123c0     retv[0] means nothing.
c4c9bd */
d41d8c
2a4db0 struct gomory_hu
f95b70 {
a2074f     struct edg
f95b70     {
765cb8         int u, v, cap;
2145c1     };
d41d8c
c4f7a0     vector<edg> eds;
d41d8c
3dd40c     void add_edge(int u, int v, int cap)
f95b70     {
265dad         eds.push_back({u, v, cap});
cbb184     }
d41d8c
051dd2     vector<int> vis;
d41d8c
0cbab3     void dfs(int a)
f95b70     {
cd2c1e         if (vis[a])
505b97             return;

```

```

18f129     vis[a] = 1;
264d9c     for (auto &e : dinic::graph[a])
0f54a3         if (e.cap - e.flow > 0)
7cafa5             dfs(e.dest);
cbb184     }
d41d8c
242d7b     vector<pair<ll, int>> solve(int n)
f95b70     {
56a8c7         vector<pair<ll, int>> retv(n); // if i > 0, stores pair(cost,
d41d8c             // parent).
aa4866         for (int i = 1; i < n; i++)
f95b70             {
93cfa5                 dinic::init(n, i, retv[i].second);
d41d8c
9e8d8e                 for (auto &e : eds)
8933a2                     dinic::put_edge_undirected(e.u, e.v, e.cap);
d41d8c
1809f8                 retv[i].first = dinic::max_flow();
d41d8c
1059c6                 vis.assign(n, 0);
1e5da3                 dfs(i);
d41d8c
197ab1                 for (int j = i + 1; j < n; j++)
a32820                     if (retv[j].second == retv[i].second && vis[j])
9cc152                         retv[j].second = i;
cbb184             }
d41d8c
6272cf         return retv;
cbb184     }
2145c1 };
Full file hash: 3fe14c

```

4.11 Heavy-Light Decomposition

```

2b74fa #include<bits/stdc++.h>
d41d8c
ca417d using namespace std;
d41d8c
eed838 #define ll long long
efe13e #define pb push_back
d41d8c
3a6c63 typedef vector<ll> vll;
990dd5 typedef vector<int> vi;
d41d8c
e06cc0 #define MAXN 100010
d41d8c
d41d8c //Vetor que guarda a arvore
698e25 vector<vi> adj;
d41d8c
9e6e6d int subsize[MAXN], parent[MAXN];
d41d8c //Inciar chainHead com -1; e chainSize e chainNo com 0.
080553 int chainNo = 0, chainHead[MAXN], chainPos[MAXN], chainInd[MAXN],
    chainSize[MAXN];
42a605 void hld(int cur){
cb42fb     if(chainHead[chainNo] == -1)
6591fe         chainHead[chainNo] = cur;
d41d8c
3a4605     chainInd[cur] = chainNo;
220e91     chainPos[cur] = chainSize[chainNo];
6f00fb     chainSize[chainNo]++;
d41d8c
89108d     int ind = -1, mai = -1;
9d9afd     for(int i = 0; i < (int)adj[cur].size(); i++){
9ff2fa         if(adj[cur][i] != parent[cur] && subsize[adj[cur][i]] > mai){
31fcc6             mai = subsize[adj[cur][i]];
b9b7e9             ind = i;
cbb184         }
cbb184     }
d41d8c
27d206     if(ind >= 0)
f23581         hld(adj[cur][ind]);
d41d8c
e506c6     for(int i = 0; i < (int)adj[cur].size(); i++)
6f7286         if(adj[cur][i] != parent[cur] && i != ind){
959ef6             chainNo++;
270563             hld(adj[cur][i]);
cbb184         }
cbb184     }
d41d8c
d41d8c //usar LCA para garantir que v eh pai de u!!
f179f7 ll query_up(int u, int v){
c20c7b     int uchain = chainInd[u], vchain = chainInd[v];

```

```

bdd5ea    ll ans = 0LL;
d41d8c
31e3cd    while(1){
f523c5        if(uchain == vchain){
d41d8c            //Query deve ir de chainPos[i] ate chainPos[v]
7d2150            ll cur = /*sum(chainPos[u], uchain) - (chainPos[u] == 0? 0LL :
                sum(chainPos[v] - 1, vchain))*/;
d133d8            ans += cur;
c2bef1            break;
cbb184        }
d41d8c
d41d8c            //Query deve ir de chainPos[i] ate o fim da estrutura
d41d8c            //ll cur = sum(chainPos[u], uchain);
d133d8            ans += cur;
a258cd            u = chainHead[uchain];
8039e1            u = parent[u];
cab24            uchain = chainInd[u];
cbb184        }
ba75d2    return ans;
cbb184    }
d41d8c
b7aa64    int dfs0(int pos, int prev = -1){
c92501        int res = 1;
97817b        for(int i = 0; i < (int)adj[pos].size(); i++){
ec49a3            int nx = adj[pos][i];
773904            if(nx != prev){
3f20e3                res += dfs0(nx, pos);
522845                parent[nx] = pos;
cbb184            }
cbb184        }
a1881c        return subsize[pos] = res;
cbb184    }
d41d8c
0b8977    int main()
f95b70    {
d41d8c        //Salvar arvore em adj
d41d8c
d41d8c        //Inicializa estrutura de dados
b75143        memset(chainHead, -1, sizeof(chainHead));
d41d8c
d41d8c        //Ou 0, se for o no raiz
bf6cde        dfs0(1);
bac429        hld(1);
d41d8c
d41d8c        //Inicializar estruturas usadas
cbb184    }

```

Full file hash: 90a698

4.12 Heavy-Light Decomposition (Dadalto)

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
e4cb61     Heavy Light Decomposition:
ea0e49         Splits a tree in a set of vertex disjoint heavy paths such
53c140         that each path from a node to the root passes at most log(n)
82946f         different heavy paths.
1abdb9         This allows data structures to be implement with queries and
db47e8         updates on tree paths in log(n) * data_structure_time.
d41d8c
b95cae     Usage:
bfc15c         Create the struct passing a tree root (a), the number of
8d22ce         vertices (n) and the graph. Tested with 1 <= a <= n, but
176e5c         should work with 0 <= a <= n.
d41d8c
b8d87e         The data structure DS class should implement single element
d42e50         updates or range updates as needed and range queries
9f615c         according to the form defined in update, update_path and
20b318         query_path.
1042ee         DS should also have a constructor specifying the size
e5e1f5         and should support operations in range [0, size - 1].
b6dfb7         IMPORTANT: DS should handle empty queries [x + 1, x].
11cc96         IMPORTANT: function applied in DS should be commutative
1856b6         and associative. (If not commutative, check out application
ac41a9         for GSS7)
d41d8c
c91108         VALUES_IN_VERTICES indicates if the tree values are in vertices
15746a         or in edges. In case of edges, update(v, value)
fcf492         should be called for the downward vertex of each edge.
d41d8c
fb2d4d         See application for more information.
d41d8c
6e9cc1         Source: adapted from codeforces blog (https://codeforces.com/blog/entry/22072).
6995a2
c4c9bd */
d41d8c
2cd8f8 template <class DS, bool VALUES_IN_VERTICES> // DS for data structure
.
d41d8c         // Values in vertices,
d41d8c         // true or false.
62b5c3 struct heavy_light
f95b70 {
119f46     vector<int> p, heavy, h; // parent, heavy child of vertex,
d41d8c         // height of vertex.
d41d8c
fbfa48     vector<int> num;         // number of vertex (in an order where
d41d8c         // paths are contiguous intervals).
d41d8c

```

```

be732c    vector<int> root;    // root of heavy path of a given vertex.
ddc355    DS ds;
d41d8c
e5fca3    template <class G>
c10e74    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)
f95b70    {
42e4d3        p[a] = a;
d3d3d3        h[a] = 0;
57e7a4        dfs(graph, a);
cad977        for (int i = 0, id = 0; i <= n; ++i)
3c79d2            if (heavy[p[i]] != i) // parent of the root is itself,
d41d8c                // so this works.
fc871b                for (int j = i; j != -1; j = heavy[j])
f95b70                    {
6d94a8                        root[j] = i;
9c5b22                        num[j] = id++;
cbb184                    }
cbb184    }
d41d8c
e5fca3    template <class G>
57ebd6    int dfs(const G &graph, int a)
f95b70    {
d0a541        int size = 1, max_subtree = 0;
23e548        for (int u : graph[a])
88c1b4            if (u != p[a])
f95b70                {
6c309e                    p[u] = a;
adab56                    h[u] = h[a] + 1;
c1c8cc                    int subtree = dfs(graph, u);
9eaa96                    if (subtree > max_subtree)
d98545                        heavy[a] = u, max_subtree = subtree;
48ec46                    size += subtree;
cbb184                }
1c6620        return size;
cbb184    }
d41d8c
2f712c    template <class B0> // B0 for binary_operation
72abf1    void process_path(int u, int v, B0 op)
f95b70    {
d42a19        for (; root[u] != root[v]; v = p[root[v]])
f95b70            {
2ceb71                if (h[root[u]] > h[root[v]])
7fa1c7                    swap(u, v);
857eb5                op(num[root[v]], num[v]);
cbb184            }
ce9c7a            if (h[u] > h[v])
7fa1c7                swap(u, v);
f0b977            op(num[u] + (VALUES_IN_VERTICES ? 0 : 1), num[v]);
cbb184    }

```



```

d41d8c
4fce64     template <class T>
44956b     void update(int v, const T &value)
f95b70     {
2bf133         ds.update(num[v], value);
cbb184     }
d41d8c
4fce64     template <class T>
67d27b     T query(int v)
f95b70     {
678f6c         return ds.get(num[v], num[v]);
cbb184     }
d41d8c
4fce64     template <class T>
47b962     void update_path(int u, int v, const T &value)
f95b70     {
bef37b         process_path(u, v, [this, &value](int l, int r) { ds.update(l, r,
            value); });
cbb184     }
d41d8c
af34ab     template <class T, class F>
ed6928     T query_path(int u, int v, T res /* initial value */, F join /*
        join value with query result */)
f95b70     {
96862e         process_path(u, v, [this, &res, &join](int l, int r) { res = join
            (res, ds.get(l, r)); });
b5053e         return res;
cbb184     }
2145c1     };
Full file hash: 80b4be

```

4.13 LCA

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
b471c1     LCA:
0e9577         Solve lowest common ancestor queries in  $O(\log(n))$ 
19ea7d         with  $O(n \cdot \log(n))$  preprocessing time and  $O(n \cdot \log(n))$ 
c6023d         memory.
d41d8c
b95cae     Usage:
a13ce5         Initialize struct with tree root, number of vertices
ed52e1         and graph. Has been tested with label in  $[1, n]$ , but should
a7e3c3         work for labels in  $[0, n]$ .
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
4bef34 struct lca_preprocess
f95b70 {
6c6155     int lgn;
5446c7     vector<int> h;
58b4fb     vector<vector<int>> p;
9cf2b9     vector<int> *graph;
d41d8c
0cbab3     void dfs(int a)
f95b70     {
1429ef         for (int i = 0; i < sz(graph[a]); i++)
cb4b44             if (graph[a][i] != p[0][a])
f95b70                 {
6f498f                     h[graph[a][i]] = h[a] + 1;
7a7911                     p[0][graph[a][i]] = a;
fded7f                     dfs(graph[a][i]);
cbb184                 }
cbb184     }
d41d8c
cf2401     lca_preprocess(int root, int n, vector<int> graph[]) : h(n + 1),
graph(graph)
f95b70     {
5ff10e         lgn = 31 - __builtin_clz(n + 1);
44565e         p.assign(lgn + 1, vector<int>(n + 1, 0));
d41d8c
8c92d0         p[0][root] = root;
7a0e2c         h[root] = 0;
14eacc         dfs(root);
d41d8c
05d1e4         for (int i = 1; i <= lgn; i++)
f630b0             for (int j = 0; j <= n; j++)
98f5dd                 p[i][j] = p[i - 1][p[i - 1][j]];
cbb184     }

```

```
d41d8c
4cdef7  int lca(int a, int b)
f95b70  {
be7ea8    if (h[a] < h[b])
2574c6    swap(a, b);
29eca1    for (int i = lgn; i >= 0; i--)
a9762b        if (h[p[i][a]] >= h[b])
e27ea8            a = p[i][a];
d41d8c
ae993e    if (a == b)
3f5343        return a;
d41d8c
29eca1    for (int i = lgn; i >= 0; i--)
e5d3c7        if (p[i][a] != p[i][b])
f95b70        {
e27ea8            a = p[i][a];
6341e6            b = p[i][b];
cbb184        }
d41d8c
d12c09    return p[0][a];
cbb184  }
d41d8c
d2d513  int dist(int a, int b)
f95b70  {
7181e5    return h[a] + h[b] - 2 * h[lca(a, b)];
cbb184  }
2145c1  };
Full file hash: 0872ca
```

4.14 Min-Cut Global

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
bed9d1     Global Min Cut  $O(n^3)$ :
b25fda         Given an undirected weighted graph, find the minimum cut
365364         regardless of which set of vertices it splits.
d41d8c
b95cae     Usage:
ec5bb0         Vertices from 0 to n-1. Give adjacency matrix of weights.
18fcb4         Function returns total cost of min cut and set of vertices
50b008         forming one side of the cut.
dd8e36         Sum of edge weights must fit int.
c4c9bd */
d41d8c
320f95 pair<int, vector<int>> mincut(int n, vector<vector<int>> g /*adj
matrix*/)
f95b70 {
d9afb8     int best_cost = inf;
6eb913     vector<int> best_cut;
d41d8c
c91f78     vector<int> v[n];
6cb8cc     for (int i = 0; i < n; ++i)
9f7778         v[i].assign(1, i);
21413c     int w[n];
9ecfcd     bool exist[n], in_a[n];
99e342     memset(exist, true, sizeof(exist));
f34066     for (int ph = 0; ph < n - 1; ++ph)
f95b70     {
9e8357         memset(in_a, false, sizeof(in_a));
e3eac6         memset(w, 0, sizeof(w));
548ac4         for (int it = 0, prev; it < n - ph; ++it)
f95b70         {
0a8883             int sel = -1;
6cb8cc             for (int i = 0; i < n; ++i)
1b38c0                 if (exist[i] && !in_a[i] && (sel == -1 || w[i] > w[sel]))
403e4b                     sel = i;
cb8b03             if (it == n - ph - 1)
f95b70             {
25f99c                 if (w[sel] < best_cost)
5e02d0                     best_cost = w[sel], best_cut = v[sel];
899cd6                 v[prev].insert(v[prev].end(), v[sel].begin(), v[sel].end());
6cb8cc                 for (int i = 0; i < n; ++i)
d18d9d                     g[prev][i] = g[i][prev] += g[sel][i];
0e82f8                 exist[sel] = false;
cbb184             }
2954e9             else
f95b70             {
96ae14                 in_a[sel] = true;

```

```
6cb8cc         for (int i = 0; i < n; ++i)
f57b34             w[i] += g[sel][i];
0b7e30         prev = sel;
cbb184         }
cbb184     }
cbb184 }
d41d8c
bc167d     return pair<int, vector<int>>(best_cost, best_cut);
cbb184 }
d41d8c
Full file hash: 40ea3a
```

4.15 Strongly Connected Components

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
eaba86     Strongly connected components in  $O(V + E)$ :
970b0b     Finds all strongly connected components of a graph.
ec1d79     A strongly connected component is a maximal set of vertices
5e5ff3     such that
de0185     every vertex can reach every other vertex in the component.
0a2f52     The graph where the SCCs are considered vertices is a DAG.
d41d8c
ca2095     Constraints:
000269     Vertices are labeled from 1 to n (inclusive).
d41d8c
b95cae     Usage:
049cff     Create the struct setting the maximum vertex label (n) and the
51006d     graph adjacency list (graph).
ee4ba7     Afterwards, ncomp has the number of SCCs in the graph and
d13654     scc[i] indicates the SCC i belongs to ( $1 \leq \text{scc}[i] \leq \text{ncomp}$ ).
d41d8c
57d50c     sorted is a topological ordering of the graph, byproduct of
b1c425     the algorithm.
484a00     if edge  $a \rightarrow b$  exists, a appears before b in the sorted list.
c4c9bd */
d41d8c
d41d8c
73e60a struct scc_decomp
f95b70 {
9cf2b9     vector<int> *graph;
00b6a0     vector<vector<int>> tgraph;
1b013f     vector<int> scc;
1ee615     vector<bool> been;
8d35d1     int ncomp;
2035f3     list<int> sorted;
d41d8c
4875fb     scc_decomp(int n, vector<int> graph[]) : graph(graph), tgraph(n +
    1), scc(n + 1, 0), been(n + 1, false), ncomp(0)
f95b70     {
5359f3         for (int i = 1; i <= n; i++)
6376e8             for (int j = 0; j < graph[i].size(); j++)
14234d                 tgraph[graph[i][j]].push_back(i);
d41d8c
5359f3         for (int i = 1; i <= n; i++)
018df6             if(!been[i])
1e5da3                 dfs(i);
d41d8c
16ef86         for(int a : sorted)
f49735             if(scc[a] == 0)
f95b70             {

```

```
a8f1f2                ncomp++;
4dd966                dfst(a);
cbb184                }
cbb184                }
d41d8c
0cbab3    void dfs(int a)
f95b70    {
1689c6        been[a] = true;
c4fff3        for(int i = 0; i < graph[a].size(); i++)
b0c443            if(!been[graph[a][i]])
fded7f                dfs(graph[a][i]);
ddb66        sorted.push_front(a);
cbb184    }
d41d8c
9b760a    void dfst(int a)
f95b70    {
1689c6        been[a] = true;
d28fcc        scc[a] = ncomp;
9c28b7        for(int i = 0; i < tgraph[a].size(); i++)
c480c5            if(scc[tgraph[a][i]] == 0)
caa482                dfst(tgraph[a][i]);
cbb184    }
2145c1 };
Full file hash: 20fe5c
```

4.16 Transitive Closure

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
a271e3     Transitive Closure:
aef5f7         Given a directed graph adjacency matrix, computes closure,
8d58ff         where closure[i][j] = 1 if there is a path from i to j
3fb41d         in the graph.
9e9ad9         Closure is computed in  $O(N^3 / 64)$  due to bitset.
be8877         Also supports adding an edge to the graph and
bf860b         updating the closure accordingly in  $O(N^2 / 64)$ .
d41d8c
ca2095     Constraints:
a5b156         Vertices are labeled from 0 to MAXN - 1 (inclusive).
d41d8c
faf237     Performance:
45d541         Solves something that should be  $1000 \times 300^3 = 27 \times 10^9$ 
061bbf         in 0.6 s (which is consistent with the approximation  $N^3 / 64$ 
8c7ca4         since dividing by 64 we get  $4 \times 10^8$ ).
c4c9bd */
d41d8c
09748b template<int MAXN>
1256bb struct transitive_closure
f95b70 {
680ec7     vector<bitset<MAXN>> closure;
d41d8c
4fce64     template<class T>
5a6965     transitive_closure(T adj_matrix) : closure(MAXN)
f95b70     {
8898d6         for (int i = 0; i < MAXN; i++)
4f1bb9             for (int j = 0; j < MAXN; j++)
69c4b3                 closure[i][j] = adj_matrix[i][j];
d41d8c
8898d6         for (int i = 0; i < MAXN; i++)
4f1bb9             for (int j = 0; j < MAXN; j++)
459b6b                 if (closure[j][i])
cbb028                     closure[j] |= closure[i];
cbb184     }
d41d8c
a7abf7     void add_edge(int a, int b)
f95b70     {
5de576         if (closure[a][b])
505b97             return;
d41d8c
ec483a         closure[a].set(b);
84196c         closure[a] |= closure[b];
d41d8c
8898d6         for (int i = 0; i < MAXN; i++)
d0ec12             if (closure[i][a])

```



```
245a4a      closure[i] |= closure[a];
cbb184      }
2145c1    };
d41d8c
Full file hash: 1aed34
```

4.17 Tree Isomorphism

```

5d1131 #include "../..//contest/header.hpp"
d41d8c /*
ca5da8     [DEFINITION]
4294c0         AHU-Algorithm to check if trees are isomorphic.
d41d8c
ff59a1     [COMPLEXITY]
a0c567         O(NlgN) // Map of strings argument + comparison-based sort
d41d8c
bd7472     [USAGE]
fd7b37         Call get_roots function to retrieve the pairs of centers for
af1fff         each tree (if the tree has just one center the pair will show
6cdd44         it twice).
de3d73         Call canonical function for each tree beginning from each
aa6964         possible center (two at most).
9b5a77         A tree is isomorphic to another iff they share one canonical
cdf966         value.
d41d8c
e7fdf7     [RESET]
f288e6         If the problem has several test cases, don't forget to reset
a6587e         the global vars 'label' and 'map_labels'
c4c9bd */
d41d8c
70eeca int label;
4ec6f8 map<vector<int>, int> map_labels;
d41d8c
a46dbb pii get_roots(vector<vector<int>> &graph)
f95b70 {
26a528     queue<int> q;
dceec4     vector<int> vis(sz(graph));
275b7a     vector<int> degree(sz(graph));
d41d8c
28e6fb     for (int i = 0; i < sz(graph); i++)
f95b70     {
f789fd         if (sz(graph[i]) == 1)
3f2886             q.push(i);
902e7a         degree[i] = sz(graph[i]);
cbb184     }
d41d8c
5c4bd8     int last = 0;
ee6bdd     while (!q.empty())
f95b70     {
e4a6a5         int u = q.front();
833270         q.pop();
d41d8c
497028         if (vis[u]) continue;
150c99         vis[u] = 1;
d41d8c
2fb98d         last = u;

```

```

d41d8c
e132d9     for (int v : graph[u])
f95b70     {
e9be50         if (degree[v] == 1)
f95b70         {
0210ef             return {u, v};
cbb184         }
c2d124         if (!vis[v])
f95b70         {
a43ae2             degree[u]--;
7a96a2             degree[v]--;
d41d8c
e9be50             if (degree[v] == 1)
2a1edb                 q.push(v);
cbb184         }
cbb184     }
d41d8c
cbb184 }
d41d8c
a90965     return {last, last};
cbb184 }
d41d8c
c449b2 int canonical(int u, int p, vector<vi> &graph)
f95b70 {
08610e     vi children_labels;
e132d9     for (int v : graph[u])
f95b70     {
f6ba43         if (v != p)
d4e118             children_labels.push_back(canonical(v, u, graph));
cbb184     }
d41d8c
7601ff     sort(all(children_labels));
08dbfd     if (map_labels.count(children_labels) == 0)
d0497d         map_labels[children_labels] = label++;
58d4fa     return map_labels[children_labels];
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

```

d41d8c // https://codeforces.com/contest/321/problem/E
d41d8c
5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
4e8f0c     Binary Search Optimization for DP:
00098e         Optimizes dp of the form (or similar)
7ddd02             dp[i][j] = min_{k < i}(dp[k][j-1] + c(k + 1, i)).
95ea11             The classical case is a partitioning dp, where k determines
b5b277             the break point for the next partition.
8c2350             In this case, i is the number of elements to partition and j
5ccc64             is the number of partitions allowed.
d41d8c
f2404a             Let opt[i][j] be the values of k which minimize the function.
7e9cfd             (in case of tie, choose the smallest)
765123             To apply this optimization, you need opt[i][j] <= opt[i+1][j].
ef8cf9             That means the when you add an extra element (i + 1), your
05f29f             partitioning choice will not be to include more elements
cb72e7             than before (e.g. will no go from choosing [k, i] to
218463             [k-1, i+1]).
242554             This is usually intuitive by the problem details.
d41d8c
4d883e             Time goes from  $O(n^2m)$  to  $O(nm \log n)$ .
d41d8c
513656             To apply try to write the dp in the format above and verify if
499d07             the property holds.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
3494e9 #define MAXN 4123
fc36c1 #define MAXM 812
d41d8c
14e0a7 int n, m;
1590eb int u[MAXN][MAXN];
2bbe6d int tab[MAXN][MAXM];
d41d8c
65a7b7 inline int c(int i, int j)
f95b70 {
229880     return (u[j][j] - u[j][i - 1] - u[i - 1][j] + u[i - 1][i - 1]) / 2;
cbb184 }
d41d8c
d41d8c // This is responsible for computing tab[l...r][j], knowing that
d41d8c // opt[l...r][j] is in range [low_opt...high_opt]
30d71a void compute(int j, int l, int r, int low_opt, int high_opt)
f95b70 {
c30a4b     int mid = (l + r) / 2, opt = -1; // mid is equivalent to i in the
d41d8c         //original dp.

```

```

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

```

5.3 DP Optimization - CHT

```

d41d8c // https://codeforces.com/contest/319/problem/C
d41d8c
ad67d1 #include "../data_structures/line_container/line_container.cpp"
d41d8c
d41d8c /*
082e10     Convex Hull Trick for DP:
5cb9e4         Transforms dp of the form (or similar)
d9020a             dp[i] = min_{j < i}(dp[j] + b[j] * a[i]).
bf02bc             Time goes from O(n^2) to O(n log n), if using online line
cdb786             container, or O(n) if lines are inserted in order of slope and
0e64e2             queried in order of x.
d41d8c
62ec4f             To apply try to find a way to write the factor inside
ea034f             minimization as a linear function of a value related to i.
c2dab7             Everything else related to j will become constant.
c4c9bd */
d41d8c
69abfb #define MAXN 112345
d41d8c
a58cd5 int a[MAXN];
c4b25f int b[MAXN];
d41d8c
f80900 ll tab[MAXN];
d41d8c
13a4b1 int main(void)
f95b70 {
1a88fd     int n;
f4c120     scanf("%d", &n);
83008c     for (int i = 0; i < n; i++)
9376f3         scanf("%d", &a[i]);
83008c     for (int i = 0; i < n; i++)
264aeb         scanf("%d", &b[i]);
d41d8c
a447b8     tab[0] = 0;
79ab5f     line_container l;
c01116     l.add(-b[0], -tab[0]);
d41d8c
aa4866     for (int i = 1; i < n; i++)
f95b70     {
23bd61         tab[i] = -l.query(a[i]);
8fd447         l.add(-b[i], -tab[i]);
cbb184     }
d41d8c
d41d8c     // Original DP O(n^2).
d41d8c     // for (int i = 1; i < n; i++)
d41d8c     // {
d41d8c     //     tab[i] = inf;
d41d8c     //     for (int j = 0; j < i; j++)

```

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


5.4 DP Optimization - Knuth

```

d41d8c // https://www.spoj.com/problems/BRKSTRNG/
d41d8c
5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
c97188     Knuth Optimization for DP:
00098e         Optimizes dp of the form (or similar)
06c39b             dp[i][j] =
572201                 min_{i <= k <= j}(dp[i][k-1] + dp[k+1][j] + c(i, j)).
a05ba8             The classical case is building a optimal binary tree, where k
fd0ee5             determines the root.
d41d8c
90e95c             Let opt[i][j] be the value of k which minimizes the function.
7e9cfd             (in case of tie, choose the smallest)
d68e79             To apply this optimization, you need
e46fb4                 opt[i][j - 1] <= opt[i][j] <= opt[i+1][j].
7c335d             That means the when you remove an element form the left
a295f1             (i + 1), you won't choose a breaking point more to the left
f8ba18             than before.
287a65             Also, when you remove an element from the right (j - 1), you
bfb1c5             won't choose a breking point more to the right than before.
242554             This is usually intuitive by the problem details.
d41d8c
cbb42a             Time goes from  $O(n^3)$  to  $O(n^2)$ .
d41d8c
513656             To apply try to write the dp in the format above and verify if
499d07             the property holds.
f76c09             Be careful with edge cases for opt.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
dbf7e4 #define MAXN 1123
d41d8c
c4b25f int b[MAXN];
1ee552 ll tab[MAXN][MAXN];
38ab0d int opt[MAXN][MAXN];
ef864b int l, n;
d41d8c
5a7750 int c(int i, int j)
f95b70 {
33e24b     return b[j + 1] - b[i - 1];
cbb184 }
d41d8c
13a4b1 int main(void)
f95b70 {
57a598     while (scanf("%d %d", &l, &n) != EOF)
f95b70     {

```

```

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

```

5.5 MOs

```

5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
5c78f8     Mo's Algorithm:
8820f1         Solve Q interval queries on a sequence of N values offline
826523         in O(N * sqrt(Q) * max(insertion time, removal time)).
d41d8c
b95cae     Usage:
adcbb9         Queries are defined by closed intervals
9b7fa6         [l, r] (1 <= l <= r <= n).
3a92e8         add(i) must add i-th element to your data structure
802f74         (1 <= i <= n).
c8d9fb         remove(i) must remove the i-th element (1 <= i <= n).
751ba3         output(id) should answer query with given id using current
27afba         state.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
806e2c struct query {
425e69     int l, r, id;
2145c1 };
d41d8c
044d82 template<class F>
e5bd28 void mos(int n, vector<query> q, const F &add, const F &remove, const
      F &output)
f95b70 {
243fd1     int bsize = 1 + n / sqrt(sz(q));
c33690     sort(q.begin(), q.end(), [&](const query &lhs, const query &rhs) {
6b713e         if (lhs.l / bsize != rhs.l / bsize)
4d6f04             return lhs.l < rhs.l;
5d9b79         if ((lhs.l / bsize) & 1)
ee7306             return (lhs.r > rhs.r);
20f134         return (lhs.r < rhs.r);
c0c97e     });
d41d8c
00ad54     int l = 1, r = 0; // int l = 0, r = -1; (if indices starts at 0)
a0a6ed     for (int i = 0; i < sz(q); i++)
f95b70     {
dc4c37         while (l > q[i].l)
194467             add(--l);
583e51         while (r < q[i].r)
3dcc02             add(++r);
4fdd74         while (l < q[i].l)
87e33e             remove(l++);
5dccbb         while (r > q[i].r)
8b6a11             remove(r--);
d41d8c

```

```
2d80c3      output(q[i].id);
cbb184      }
cbb184  }
Full file hash: 0cbc87
```

5.6 MOs – Tree (Edge Query)

```

a34983 #include "../..graph/lca/lca.cpp"
d41d8c
d41d8c /*
7418d3     Refer to the vertex queries option for more info.
aef741     This version is different since it is made to handle queries on
babd44     the cost of the edges in a path.
d41d8c
a7682c     To do that, transfer the cost to the vertex down in the rooted
db6efd     tree and use this.
d41d8c
89d138     Remember to use a valid value for the root (even if it never will
c00b42     be in a query).
08e268     Also remember some queries will be empty.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
fa0e65 struct query
f95b70 {
6ac794     int l, r, id, lc;
2145c1 };
d41d8c
044d82 template <class F>
23d154 void mos_tree(int root, int n, vector<pii> pq, vector<int> graph[],
      const F &add, const F &remove, const F &output)
f95b70 {
ba2a0c     int a, b;
32ee7e     lca_preprocess lca(root, n, graph);
d41d8c
ad35dc     vector<int> st(n + 1, 0), en(n + 1, 0), v(2 * n + 3, 0), cnt(n + 1,
      0), s;
04c09b     int id = 0;
45fa16     s.push_back(root);
365295     while (!s.empty())
f95b70     {
2ecd18         a = s.back();
342ca4         s.pop_back();
d41d8c
f4f8d2         if (st[a])
2e4e50             v[en[a] = ++id] = a;
2954e9         else
f95b70         {
bab970             v[st[a] = ++id] = a;
bcc44e             s.push_back(a);
1429ef             for (int i = 0; i < sz(graph[a]); i++)
9d7e46                 if (graph[a][i] != lca.p[0][a])
bbdabd                     s.push_back(graph[a][i]);
cbb184         }

```

```

cbb184     }
d41d8c
4007f6     vector<query> q;
d41d8c
1605f4     for (int i = 0; i < sz(pq); i++)
f95b70     {
5ca0a2         tie(a, b) = pq[i];
d41d8c
a917ad         if (st[a] > st[b])
2574c6             swap(a, b);
3f022f         int y = lca.lca(a, b);
84ab82         if (a == y)
173360             q.push_back({st[a], st[b], i, st[y]});
2954e9         else
e653b6             q.push_back({en[a], st[b], i, -1});
cbb184     }
d41d8c
7dd140     int bsize = 1 + (2 * n) / sqrt(sz(q));
c33690     sort(q.begin(), q.end(), [&](const query &lhs, const query &rhs) {
6b713e         if (lhs.l / bsize != rhs.l / bsize)
7124a1             return (lhs.l / bsize < rhs.l / bsize);
1e42a4         return lhs.r < rhs.r;
c0c97e     });
d41d8c
568e70     auto consider = [&](int i) {
e8ecf4         cnt[v[i]]++;
9308c6         if (cnt[v[i]] % 2 == 1)
9a7780             add(v[i]);
2954e9         else
8e6b47             remove(v[i]);
2145c1     };
d41d8c
00ad54     int l = 1, r = 0;
a0a6ed     for (int i = 0; i < sz(q); i++)
f95b70     {
dc4c37         while (l > q[i].l)
47b649             consider(--l);
583e51         while (r < q[i].r)
04acf1             consider(++r);
4fdd74         while (l < q[i].l)
b33d23             consider(l++);
5dccbb         while (r > q[i].r)
b1b314             consider(r--);
d41d8c
4e8f49         if (q[i].lc != -1) // Remove LCA weight if necessary.
1dfb6d             consider(q[i].lc);
d41d8c
2d80c3         output(q[i].id);
d41d8c
4e8f49         if (q[i].lc != -1)

```

```
1dfb6d      consider(q[i].lc);
cbb184      }
cbb184      }
Full file hash: e7d7ff
```

5.7 M0s - Tree (Vertex Query)

```

a34983 #include "../..graph/lca/lca.cpp"
d41d8c
d41d8c /*
7418d3     Refer to the vertex queries option for more info.
aef741     This version is different since it is made to handle queries on
babd44     the cost of the edges in a path.
d41d8c
a7682c     To do that, transfer the cost to the vertex down in the rooted
db6efd     tree and use this.
d41d8c
89d138     Remember to use a valid value for the root (even if it never will
c00b42     be in a query).
08e268     Also remember some queries will be empty.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
fa0e65 struct query
f95b70 {
6ac794     int l, r, id, lc;
2145c1 };
d41d8c
044d82 template <class F>
23d154 void mos_tree(int root, int n, vector<pii> pq, vector<int> graph[],
      const F &add, const F &remove, const F &output)
f95b70 {
ba2a0c     int a, b;
32ee7e     lca_preprocess lca(root, n, graph);
d41d8c
ad35dc     vector<int> st(n + 1, 0), en(n + 1, 0), v(2 * n + 3, 0), cnt(n + 1,
      0), s;
04c09b     int id = 0;
45fa16     s.push_back(root);
365295     while (!s.empty())
f95b70     {
2ecd18         a = s.back();
342ca4         s.pop_back();
d41d8c
f4f8d2         if (st[a])
2e4e50             v[en[a] = ++id] = a;
2954e9         else
f95b70         {
bab970             v[st[a] = ++id] = a;
bcc44e             s.push_back(a);
1429ef             for (int i = 0; i < sz(graph[a]); i++)
9d7e46                 if (graph[a][i] != lca.p[0][a])
bbdabd                     s.push_back(graph[a][i]);
cbb184         }

```



```

cbb184     }
d41d8c
4007f6     vector<query> q;
d41d8c
1605f4     for (int i = 0; i < sz(pq); i++)
f95b70     {
5ca0a2         tie(a, b) = pq[i];
d41d8c
a917ad         if (st[a] > st[b])
2574c6             swap(a, b);
3f022f         int y = lca.lca(a, b);
84ab82         if (a == y)
173360             q.push_back({st[a], st[b], i, st[y]});
2954e9         else
e653b6             q.push_back({en[a], st[b], i, -1});
cbb184     }
d41d8c
7dd140     int bsize = 1 + (2 * n) / sqrt(sz(q));
c33690     sort(q.begin(), q.end(), [&](const query &lhs, const query &rhs) {
6b713e         if (lhs.l / bsize != rhs.l / bsize)
7124a1             return (lhs.l / bsize < rhs.l / bsize);
1e42a4         return lhs.r < rhs.r;
c0c97e     });
d41d8c
568e70     auto consider = [&](int i) {
e8ecf4         cnt[v[i]]++;
9308c6         if (cnt[v[i]] % 2 == 1)
9a7780             add(v[i]);
2954e9         else
8e6b47             remove(v[i]);
2145c1     };
d41d8c
00ad54     int l = 1, r = 0;
a0a6ed     for (int i = 0; i < sz(q); i++)
f95b70     {
dc4c37         while (l > q[i].l)
47b649             consider(--l);
583e51         while (r < q[i].r)
04acf1             consider(++r);
4fdd74         while (l < q[i].l)
b33d23             consider(l++);
5dccbb         while (r > q[i].r)
b1b314             consider(r--);
d41d8c
4e8f49         if (q[i].lc != -1) // Remove LCA weight if necessary.
1dfb6d             consider(q[i].lc);
d41d8c
2d80c3         output(q[i].id);
d41d8c
4e8f49         if (q[i].lc != -1)

```

```
1dfb6d      consider(q[i].lc);
cbb184      }
cbb184      }
Full file hash: e7d7ff
```

5.8 MOs - Hilbert

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
bd078f     MO's using Hilbert Curve to sort the queries.
91d19a     O(N * sqrt(Q) * max(insertion time, removal time)).
d41d8c
97c31d     Applicability:
2237d1     When Q is significantly less than N, it works much faster than
f49db1     the classical version.
d41d8c
b95cae     Usage:
02fbdb     Same as the classical version, but use the query's hilbertorder
6fc9ac     as comparator
c4c9bd */
d41d8c
547534 constexpr int logn = 20;
a314b2 constexpr int maxn = 1 << logn;
10e484 ll hilbertorder(int x, int y)
f95b70 {
b72b1c     ll d = 0;
cd0f95     for (int s = 1 << (logn - 1); s; s >>= 1)
f95b70     {
0e17cb         bool rx = x & s, ry = y & s;
9dc9be         d = d << 2 | rx * 3 ^ static_cast<int>(ry);
6b46cf         if (!ry)
f95b70         {
5f9e73             if (rx)
f95b70             {
e0b812                 x = maxn - x;
617448                 y = maxn - y;
cbb184             }
9dd20c             swap(x, y);
cbb184         }
cbb184     }
be245b     return d;
cbb184 }
d41d8c
806e2c struct query {
425e69     int l, r, id;
1e3c17     ll ord() const
f95b70     {
825ea3         return hilbertorder(l, r);
cbb184     }
2145c1 };
d41d8c
044d82 template<class F>
e5bd28 void mos(int n, vector<query> q, const F &add, const F &remove, const
      F &output)

```

```
f95b70 {
243fd1     int bsize = 1 + n / sqrt(sz(q));
c33690     sort(q.begin(), q.end(), [&](const query &lhs, const query &rhs) {
7dff82         return lhs.ord() < rhs.ord();
c0c97e     });
d41d8c
00ad54     int l = 1, r = 0; // int l = 0, r = -1; (if indices starts at 0)
a0a6ed     for (int i = 0; i < sz(q); i++)
f95b70     {
dc4c37         while (l > q[i].l)
194467             add(--l);
583e51         while (r < q[i].r)
3dcc02             add(++r);
4fdd74         while (l < q[i].l)
87e33e             remove(l++);
5dccbb         while (r > q[i].r)
8b6a11             remove(r--);
d41d8c
2d80c3         output(q[i].id);
cbb184     }
cbb184 }
Full file hash: 4195b8
```

5.9 Ternary Search (continuous)

```

d41d8c  /*
0a2f9f  Ternary Search:
550b14    Finds x such that f(x) is minimum in range [bot, top] in
387a9b    O(lg((top - bot) / eps)).
6f01ba    Value is correct within the specified precision eps.
d41d8c
ca2095  Constraints:
d370c2    f(x) is strictly decreasing for some interval [bot, x1],
28cacc    constant in an interval [x1, x2]
564c65    and strictly increasing in a interval [x2, top]. x1 <= x2 are
e4fd1f    arbitrary values where [x1, x2] is a plateau of optimal
86238d    solutions.
d41d8c
b95cae  Usage:
5b60e3    Call the function passing a lambda expression or function f.
33c10e    If there are multiple possible solutions, assume that an
662d1e    arbitrary one in the plateau is returned.
d41d8c
3db72f  Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
398727  template <typename F>
ca64a1  double ternary_search(const F &f, double bot = -1e9, double top = 1e9
    , double eps = 1e-9)
f95b70  {
14d91b    while (top - bot > eps)
f95b70    {
8e37d7        double x1 = (0.55*bot + 0.45*top); // (2*bot + top) / 3 is
d41d8c        // more stable, but slower.
3f8318        double x2 = (0.45*bot + 0.55*top);
9482ba        if (f(x1) > f(x2))
443914            bot = x1;
2954e9        else
16b1b8            top = x2;
cbb184    }
d41d8c
05eb81    return (bot + top) / 2;
cbb184  }
Full file hash: 082811

```

5.10 Ternary Search (discrete)

```

5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
0a2f9f Ternary Search:
5d5b1f Finds the smallest x in range [bot, top] such that f(x) is
792c4f maximum in O(lg(top - bot)).
d41d8c
ca2095 Constraints:
de7517 f(x) is strictly increasing for some interval [bot, x1],
ffacba constant in an interval [x1, x2] and strictly decreasing in a
bc602c interval [x2, top]. x1 <= x2 are arbitrary values where
a05aaa [x1, x2] is a plateau of optimal solutions.
d41d8c
b95cae Usage:
5b60e3 Call the function passing a lambda expression or function f.
d41d8c
3997db Source: modified from https://github.com/
6508de kth-competitive-programming/kactl/blob/master/content/
36db82 various/TernarySearch.h
c4c9bd */
d41d8c
398727 template <typename F>
2ba27e int ternary_search(const F &f, int bot, int top)
f95b70 {
03c8fe while (top - bot >= 5)
f95b70 {
5ada06 int mid = (bot + top) / 2;
2ef7fe if (f(mid) < f(mid + 1))
6a2c3c bot = mid;
2954e9 else
2b8e69 top = mid + 1;
cbb184 }
d41d8c
c6fbb5 for (int i = bot + 1; i <= top; i++)
773cb2 if (f(i) > f(bot))
8a7cc1 bot = i;
d41d8c
6ff46b return bot;
cbb184 }
Full file hash: 5b5ba5

```

6 Combinatorial

6.1 Binomial – Pascal’s Triangle

```

d41d8c
be64a6 #include "../.../contest/header.hpp"
d41d8c /*
8c1d32 [DESCRIPTION]
7988e1     Pre-computing all binomial coefficient (% MOD) up to
1ae475     C(MAXN, MAXN) into a matrix using pascal's triangle.
d41d8c
ff59a1 [COMPLEXITY]
544b30     O(n^2) to Pre-computing
6d7bb6     O(1) to lookup
c4c9bd */
d41d8c
fd83cb #define MAXN 3123
39b5cf #define MOD 1000000007
f93f90 long long C[MAXN][MAXN];
d41d8c
51e552 void init_ncr()
f95b70 {
70057c     C[0][0] = 1;
a04712     for (int n = 1; n < MAXN; ++n) {
608658         C[n][0] = C[n][n] = 1;
fe813a         for (int k = 1; k < n; ++k)
06438c             C[n][k] = (C[n - 1][k - 1] + C[n - 1][k]) % MOD;
cbb184     }
cbb184 }
Full file hash: 88d086

```

6.2 Binomial - Lucas' Theorem

```

be64a6 #include "../.../contest/header.hpp"
b92926 #include "../.../number_theory/mod_inverse/mod_inverse.cpp"
d41d8c /*
8c1d32 [DESCRIPTION]
392786 Lucas' theorem to calculate  $C(N, M) \% P$  where  $N, M$  be
4921be non-negative integers and  $P$  a prime.
d41d8c
9af554 Write  $N$  and  $M$  in the base  $P$ :
102292  $N_p = n_{kp}^k + \dots + n_{1p} + n_0$  and  $M_p = m_{kp}^k + \dots + m_{1p} + m_0$ .
0e88ed Then  $C(N, M) == \text{Prod}(C(n_i, m_i)) \% P$ 
d41d8c
470e5b [CONSEQUENCE]
da479c A binomial coefficient  $C(N, M)$  is divisible by a prime  $P$  iff
fea373 there is an index  $i$  where
4cfd1d  $N_p[i] < M_p[i]$  which leads to  $C(N_p[i], M_p[i]) = 0$ .
14da71 Hence,  $C(N, M) \% P = 0$ 
d41d8c
bd7472 [USAGE]
ed7050 Pre-compute all factorials (mod  $P$ ) up to the  $P$  prime chosen.
904b18 all the function choseModP.
9820f1 You can also pre-compute all factorials' modular inverses to
fd6c3e boost the performance.
d41d8c
ff59a1 [COMPLEXITY]
c4de22  $O(\log_p(N) * \text{mod\_inverse}())$ 
c4c9bd */
d41d8c
2f5a93 ll chooseModP(ll n, ll m, int p, vector<ll> &fact)
f95b70 {
386747 ll c = 1;
bce30e while (n || m)
f95b70 {
5ec235 ll a = n % p, b = m % p;
545267 if (a < b)
bb30ba return 0;
9f365c c = c * fact[a] % p * mod_inverse<ll>(fact[b], p) % p *
mod_inverse<ll>(fact[a - b], p) % p;
f4a0f7 n /= p;
b499ca m /= p;
cbb184 }
807fb1 return c;
Full file hash: b91f81

```


6.3 Grundy

```

d41d8c
5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
e1025f     Mex for Grundy Number O(N):
aaf075     To calculate the Grundy Number for a set of states, first
7603df     set terminal states' Grundy Number (zero if no move is a
da287c     loss condition).
39203d     Then, for each state, find the MEX for the Grundy Numbers of
7b06af     reachable states (i.e. the lowest Grundy Number not
58630c     present).
d1d683     This will be the current state's Grundy Number.
d9551a     If you have many parallel games, you can find the equivalent
54f13f     Grundy number by doing XOR of individual Grundy Numbers.
777fcf     0 equals losing position, any other value is a winning
c7295a     position.
d41d8c
ca2095     Constraints:
c91487     The game must be symmetrical (same moves are available for
4b3a24     both players); have perfect information (no hidden or
838e2e     random stuff);
04a577     be finite (no loops).
d41d8c
b95cae     Usage:
90c9f6     For each state, save the Grundy Number of all reachable
6be7b6     states in a vector v, and pass as argument to mex().
d41d8c
d41d8c
2c0da0     Source: my head
c4c9bd */
d41d8c
b7803b int mex(vector<int> v){
d41d8c     //Place every value in the position with same index (if possible
d41d8c     //and it's not already there)
f14e7f     for(int i = 0; i < v.size(); i++){
bd35f8         while(v[i] < v.size() && v[v[i]] != v[i])
dda303             swap(v[v[i]], v[i]);
cbb184     }
d41d8c     //Verify the first missing number
c6ecd5     for(int i = 0; i < v.size(); i++)
fce28f         if(v[i] != i)
d9a594             return i;
d41d8c
5f47f2     return v.size();
cbb184 }
Full file hash: 6fe471

```

6.4 Surreal Numbers

```

5d1131 #include "../..contest/header.hpp"
d41d8c
d41d8c /*
3fb33e     General Theory for Two Player Game
d41d8c
431282     Take a perfect information game involving two players,
5d55cf     Left and Right, where either one can start the game.
d41d8c
1dd59e     This is a partial game in which the allowable moves
5d0863     depend on which of the two players is currently moving
6b0846     (e.g chess).
d41d8c
c0d51d     There are four possible scenarios for an initial configuration:
81bc0d         - Left wins (does not matter if first or second).
494fff         - Right wins
f369c0         - First player wins.
4840f6         - Second player wins
d41d8c
305ba4     If a game has no "First player wins" configurations,
944570     the configurations of the game can be mapped to real
295eb3     numbers  $s(i)$  of the form  $a/2^b$  such that.
585476         -  $s(i) > 0 \rightarrow$  Left wins
fc513a         -  $s(i) < 0 \rightarrow$  Right wins
b108f3         -  $s(i) = 0 \rightarrow$  second player wins
d41d8c
450004     The union of two states  $i, j$  is mapped to  $s(i) + s(j)$ 
d41d8c
2221d9     More formally:
d62d80     A game is a position in a contest between two players, Left and
f2a2de     Right.
50071c     Each player has a set of games called options to choose from in
b9d920     turn.
2b0956     Games are written  $\{L|R\}$  where  $L$  is the set of Left's options and
cf5bd1      $R$  is the set of Right's options.
d41d8c
532c5f     At the start there are no games at all, so the empty set is the
6a6278     only set of options we can provide to the players.
1fae7d     This defines the game  $\{\}$ , which is called 0. We consider a
2121f8     player who must play a turn but has no options to have lost the
8e400a     game (so in game 0 second player wins). Given this game 0 there
59fbc4     are now two possible sets of options, the empty set and the set
c31906     whose only element is zero.
e06980     The game  $\{0|\}$  is called 1, and the game  $\{|\}$  is called -1.
d6fd85     In game 1, if Right goes first, Left wins. And if Left goes
765ee8     first, he will choose game 0 to be next and win (because Right
4c256d     will have no moves).
1cde03     So game 1 is of the type Left wins, as expected.
6e4cbe     The game  $\{0|0\}$  is called  $*$  (star), and is the first game we find

```

```

d3107a     that is not a number (in this game, first player wins).
d41d8c
422b87     All numbers are positive, negative, or zero, and we say that a
079aff     game is positive if Left will win, negative if Right will win,
53c31a     or zero if the second player will win. Games that are not numbers
bde09b     have a fourth possibility: they may be fuzzy, meaning that the
4f5c39     first player will win. * is a fuzzy game.[4]
c4c9bd */
d41d8c
13a4b1     int main(void)
f95b70     {
b3f86c         cin.sync_with_stdio(0);
b95c6c         cin.tie(0);
1a88fd         int n;
ba5ec5         while (cin >> n)
f95b70         {
55c786             ll mult = (1ll << 40);
d41d8c             // This will store the surreal number for each game times
d41d8c             // 2^40 (just to avoid doubles).
75f676             vector<ll> s(n);
fb8240             vector<int> val(n);
d41d8c
83008c             for (int i = 0; i < n; i++)
f95b70             {
905799                 string a;
964d25                 cin >> a;
e4c311                 ll x = mult;
0f6f6d                 bool change = false;
784833                 for (int j = 0; j < sz(a); j++)
f95b70                 {
194160                     if (a[j] != a[0]) // After first different, start
d41d8c                         // changing x.
981b92                     change = true;
d41d8c
b92c76                     if (change)
4feb2a                         x /= 2;
d41d8c
525d22                     if (a[j] == 'B')
91055a                         s[i] += x;
2954e9                     else
6b1445                         s[i] -= x;
cbb184                 }
d41d8c
7e757f                 val[i] = sz(a);
cbb184             }
d41d8c
d41d8c             // Now we have s[i] for each game.
d41d8c             // If we join two games i,j we get a game x with
d41d8c             // s(x) = s[i] + s[j] and val(x) = val[i] + val[j].
d41d8c             // So to find a fair game with s[x] = 0 and maximum val

```

```

d41d8c // We need to find a subset with zero sum and maximum val.
d41d8c
d41d8c // Here onwards we just solve this problem with meet in
d41d8c // the middle.
d41d8c
60d900 unordered_map<ll, int> tab;
904add int mid = (n + 1) / 2;
d096c6 for (int i = 0; i < (1 << mid); i++)
f95b70 {
00d744     ll x = 0;
39d9dc     int y = 0;
7a6f84     for (int j = 0; j < mid; j++)
8aa127         if (i & (1 << j))
f95b70             {
43ea17                 x += s[j];
ab019b                 y += val[j];
cbb184             }
d41d8c
0f0323     tab[x] = max(tab[x], y);
cbb184 }
d41d8c
1a4d4a int ans = 0;
d41d8c
611f57 for (int i = 0; i < (1 << (n - mid)); i++)
f95b70 {
00d744     ll x = 0;
39d9dc     int y = 0;
6ccdb0     for (int j = 0; j < (n - mid); j++)
8aa127         if (i & (1 << j))
f95b70             {
b7e638                 x += s[mid + j];
7d4407                 y += val[mid + j];
cbb184             }
d41d8c
f85787     auto it = tab.find(-x);
e1c80c     if (it != tab.end())
f95b70     {
fb7de6         ans = max(ans, y + it->second);
cbb184     }
cbb184 }
d41d8c
886648     cout << ans << endl;
cbb184 }
cbb184 }
Full file hash: 767d32

```

7 Number Theory

7.1 General Chinese Remainder Theorem

```

5d1131 #include "../contest/header.hpp"
771bdd #include "../euclid/euclid.cpp"
d41d8c
d41d8c /*
8c1d32     [DESCRIPTION]
055dbf     Returns a number  $x \% \text{lcm}(m,n)$ , such that:
096aa3          $x \equiv a \pmod{m}$ 
fdeb15          $x \equiv b \pmod{n}$ 
d41d8c
9c7922     returns -1 if there is no solution
d41d8c
ff59a1     [COMPLEXITY]
9a331b          $\log(n)$ 
d41d8c
c1fa90     [CONSTRAINTS]
832081         LCM(m, n) should fit in a long long variable.
c4c9bd */
d41d8c
5d15e5 ll crt(ll a, ll m, ll b, ll n)
f95b70 {
e6a14b     if (n > m)
134a9d         swap(a, b), swap(m, n);
3e2193     ll x, y, g = gcd<ll>(m, n, x, y);
f08934     if ((a - b) % g != 0)
daa4d1         return -1;
ef8b1c     x = (b - a) % n * x % n / g * m + a;
16c358     return x < 0 ? x + m * n / g : x;
cbb184 }
Full file hash: 261c61

```

7.2 General Chinese Remainder Theorem – System

```

5d1131 #include "../contest/header.hpp"
5c6423 #include "crt.cpp"
d41d8c /*
8c1d32     [DESCRIPTION]
4c169b     Returns a integer a number x, such that:
773426         x === a[0] (mod m[0])
726304         x === a[1] (mod m[1])
2f43b4         ...
f8e20b         x === a[n - 1] (mod m[n - 1])
d41d8c
06b9e3     The m[] set does not need to be only of coprimes, because it uses
e630ba     the generalized version of CRT.
d41d8c
bd7472     [USAGE]
8a1016     Just pass the arrays as shown above in the description and their
90261b     size.
deb107     It's 0-indexed, but its trivial to change it to 1-indexed.
d41d8c
8a9322     [RESULT]
33638d     The function returns x % LCM(m[0], m[1], ..., m[n - 1]) if a
2d9b26     answer exists. Otherwise it returns -1.
d41d8c
ff59a1     [COMPLEXITY]
a28f97     O( n * log(LCM(m)) )
d41d8c
c1fa90     [CONSTRAINTS]
6a09ab         LCM(m[0], m[1], ..., m[n - 1]) should fit in a long long
65b604         variable.
e2e167         The values of a[] can be arbitrary, because they are
806706         normalized inside the function
d41d8c
f55d9e     source: https://codeforces.com/blog/entry/61290
d41d8c
c4c9bd */
d41d8c
815f3a ll crt_system(ll a[], ll m[], int n)
f95b70 {
d41d8c     // normalize
83008c     for (int i = 0; i < n; i++)
e05f2f         a[i] = (a[i] % m[i] + m[i]) % m[i];
d41d8c
a78b68     ll ans = a[0];
20b4de     ll lcm = m[0];
aa4866     for (int i = 1; i < n; i++)
f95b70     {
1732b5         ans = crt(ans, lcm, a[i], m[i]);
7872a9         if (ans == -1)
daa4d1             return -1;

```

```
0be4e5      ll x, y;  
e2e58b      ll d = gcd<ll>(lcm, m[i], x, y);  
930f87      lcm = lcm * m[i] / d;  
cbb184      }  
ba75d2      return ans;  
Full file hash: dd8682
```

7.3 Euclid

```

5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
4b5b70     Extended Euclidean Algorithm:
71fa74     Returns the gcd of a and b.
49d10b     Also finds numbers x and y for which  $a * x + b * y = \text{gcd}(a, b)$ 
f41d43     (not unique).
cd0ca0     All pairs can be represented in the form
c42c4e      $(x + k * b / \text{gcd}, y - k * a / \text{gcd})$ 
76eea3     for k an arbitrary integer.
829635     If there are several such x and y, the function returns the
e3988a     pair for which  $|x| + |y|$  is minimal.
232b60     If there are several x and y satisfying the minimal criteria,
2eda4e     it outputs the pair for which  $X \leq Y$ .
d41d8c
3997db     Source: modified from https://cp-algorithms.com/algebra/
df5dd3     extended-euclid-algorithm.html
d41d8c
b95cae     Usage:
6475da     For non-extendend version, c++ has __gcd and __lcm.
d41d8c
ca2095     Constraints:
30a9e9     Produces correct results for negative integers as well.
c4c9bd */
d41d8c
4fce64 template<class T>
94606e T gcd(T a, T b, T &x, T &y)
f95b70 {
fcbb63     if (b == 0)
f95b70     {
483406         x = 1;
01dbf4         y = 0;
3f5343         return a;
cbb184     }
d41d8c
32895f     T x1, y1;
254183     T d = gcd(b, a % b, x1, y1);
711e33     x = y1;
a2a46d     y = x1 - y1 * (a / b);
be245b     return d;
cbb184 }
Full file hash: 0c35ae

```


7.4 Factorization (Pollard rho)

```

d41d8c  /*
d0042a  Description:
b75bc0  Pollard-rho randomized factorization algorithm. Returns prime
9374f8  factors of a number, in arbitrary order (e.g. 2299 ->
fbf488  {11, 19, 11}).
d41d8c
421b12  Time:
018897   $O(n^{1/4})$  gcd calls, less for numbers with small factors.
d41d8c
1d1558  Source: https://github.com/kth-competitive-programming/
c4c9bd  */
d41d8c
5d1131  #include "../contest/header.hpp"
09d4b3  #include "../primality_test/millerRabin.cpp"
d41d8c
97d675  ull pollard(ull n)
f95b70  {
4de5da      auto f = [n](ull x) { return (mod_mul(x, x, n) + 1) % n; };
6b3881      if (!(n & 1))
18b932          return 2;
8c364a      for (ull i = 2;; i++)
f95b70      {
e17462          ull x = i, y = f(x), p;
332fe8          while ((p = __gcd(n + y - x, n)) == 1)
b789c2              x = f(x), y = f(f(y));
94037d          if (p != n)
74e469              return p;
cbb184      }
cbb184  }
d41d8c
2de7b2  vector<ull> factorize(ull n)
f95b70  {
e7f697      if (n == 1)
21d05e          return {};
121b21      if (isPrime(n))
48e372          return {n};
bc6125      ull x = pollard(n);
b3b29a      auto l = factorize(x), r = factorize(n / x);
7af87c      l.insert(l.end(), all(r));
792fd4      return l;
cbb184  }
Full file hash: 66d5a6

```

7.5 Modular Inverse

```

771bdd #include "../euclid/euclid.cpp"
d41d8c
d41d8c /*
18a91e     Modular Inverse:
76e032         Returns an integer x such that (a * x) % m == 1.
71575f         The modular inverse exists if and only if a and m are
8dc50f         relatively prime.
ff1c03         Modular inverse is also equal to a^(phi(m) - 1) % m.
eb3b67         In particular, if m is prime a^(-1) == a^(m-2), which might be
6bc420         faster to code.
d41d8c
3997db     Source: modified from https://cp-algorithms.com/algebra/
f41770         module-inverse.html
c4c9bd */
d41d8c
4fce64 template<class T>
b267c1 T mod_inverse(T a, T m)
f95b70 {
645c5d     T x, y;
6553c1     assert(gcd(a, m, x, y) == 1); // Or return something, if gcd is
d41d8c         // not 1 the inverse doesn't exist.
08ffd4     return (x % m + m) % m;
cbb184 }
Full file hash: 7efa11

```

7.6 Large modular mult/pow

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
d0042a     Description:
b31056     Calculate  $a * b \bmod c$  (or  $a^b \bmod c$ )
002948     for  $0 \leq a, b < c < 2^{63}$ .
421b12     Time:
666f83     mod_mul: 0 (1)
19c3bc     mod_pow: (log b)
1d1558     Source: https://github.com/kth-competitive-programming/
c4c9bd */
d41d8c
1199bf ull mod_mul(ull a, ull b, ull M)
f95b70 {
053258     ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
964402     return ret + M * (ret < 0) - M * (ret >= (ll)M);
cbb184 }
b40e0d ull mod_pow(ull b, ull e, ull mod)
f95b70 {
c1a4a1     ull ans = 1;
4d1884     for (; e; b = mod_mul(b, b, mod), e /= 2)
654342         if (e & 1)
69ff2e             ans = mod_mul(ans, b, mod);
ba75d2     return ans;
Full file hash: ebdfdb

```

7.7 Modular Arithmetic

```

90fb23 #include "../mod_inverse/mod_inverse.cpp"
d41d8c
d41d8c /*
d0cd85     Modular Arithmetic:
7bc537     Struct wrapper on to of modular arithmetics.
d41d8c
3997db     Source: modified from https://github.com/
6508de     kth-competitive-programming/kactl/blob/master/content/
591dff     number-theory/ModularArithmetic.h
c4c9bd */
d41d8c
31e95d template <ll mod>
072773 struct mod_num
f95b70 {
4ad8b8     ll x;
de85bc     explicit mod_num(ll x = 0) : x(x % mod) {}
f764be     mod_num operator+(mod_num b) { return mod_num(x + b.x); }
b1d9a9     mod_num operator-(mod_num b) { return mod_num(x - b.x + mod); }
b2fd70     mod_num operator*(mod_num b) { return mod_num(x * b.x); }
09dd48     mod_num operator/(mod_num b) { return mod_num(x * mod_inverse(b.x,
    mod)); }
583822     mod_num operator^(ll e)
f95b70     {
972df3         mod_num ans(1);
6d7204         mod_num b = *this;
25d98a         for (; e; b = b * b, e /= 2)
654342             if (e & 1)
bfb0b         ans = ans * b;
ba75d2         return ans;
cbb184     }
d41d8c
6dcc99     void operator+=(mod_num b) { x = (x + b.x) % mod; }
2145c1 };
d41d8c
31e95d template <ll mod>
58e3fb ostream &operator<<(ostream &os, mod_num<mod> x)
f95b70 {
55e0cb     return os << x.x;
cbb184 }
Full file hash: f151a0

```

7.8 Phi

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
bf26c1 Euler's totient function (PHI):
fab9b5 Euler's totient function, also known as phi-function PHI(n),
f44cb7 counts the number of integers between 1 and n inclusive, which
8d19a8 are coprime to n. Two numbers are coprime if their greatest
d6ae51 common divisor equals 1 (1 is considered to be coprime to any
83f472 number).
d41d8c
d41d8c
3997db Source: modified from https://cp-algorithms.com/algebra/
356259 phi-function.html
e1fde7 and https://github.com/kth-competitive-programming/kactl/blob/
a57945 master/content/number-theory/phiFunction.h
d41d8c
b95cae Usage:
a1b248 Some useful properties:
9f5d84 - If p is a prime number,  $\text{PHI}(p)=p-1$ .
d4d311 - If a and b are relatively prime,  $\text{PHI}(ab)=\text{PHI}(a)*\text{PHI}(b)$ .
343fa6 - In general, for not coprime a and b,
0456bd  $\text{PHI}(ab)=\text{PHI}(a)*\text{PHI}(b)*d/\text{PHI}(d)$ , with  $d=\text{gcd}(a,b)$  holds.
417c3d -  $\text{PHI}(\text{PHI}(m)) \leq m / 2$ 
25b5a9 - Euler's theorem:  $a^{\text{PHI}(m)} \equiv 1 \pmod{m}$ , for a and m coprime
bffb1c - For a and m coprime:  $a^n \equiv a^{(n \% \text{PHI}(m))} \pmod{m}$ 
27900f - For arbitrary x,m and  $n \geq \log_2(m)$ :
131b8b  $x^n \equiv x^{(\text{PHI}(m)+[n \% \text{PHI}(m)])} \pmod{m}$ 
e1e6a8 The one above allows computing modular exponentiation for
6e5ead really large exponents.
565281 - If d is a divisor of n, then there are  $\phi(n/d)$  numbers
83746a  $i \leq n$  for which  $\text{gcd}(i,n)=d$ 
137411 -  $\sum_{d|n} \phi(d) = n$ 
c228b3 -  $\sum_{1 \leq k \leq n, \text{gcd}(k,n)=1} k = n * \phi(n) / 2$ , for  $n > 1$ 
c4c9bd */
d41d8c
d41d8c // Use this one for few values of phi.
b5f6f9 int phi(int n)
f95b70 {
efa47a int result = n;
83f497 for (int i = 2; i * i <= n; i++)
f95b70 {
775f6d if (n % i == 0)
f95b70 {
49edb8 while (n % i == 0)
1358bf n /= i;
21cd49 result -= result / i;
cbb184 }
cbb184 }

```

```
f3d362     if (n > 1)
e48781         result -= result / n;
dc8384     return result;
cbb184 }
d41d8c
4fee4d namespace totient
f95b70 {
2d637a     const int MAXV = 1000001; // Takes ~0.03 s for 10^6.
6e559b     int phi[MAXV];
d41d8c
b2a56e     void init()
f95b70     {
9484bb         for (int i = 0; i < MAXV; i++)
ed1f90             phi[i] = i & 1 ? i : i / 2;
9be7d6         for (int i = 3; i < MAXV; i += 2)
a2252f             if (phi[i] == i)
8a4437                 for (int j = i; j < MAXV; j += i)
a9ba36                     phi[j] -= phi[j] / i;
cbb184     }
cbb184 } // namespace totient
Full file hash: e79764
```

7.9 Primality Test

```

d41d8c  /*
d0042a      Description:
37dcd5          Deterministic Miller-Rabin primality test.
334ea2          Guaranteed to work for numbers up to 2^64 (for larger
824624          numbers, extend A randomly).
d41d8c
421b12      Time:
6718e3          7 * O(log b)
d41d8c
1d1558      Source: https://github.com/kth-competitive-programming/
c4c9bd  */
d41d8c
5d1131  #include "../contest/header.hpp"
cfa7c0  #include "../mod_mul/mod_mul.cpp"
d41d8c
91e805  bool isPrime(ull n)
f95b70  {
e001bf      if (n < 2 || n % 6 % 4 != 1)
d15826          return n - 2 < 2;
43a246      ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
c17dd6          s = __builtin_ctzll(n - 1), d = n >> s;
56ef5c      for (auto &a : A)
f95b70      { // ^ count trailing zeroes
8a86e5          ull p = mod_pow(a, d, n), i = s;
274cbc          while (p != 1 && p != n - 1 && a % n && i--)
2cbb80              p = mod_mul(p, p, n);
5fdfe0          if (p != n - 1 && i != s)
bb30ba              return 0;
cbb184      }
6a5530      return 1;
cbb184  }
Full file hash: df75c7

```

7.10 Sieve

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
d5cfbe     Sieve of Eratosthenes:
0a5343         Finds all primes in interval [2, MAXP] in O(MAXP) time.
91376b         Also finds lp[i] for every i in [2, MAXP], such that lp[i] is
0aab64         the minimum prime factor of i.
6dbf8e         Particularly useful for factorization.
d41d8c
3997db     Source: modified from https://cp-algorithms.com/algebra/
874189         prime-sieve-linear.html
d41d8c
b95cae     Usage:
9290fb         Set MAXP and call init.
85265b         Sieve for 10^7 should run in about 0.2 s.
c4c9bd */
d41d8c
2ca8b0 namespace sieve
f95b70 {
bace98     const int MAXP = 10000000; // Will find primes in interval [2, MAXP].
39b809     int lp[MAXP + 1]; // lp[i] is the minimum prime factor of i.
632f82     vector<int> p; // Ordered list of primes up to MAXP.
d41d8c
b2a56e     void init()
f95b70     {
008cd3         for (int i = 2; i <= MAXP; i++)
f95b70         {
d4a1cc             if (lp[i] == 0)
b6fab7                 p.push_back(lp[i] = i);
d41d8c
9d854a             for (int j = 0; j < (int)p.size() && p[j] <= lp[i] && i * p[j] <=
MAXP; j++)
fb7d48                 lp[i * p[j]] = p[j];
cbb184         }
cbb184     }
cbb184 } // namespace sieve
Full file hash: e9076d

```


8 Numerical

8.1 Big Int

```

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

```

```

ccf902     BigInt(long long v) {
324222         *this = v;
cbb184     }
235125     BigInt& operator = (long long v) {
ce6fc2         sign = 1;
ea2149         if (v < 0) {
6a74a9             sign = -1;
6fab41             v = -v;
cbb184         }
22838a         a.clear();
fefe2d         for (; v > 0; v = v / BASE)
c237f1             a.push_back(v % BASE);
357a55         return *this;
cbb184     }

d41d8c     // Initialize from string.
d41d8c     BigInt(const string& s) {
c710ec         read(s);
cbb184     }

d41d8c     // ----- Input / Output -----
d41d8c     void read(const string& s) {
6c30c4         sign = 1;
ce6fc2         a.clear();
22838a         int pos = 0;
bec7a6         while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+'
a68fdf             ')) {
dbe226             if (s[pos] == '-')
2b8bd1                 sign = -sign;
17dad0             ++pos;
cbb184         }
7959ef         for (int i = s.size() - 1; i >= pos; i -= BASE_DIGITS) {
c67d6f             int x = 0;
d343c4             for (int j = max(pos, i - BASE_DIGITS + 1); j <= i; j++)
cfc7e4                 x = x * 10 + s[j] - '0';
7c6978             a.push_back(x);
cbb184         }
0ebb65         trim();
cbb184     }

bd2995     friend istream& operator>>(istream &stream, BigInt &v) {
ac0066         string s;
e0c759         stream >> s;
c4002a         v.read(s);
a87cf7         return stream;
cbb184     }

d41d8c     friend ostream& operator<<(ostream &stream, const BigInt &v) {
44647f         if (v.sign == -1 && !v.isZero())
b5c525             stream << '-';
27bc2a         stream << (v.a.empty() ? 0 : v.a.back());
4fda68

```

```

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

```

```

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

```

```

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

```

```

d41d8c
d41d8c // ----- Operators * / % -----
a179f4 friend pair<BigInt, BigInt> divmod(const BigInt& a1, const BigInt
& b1) {
872d46     assert(b1 > 0); // divmod not well-defined for b < 0.
d41d8c
25f4e9     long long norm = BASE / (b1.a.back() + 1);
7c41dc     BigInt a = a1.abs() * norm;
ecd4f4     BigInt b = b1.abs() * norm;
da5ddc     BigInt q = 0, r = 0;
90ee93     q.a.resize(a.a.size());
d41d8c
72b5b8     for (int i = a.a.size() - 1; i >= 0; i--) {
79aca3         r *= BASE;
0caac0         r += a.a[i];
0eeb4e         long long s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.
size()];
bcl1a99         long long s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a
.size() - 1];
0ebba0         long long d = ((long long) BASE * s1 + s2) / b.a.back();
5d4f85         r -= b * d;
612239         while (r < 0) {
bd3902             r += b, --d;
cbb184         }
5898c8         q.a[i] = d;
cbb184     }
d41d8c
535024     q.sign = a1.sign * b1.sign;
a29af3     r.sign = a1.sign;
36a918     q.trim();
9a35fd     r.trim();
38a539     auto res = make_pair(q, r / norm);
458098     if (res.second < 0) res.second += b1;
b5053e     return res;
cbb184 }
547e4b BigInt operator/(const BigInt &v) const {
ce8f7c     return divmod(*this, v).first;
cbb184 }
d41d8c
ee46c3 BigInt operator%(const BigInt &v) const {
7a671a     return divmod(*this, v).second;
cbb184 }
d41d8c
c2998e void operator/=(int v) {
d1ee66     assert(v > 0); // operator / not well-defined for v <= 0.
dd9f94     if (llabs(v) >= BASE) {
85cc00         *this /= BigInt(v);
505b97         return ;
cbb184     }
8e679f     if (v < 0)

```

```

20198f         sign = -sign, v = -v;
8e5533     for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
cbe153         long long cur = a[i] + rem * (long long) BASE;
8d1e71         a[i] = (int) (cur / v);
cb35e0         rem = (int) (cur % v);
cbb184     }
0ebb65     trim();
cbb184 }

d41d8c
49658a     BigInt operator/(int v) const {
d1ee66         assert(v > 0); // operator / not well-defined for v <= 0.
d41d8c
dd9f94         if (llabs(v) >= BASE) {
ed0225             return *this / BigInt(v);
cbb184         }
18bf1f         BigInt res = *this;
37184f         res /= v;
b5053e         return res;
cbb184     }
3b4fa6     void operator/=(const BigInt &v) {
e51f70         *this = *this / v;
cbb184     }
d41d8c

54c35d     long long operator%(long long v) const {
d1ee66         assert(v > 0); // operator / not well-defined for v <= 0.
a1e888         assert(v < BASE);
cbcd95         int m = 0;
947442         for (int i = a.size() - 1; i >= 0; --i)
95269a             m = (a[i] + m * (long long) BASE) % v;
9af577         return m * sign;
cbb184     }
d41d8c

a0b62a     void operator*=(int v) {
dd9f94         if (llabs(v) >= BASE) {
014cdd             *this *= BigInt(v);
505b97             return ;
cbb184         }
8e679f         if (v < 0)
20198f             sign = -sign, v = -v;
c6279c         for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i)
        {
74ab7d             if (i == (int) a.size())
ddfb75                 a.push_back(0);
d09f08             long long cur = a[i] * (long long) v + carry;
98cd39             carry = (int) (cur / BASE);
861843             a[i] = (int) (cur % BASE);
d41d8c             //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur),
            "c"(base));
d41d8c             /*
97f03f             int val;

```

```

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

```



```

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

```

```

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

```

```

6a2f29      vll r = karatsubaMultiply(a2, b2);
be9bd2      for (int i = 0; i < (int) a1b1.size(); i++)
47fef2          r[i] -= a1b1[i];
cf04ec      for (int i = 0; i < (int) a2b2.size(); i++)
00a00c          r[i] -= a2b2[i];
d41d8c
5951a9      for (int i = 0; i < (int) r.size(); i++)
1bf61e          res[i + k] += r[i];
be9bd2      for (int i = 0; i < (int) a1b1.size(); i++)
d6cf88          res[i] += a1b1[i];
cf04ec      for (int i = 0; i < (int) a2b2.size(); i++)
ab9916          res[i + n] += a2b2[i];
b5053e      return res;
cbb184    }
d41d8c
287510    BigInt mul_karatsuba(const BigInt &v) const {
48c647        vector<int> a6 = convert_base(this->a, BASE_DIGITS, 6);
f64a05        vector<int> b6 = convert_base(v.a, BASE_DIGITS, 6);
e1cb30        vll a(a6.begin(), a6.end());
5ed74f        vll b(b6.begin(), b6.end());
1a813e        while (a.size() < b.size())
ddf7b5            a.push_back(0);
0d118e        while (b.size() < a.size())
c40831            b.push_back(0);
634b60        while (a.size() & (a.size() - 1))
eed3fb            a.push_back(0), b.push_back(0);
16bf35        vll c = karatsubaMultiply(a, b);
02a624        BigInt res;
325cfe        res.sign = sign * v.sign;
6e20af        long long carry = 0;
7dbc9f        for (int i = 0; i < (int) c.size(); i++) {
dc97b8            long long cur = c[i] + carry;
cdf472            res.a.push_back((int) (cur % 1000000));
735fb2            carry = cur / 1000000;
cbb184        }
7b10c4        res.a = convert_base(res.a, 6, BASE_DIGITS);
d7ee6d        res.trim();
b5053e        return res;
cbb184    }
d41d8c
933d02    void operator*=(const BigInt &v) {
fa4bc1        *this = *this * v;
cbb184    }
24478f    BigInt operator*(const BigInt &v) const {
de6792        if (a.size() * v.a.size() <= 1000111) return mul_simple(v);
fec548        if (a.size() > 500111 || v.a.size() > 500111) return mul_fft(
    v);
a67c32        return mul_karatsuba(v);
cbb184    }
d41d8c

```

```

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

```

```

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

```

8.2 FFT

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
27ada6     FFT:
f7b296     FFT allows multiplication of two polynomials in  $O(n \log n)$ .
420c7a     This can also be used to multiply two long numbers faster.
c00ff6     Other applications:
c35b73     - All possible sums of two arrays.
1da5a4     - Dot product of vector a with every cyclic shift of vector b.
3807b3     - Attaching two boolean stripes without two 1s next to each
b26527     other.
52f6a3     - String matching.
d41d8c
b95cae     Usage:
178aa5     long double is a lot slower. 3s with ld and 0.7 with double
93dc1c     for  $10^6$  size vectors.
d41d8c
1d1558     Source: https://cp-algorithms.com/algebra/fft.html
c4c9bd */
d41d8c
99bb89 using cd = complex<ld>;
c4f8de const ld PI = acos(-1.0L);
d41d8c
9b5b94 void fft(vector<cd> &a, bool invert)
f95b70 {
6c3f33     int n = sz(a);
d41d8c
d94885     for (int i = 1, j = 0; i < n; i++)
f95b70     {
4af5d7         int bit = n >> 1;
474fac         for (; j & bit; bit >>= 1)
53c7ca             j ^= bit;
53c7ca         j ^= bit;
d41d8c
9dcc5c         if (i < j)
33275d             swap(a[i], a[j]);
cbb184     }
d41d8c
2fe9ad     for (int len = 2; len <= n; len <= 1)
f95b70     {
c19c97         ld ang = 2 * PI / len * (invert ? -1 : 1);
808a0b         cd wlen(cos(ang), sin(ang));
3dd9d3         for (int i = 0; i < n; i += len)
f95b70         {
8c3c80             cd w(1);
5594fb             for (int j = 0; j < len / 2; j++)
f95b70             {
cf0824                 cd u = a[i + j], v = a[i + j + len / 2] * w;

```

```

6c3014         a[i + j] = u + v;
273255         a[i + j + len / 2] = u - v;
3e4104         w *= wlen;
cbb184     }
cbb184     }
cbb184 }
d41d8c
2111a0     if (invert)
f95b70     {
0b5665         for (cd &x : a)
b6d31b             x /= n;
cbb184     }
cbb184 }
d41d8c
d41d8c // Input a[0] + a[1]x + a[2]x^2 ...
d41d8c // Returns polynomial of size equal to the smallest power of two at
d41d8c // least as large as a.size() + b.size(). This can have some extra
d41d8c // zeros.
d41d8c // Use long double if using long long.
4fce64 template <class T>
a3a2ed vector<T> multiply(vector<T> const &a, vector<T> const &b)
f95b70 {
6fa6b9     vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
43ec81     int n = 1;
86a505     while (n < sz(a) + sz(b))
c149a4         n <<= 1;
37aa6c     fa.resize(n);
870070     fb.resize(n);
d41d8c
3a13f2     fft(fa, false);
c76760     fft(fb, false);
83008c     for (int i = 0; i < n; i++)
940eb7         fa[i] *= fb[i];
959d01     fft(fa, true);
d41d8c
ebf3b6     vector<T> result(n);
83008c     for (int i = 0; i < n; i++)
5b32dc         result[i] = (T)round(fa[i].real()); // Remember to remove
d41d8c                                         // rounding if working
d41d8c                                         // with floats.
dc8384     return result;
cbb184 }
Full file hash: d7b825

```

8.3 Fraction

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
390211     Fraction representation:
4d1181         All operations run in  $O(\gcd) = O(\log)$ .
d41d8c
b95cae     Usage:
70e7d7         Don't modify internal values, use constructor.
a5c1f6         Some nice things about the constructor:
a0fd50             frac() = 0/1, frac(5) = 5/1.
d41d8c
77eacc         Be careful that the numerator and denominator might overflow
ecaa93         if lcm is too big.
d15290         In those cases, you can always do frac<big_int>, but that will
162d84         be painful to code.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
4fce64 template <class T>
4cf1ca struct frac
f95b70 {
e75828     T a, b; // b can't be negative, very important.
d41d8c
191fc6     explicit frac(T a = 0, T b = 1) : a(a), b(b) { simpl(); }
d41d8c
7d70f7     void simpl()
f95b70     {
8eb5bb         T g = __gcd(abs(a), abs(b)) * sign(b); // Make b positive.
fe7245         a /= g;
ee2d42         b /= g;
cbb184     }
d41d8c
d59b8a     bool operator<(const frac &rhs) const
f95b70     {
5c6427         return a * rhs.b < rhs.a * b;
cbb184     }
d41d8c
7ebf19     bool operator>(const frac &rhs) const
f95b70     {
2ab79c         return rhs < *this;
cbb184     }
d41d8c
d60bf3     bool operator==(const frac &rhs) const // TODO: untested.
f95b70     {
77c0b8         return !(*this < rhs) && !(rhs < *this);
cbb184     }
d41d8c

```



```
473b74     frac operator*(const frac &rhs) const
f95b70     {
f0117d         return frac(a * rhs.a, b * rhs.b);
cbb184     }
d41d8c
04b5a1     frac operator+(const frac &rhs) const
f95b70     {
3ff11f         T m = (b * rhs.b) / __gcd(b, rhs.b);
24edd6         return frac(a * (m / b) + rhs.a * (m / rhs.b), m);
cbb184     }
d41d8c
c8ca1d     frac operator-(void) const
f95b70     {
132fb3         return frac(-a, b);
cbb184     }
d41d8c
de243f     frac operator-(const frac &rhs) const
f95b70     {
111760         return (*this) + (-rhs);
cbb184     }
d41d8c
d63a85     frac operator/(const frac &rhs) const
f95b70     {
f5299b         return (*this) * frac(rhs.b, rhs.a);
cbb184     }
d41d8c
9e018a     friend ostream &operator<<(ostream &os, const frac &f)
f95b70     {
891d94         return os << f.a << "/" << f.b;
cbb184     }
2145c1 };
d41d8c
Full file hash: c8862e
```

8.4 Integration

```

d41d8c  /*
f64ead    Numerical Integration:
c14d11      Given a function f and an interval [a, b] estimates integral
1aa194      of f(x) dx from a to b.
bfe460      Error is in theory inversely proportional to n^4.
d41d8c
b95cae    Usage:
be1ead      n, the number of intervals must be even.
d41d8c
3db72f    Author: Arthur Pratti Dadalto
c4c9bd  */
d41d8c
044d82  template <class F>
7d9945  double simpsons(const F &f, int n /* even */, double a, double b)
f95b70  {
46af34      double retv = f(a) + f(b);
d025af      double h = (b - a) / n;
acfc81      for (int i = 1; i < n; i += 2)
900086          retv += 4 * f(a + i * h);
1c3900      for (int i = 2; i < n; i += 2)
6c1313          retv += 2 * f(a + i * h);
d41d8c
055fe5      retv *= h / 3;
6272cf      return retv;
cbb184  }
d41d8c
d41d8c  // Sample usage:
d41d8c  // int main(void)
d41d8c  // {
d41d8c  //     printf("%.20lf\n", simpsons([](double x) { return pow(sin(M_PI *
      x / 2.0), 3.2);}, 2000, 0, 2));
d41d8c  // }
Full file hash: caa0e5

```

8.5 linalg

```

5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
f92339     Vector and matrix operations:
687bbc     Details are given in each function.
a2b08b     vec inherits from vector<T>, so there is a lot you can do
a0e9e1     with it.
5ae524     Also, mat inherits from vector<vec<T>>.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
d41d8c
1ef4c8     Source: some of it from https://github.com/
9d5d39         kth-competitive-programming/
3dab19         kactl/blob/master/content/numerical/
9a6abd         MatrixInverse.h
c4c9bd */
d41d8c
4fce64 template <class T>
fe4002 struct vec : vector<T>
f95b70 {
469362     vec(int n) : vector<T>(n) {}
d41d8c
d41d8c     // c = a*x + b*y
e918cb     static void linear_comb(const vec &a, T x, const vec &b, T y, vec &
c)
f95b70     {
8fe753         for (int i = 0; i < sz(a); i++)
75e753             c[i] = a[i] * x + b[i] * y;
cbb184     }
d41d8c
d41d8c     // return a*x + b*y
250f88     static vec linear_comb(vec a, T x, const vec &b, T y)
f95b70     {
4fec85         linear_comb(a, x, b, y, a);
3f5343         return a;
cbb184     }
2145c1 };
d41d8c
4fce64 template <class T>
dade1f struct mat : vector<vec<T>>
f95b70 {
d41d8c     // Creates a zero-filled matrix of n rows and m columns.
2d2b5d     mat(int n, int m) : vector<vec<T>>(n, vec<T>(m)) {}
d41d8c
d41d8c     // c = a * x + b * y
762fbc     static void linear_comb(const mat &a, T x, const mat &b, T y, mat &
c)
f95b70     {

```

```

8fe753     for (int i = 0; i < sz(a); i++)
f47ed7         for (int j = 0; j < sz(a[i]); j++)
4f844b             c[i][j] = a[i][j] * x + b[i][j] * y;
cbb184     }
d41d8c
d41d8c     // return a * x + b * y
08e6ea     static mat linear_comb(mat a, T x, const mat &b, T y)
f95b70     {
4fec85         linear_comb(a, x, b, y, a);
3f5343         return a;
cbb184     }
d41d8c
13fd2a     mat operator-(const mat &b) const { return linear_comb(*this, T(1),
    b, T(-1)); }
d41d8c
0138fa     mat operator+(const mat &b) const { return linear_comb(*this, T(1),
    b, T(1)); }
d41d8c
93d3e8     mat operator*(const T &x) { return linear_comb(*this, x, *this, T
    (0)); }
d41d8c
d41d8c     // Absolutely does not work for int.
72c1fd     mat operator/(const T &x) const { return linear_comb(*this, T(1) /
    x, *this, T(0)); }
d41d8c
d41d8c     // Multiplication of NxR matrix and a RxM matrix.
d41d8c     // TODO test me on non-square.
c36e39     mat operator*(mat b) const
f95b70     {
3f45f0         int n = (*this).size();
29292f         int m = b[0].size();
2fda01         int r = (*this)[0].size();
a13aec         mat retv(n, m);
83008c         for (int i = 0; i < n; i++)
a75dd3             for (int j = 0; j < m; j++)
608272                 for (int k = 0; k < r; k++)
7c3a99                     retv[i][j] = retv[i][j] + (*this)[i][k] * b[k][j];
d41d8c
6272cf         return retv;
cbb184     }
d41d8c
d41d8c     // Returns inverse of matrix (assuming it is square and
d41d8c     // non-singular).
d41d8c     // Runs in O(n^3).
d41d8c     // Absolutely does not work for int.
14566d     mat inverse() // TODO: test singular.
f95b70     {
d23a72         int n = sz(*this);
bca455         mat a(n, 2 * n); // A is Nx2N: X|I.
f7f2d1         vector<int> col(n); // Will be using column pivoting,

```

```

d41d8c          // so need to remember original columns.
83008c  for (int i = 0; i < n; i++)
f95b70  {
f90a6b    for (int j = 0; j < n; j++)
c1c7c0      a[i][j] = (*this)[i][j];
34ac5b      a[i][i + n] = T(1);
6dcd38      col[i] = i;
cbb184    }

d41d8c  for (int i = 0; i < n; i++)
f95b70  {
903ccf    int r = i, c = i;
775cab    for (int j = i; j < n; j++)
90f1d8      for (int k = i; k < n; k++)
f78c7f        if (abs(a[j][k]) > abs(a[r][c]))
d4c894          r = j, c = k;

d41d8c    // assert(abs(a[r][c]) > EPS); Uncomment to check singular
d41d8c    // matrix
a2fa24    swap(a[i], a[r]);

d41d8c    for (int j = 0; j < n; j++)
f90a6b      swap(a[j][i], a[j][c]), swap(a[j][i + n], a[j][c + n]);
c8cc8f    swap(col[i], col[c]);
c1d48e

d41d8c    vec<T>::linear_comb(a[i], T(1) / a[i][i], a[i], T(0), a[i]);
b70d15    a[i][i] = T(1);
67830d

d41d8c    for (int j = i + 1; j < n; j++)
197ab1      vec<T>::linear_comb(a[j], T(1), a[i], -a[j][i], a[j]);
3704dc    }

cbb184    // Right now A is:
d41d8c    //
d41d8c    // 1 * *
d41d8c    // 0 1 *
d41d8c    // 0 0 1
d41d8c    //
d41d8c    // Next we remove non-1s from right to left.

d41d8c  for (int i = n - 1; i > 0; i--)
917d8b    for (int j = 0; j < i; j++)
c791cd      vec<T>::linear_comb(a[j], T(1), a[i], -a[j][i], a[j]);
3704dc

d41d8c  mat retv(n, n);
c70ad2  for (int i = 0; i < n; i++)
83008c    for (int j = 0; j < n; j++)
f90a6b      retv[col[i]][col[j]] = a[i][j + n];
4eb40a  return retv;
6272cf
cbb184  }

```

```
2145c1  };
```

```
Full file hash: 2c7bde
```

8.6 NTT

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
79dc60     Number Theoretic Transform:
5dda9c         FFT allows multiplication of two polynomials in  $O(n \log n)$ 
ebdf6e         where you need the coefficients modulo some specific prime.
d41d8c
b95cae     Usage:
e2ebcc         Can be used for convolutions modulo specific nice primes
5b2366         of the form  $(b * 2^a + 1)$ , where the convolution result
149fb3         has size at most  $2^a$ .
f58885         Inputs must be in  $[0, \text{mod})$ .
d41d8c
1d1558     Source: https://cp-algorithms.com/algebra/fft.html
c4c9bd */
d41d8c
b5e822 const ll mod = (119 << 23) + 1, root = 62; // = 998244353
d41d8c // For  $p < 2^{30}$  there is also e.g.  $5 << 25$ ,  $7 << 26$ ,  $479 << 21$ 
d41d8c // and  $483 << 21$  (same root). The last two are  $> 10^9$ .
d41d8c
4c88e6 ll modpow(ll b, ll e)
f95b70 {
d5469f     ll ans = 1;
36e90c     for (; e; b = b * b % mod, e /= 2)
654342         if (e & 1)
6a3d49             ans = ans * b % mod;
ba75d2     return ans;
cbb184 }
d41d8c
192b04 typedef vector<ll> vl;
3f39ef void ntt(vl &a, vl &rt, vl &rev, int n)
f95b70 {
83008c     for (int i = 0; i < n; i++)
b3fd0     if (i < rev[i])
1e645d         swap(a[i], a[rev[i]]);
657bcd     for (int k = 1; k < n; k *= 2)
1e5238         for (int i = 0; i < n; i += 2 * k)
68f8d4             for (int j = 0; j < k; j++)
f95b70                 {
86eb2e                     ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
93de95                     a[i + j + k] = (z > ai ? ai - z + mod : ai - z);
589658                     ai += (ai + z >= mod ? z - mod : z);
cbb184                 }
cbb184 }
d41d8c
92da3f vl conv(const vl &a, const vl &b)
f95b70 {
41f63a     if (a.empty() || b.empty())

```

```

21d05e         return {};
6b2b5b     int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1 << B;
642805     vl L(a), R(b), out(n), rt(n, 1), rev(n);
6b422b     L.resize(n), R.resize(n);
d41d8c
83008c     for (int i = 0; i < n; i++)
a17004         rev[i] = (rev[i / 2] | (i & 1) << B) / 2;
d41d8c
b2c3ee     ll curL = mod / 2, inv = modpow(n, mod - 2);
41487f     for (int k = 2; k < n; k *= 2)
f95b70     {
8939f5         ll z[] = {1, modpow(root, curL /= 2)};
1d31c1         for (int i = k; i < 2 * k; i++)
25629e             rt[i] = rt[i / 2] * z[i & 1] % mod;
cbb184     }
d41d8c
2fa9bc     ntt(L, rt, rev, n);
89ff07     ntt(R, rt, rev, n);
83008c     for (int i = 0; i < n; i++)
1cbba8         out[-i & (n - 1)] = L[i] * R[i] % mod * inv % mod;
bc5af9     ntt(out, rt, rev, n);
c20361     return {out.begin(), out.begin() + s};
cbb184 }
Full file hash: ebd7dd

```


8.7 Simplex

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
458b90     Simplex:
6956ec         Optimizes a linear program of the form:
15b127             maximize c*x, s.t. a*x <ops> b, x >= 0.
7b88d6             Each constraint can use a different operator from {<= >= ==}.
352cc0             Not polynomial, but got AC 150 ms with 4000 constraints and
021e31             200 variables.
d41d8c
b95cae     Usage:
6c3c6c         Call run_simplex, with the number of constraints and
852ff8         variables, a, b, ops and c (as specified above).
f28b8e         Return value is ok if solution was found, unbounded if
cf1b1b         objective value can be infinitely large
eb42f2         or infeasible if there is no solution given the constraints.
d41d8c
38fe13         The value of each variable is returned in vector res.
90f249         Objective function optimal value is also returned.
060dc4         Sample usage is commented below.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
4fce64 template <class T>
fe4002 struct vec : vector<T>
f95b70 {
469362     vec(int n) : vector<T>(n) {}
d41d8c
d41d8c     // c = a*x + b*y
e918cb     static void linear_comb(const vec &a, T x, const vec &b, T y, vec &
c)
f95b70     {
8fe753         for (int i = 0; i < sz(a); i++)
75e753             c[i] = a[i] * x + b[i] * y;
cbb184     }
2145c1 };
d41d8c
4fce64 template <class T>
dade1f struct mat : vector<vec<T>>
f95b70 {
d41d8c     // Creates a zero-filled matrix of n rows and m columns.
2d2b5d     mat(int n, int m) : vector<vec<T>>(n, vec<T>(m)) {}
d41d8c
d41d8c     // Erase row 0(n^2).
82436c     void erase_row(int i)
f95b70     {
7c9f9f         this->erase(this->begin() + i);

```

```

cbb184     }
d41d8c
d41d8c     // Erase column 0(n^2).
1b22c6     void erase_col(int j)
f95b70     {
798fc8         for (int i = 0; i < sz(*this); i++)
a7796a             (*this)[i].erase((*this)[i].begin() + j);
cbb184     }
2145c1 };
d41d8c
d3ff82 namespace simplex
f95b70 {
d41d8c     // Any value within [-EPS, +EPS] will be considered equal to 0.
05667a     const double EPS = 1e-6;
d41d8c
5e6f5b     enum op { ge, le, eq };
d41d8c
242dbb     enum optimization_status { ok, unbouded, infeasible };
d41d8c
4d9580     int get_entering_var(mat<double> &tab)
f95b70     {
d41d8c         // Get first non-artificial variable with negative objective
d41d8c         // coefficient. If none, return -1. (could instead return most
d41d8c         // negative, but that could cycle)
682f62         for (int i = 0; i < sz(tab[0]) - 1; i++)
72e0d2             if (tab[0][i] < -EPS)
d9a594                 return i;
daa4d1         return -1;
cbb184     }
d41d8c
201003     int get_exiting_var_row(mat<double> &tab, int entering_var)
f95b70     {
d41d8c         // Get smallest value of val and first in case of tie. If none,
d41d8c         // return -1.
fcb2fc         int retv = -1;
6213b9         double val = -1.0;
a07064         for (int i = 1; i < sz(tab); i++)
f95b70         {
d41d8c             // If strictly positive, it bounds the entering var.
dcda72             if (tab[i][entering_var] > EPS)
f95b70             {
d41d8c                 // Entering var will be bounded by
d41d8c                 // tab[i][tab.size().second - 1] / tab[i][entering_var].
d41d8c                 // val could be slightly negative if
d41d8c                 // tab[i][tab.size().second - 1] = -0.
393d3f                 if (val == -1.0 || tab[i][sz(tab[i]) - 1] / tab[i][entering_var
] < val)
f95b70             {
78d87c                 val = tab[i][sz(tab[i]) - 1] / tab[i][entering_var];
52cece                 retv = i;

```

```

cbb184     }
cbb184     }
cbb184     }
d41d8c
6272cf     return retv;
cbb184     }
d41d8c
ed25d2 optimization_status solve_tab(mat<double> &tab, vector<int> &
    basic_var)
f95b70     {
d41d8c         // artificial_count is the number of variables at the end we
d41d8c         // should ignore.
a17ec7         int entering_var;
6b7846         while ((entering_var = get_entering_var(tab)) != -1)
f95b70         {
6c0a23             int exiting_var_row = get_exiting_var_row(tab, entering_var);
d41d8c
d41d8c             // If no exiting variable bounds the entering variable, the
d41d8c             // objective is unbounded.
813335             if (exiting_var_row == -1)
914a2e                 return optimization_status::unbounded;
d41d8c
d41d8c             // Set new basic var coefficient to 1.
89c7a2             vec<double>::linear_comb(tab[exiting_var_row], (1.0 / tab[
                exiting_var_row][entering_var]), tab[exiting_var_row], 0.0, tab[
                exiting_var_row]);
d41d8c
d41d8c             // Gaussian elimination of the other rows.
c7a773             for (int i = 0; i < sz(tab); i++)
81c379                 if (i != exiting_var_row)
ed2730                     if (abs(tab[i][entering_var]) > EPS)
7ad878                         vec<double>::linear_comb(tab[i], 1.0, tab[exiting_var_row],
                            -tab[i][entering_var], tab[i]);
d41d8c
64dd6a             basic_var[exiting_var_row] = entering_var;
cbb184         }
d41d8c
c52f1c         return optimization_status::ok;
cbb184     }
d41d8c
d41d8c     // maximize c*x, s.t. a*x <ops> b. x >= 0.
f1a105 optimization_status run_simplex(int num_constraints, int num_vars,
    mat<double> a, vec<op> ops, vec<double> b, vec<double> c, vec<double> &res
    , double &obj_val)
f95b70     {
334f46         for (int i = 0; i < num_constraints; i++)
5f946c             if (ops[i] == op::ge)
f95b70             {
d41d8c                 // Beyond this point "ge" constraints won't exist.
44438f                 vec<double>::linear_comb(a[i], -1, a[i], 0, a[i]); // a[i] *=

```

```

-1;
250b4d      b[i] *= -1;
1c38d4      ops[i] = op::le;
cbb184     }
d41d8c
0264da     int num_artificial_variables = 0;
371f2b     int num_slack_variables = 0;
334f46     for (int i = 0; i < num_constraints; i++)
f95b70     {
0ec40f         if (ops[i] == op::le)
f95b70         {
37acf9             num_slack_variables++;
cbb184         }
d41d8c
359aa4         if ((ops[i] == op::le && b[i] < -EPS) || ops[i] == op::eq)
f95b70         {
d41d8c             // If we have rhs strictly negative in a inequality or an
d41d8c             // equality constraint, we need an artificial val.
fc36e6             num_artificial_variables++;
cbb184         }
cbb184     }
d41d8c
854c33     mat<double> tab(num_constraints + 1, num_vars + num_slack_variables
+ num_artificial_variables + 1);
9a9a70     vector<int> basic_var(num_constraints + 1);
775265     vector<int> slack_cols, artificial_cols;
7f63aa     for (int i = num_vars; i < num_vars + num_slack_variables; i++)
10c71f         slack_cols.push_back(i);
e0b615     for (int i = num_vars + num_slack_variables; i < num_vars +
num_slack_variables + num_artificial_variables; i++)
eafbfb         artificial_cols.push_back(i);
c70a50     int rhs_col = num_vars + num_slack_variables +
num_artificial_variables;
d41d8c
d41d8c     // First objective will be to have artificial variables equal to 0
017565     for (int i : artificial_cols)
b98201         tab[0][i] = 1;
d41d8c
9c49f5     for (int i = 0, k = 0, l = 0; i < num_constraints; i++)
f95b70     {
861a15         for (int j = 0; j < num_vars; j++)
e3832e             tab[i + 1][j] = a[i][j];
d41d8c
0ec40f         if (ops[i] == op::le)
141495             tab[i + 1][slack_cols[l++]] = 1;
d41d8c
142f37         tab[i + 1][rhs_col] = b[i];
d41d8c
359aa4         if ((ops[i] == op::le && b[i] < -EPS) || ops[i] == op::eq)
f95b70         {

```

```

d41d8c      // Basic var will be artificial
2a6978      if (b[i] < -EPS)
009fda      vec<double>::linear_comb(tab[i + 1], -1, tab[i + 1], 0, tab[i
+ 1]); // a[i] *= -1;
d41d8c
86fab4      tab[i + 1][artificial_cols[k++]] = 1;
116454      basic_var[i + 1] = artificial_cols[k - 1];
d41d8c
06db08      vec<double>::linear_comb(tab[0], 1.0, tab[i + 1], -1.0, tab[0])
;
cbb184      }
2954e9      else // Basic var will be slack var.
f95b70      {
ae77b6      basic_var[i + 1] = slack_cols[l - 1];
cbb184      }
cbb184      }
d41d8c
df8d17      assert(solve_tab(tab, basic_var) == optimization_status::ok);
d41d8c
d41d8c      // Best solution could not bring artificial variables to 0
d41d8c      // (objective max Z = sum(-xa)).
fe0d64      if (tab[0][sz(tab[0]) - 1] < -EPS)
94b8a3      return optimization_status::infeasible;
d41d8c
d41d8c      // If we have an artificial variable on the base with xb = 0, we
d41d8c      // need to remove it.
e6411b      for (int i = 1; i < sz(basic_var); i++)
0778cb      if (basic_var[i] >= num_vars + num_slack_variables)
f95b70      {
d41d8c      // Find non-artificial replacement.
e2f213      for (int j = 0; j < sz(tab[i]) - 1 - num_artificial_variables;
j++)
f95b70      {
d41d8c      // If non-zero value in row, we can replace.
a8880b      if (j != basic_var[i] && abs(tab[i][j]) > EPS)
f95b70      {
d41d8c      // Remove from the other rows.
b5fa44      vec<double>::linear_comb(tab[i], 1.0 / tab[i][j], tab[i],
0, tab[i]);
d41d8c
443db5      for (int k = 0; k < sz(tab); k++)
635b4c      if (k != i)
f95b70      {
e76184      if (abs(tab[k][j]) > EPS)
4b6b27      vec<double>::linear_comb(tab[k], 1, tab[i], -tab[k][j
], tab[k]);
cbb184      }
d41d8c
d41d8c      // Basic variable replacement done, so proceed to
d41d8c      // next basic_var.

```

```

7e0f27         basic_var[i] = j;
c2bef1         break;
cbb184     }
cbb184     }
cbb184     }
d41d8c
ca2210     for (int i = sz(tab) - 1; i > 0; i--)
0778cb         if (basic_var[i] >= num_vars + num_slack_variables)
f95b70     {
d41d8c         // Could not replace basic var, so constraint is redundant
2cd1fb         tab.erase_row(i);
fe14c7         basic_var.erase(basic_var.begin() + i);
cbb184     }
d41d8c
d41d8c     // Remove artificial variable columns.
5c3178     for (int i = sz(artificial_cols) - 1; i >= 0; i--)
9a226e         tab.erase_col(artificial_cols[i]);
d41d8c
1311b7     for (int i = 0; i < sz(tab[0]); i++)
d2677f         tab[0][i] = 0;
f17293     for (int i = 0; i < num_vars; i++)
94256d         tab[0][i] = -c[i];
d41d8c
a07064     for (int i = 1; i < sz(tab); i++)
b39526         vec<double>::linear_comb(tab[0], 1, tab[i], -tab[0][basic_var[i]
    ]], tab[0]);
d41d8c
54ad02     optimization_status status = solve_tab(tab, basic_var);
d41d8c
b68670     res = vec<double>(num_vars);
e6411b     for (int i = 1; i < sz(basic_var); i++)
047b20         if (basic_var[i] < num_vars)
81f54e             res[basic_var[i]] = tab[i][sz(tab[i]) - 1];
d41d8c
a3473e     obj_val = tab[0][sz(tab[0]) - 1];
d41d8c
62d3d5     return status;
cbb184 }
cbb184 } // namespace simplex
d41d8c
d41d8c /*
13a4b1 int main(void)
f95b70 {
14e0a7     int n, m;
aa3380     cin >> n >> m;
d41d8c
37ce14     int num_constraints = m, num_vars = n;
d41d8c
d41d8c     // maximize c*x, s.t. a*x <ops> b. x >= 0.
2626bb     mat<double> a(num_constraints, num_vars);

```

```
84d434    vec<double> b(num_constraints);
01b2af    vec<simplex::op> ops(num_constraints);
dabb12    vec<double> c(num_vars);
40ca17    vec<double> res(num_vars);
d41d8c
83008c    for (int i = 0; i < n; i++)
a733f7        cin >> c[i];
d41d8c
94f72b    for (int i = 0; i < m; i++)
f95b70    {
7ba74c        int l, r, x;
15994b        cin >> l >> r >> x;
0dfebd        for (int j = l - 1; j <= r - 1; j++)
a21125            a[i][j] = 1;
df0b9d        b[i] = x;
80367f        ops[i] = simplex::op::le;
cbb184    }
d41d8c
1afc12    double ans;
dd6c28    simplex::run_simplex(num_constraints, num_vars, a, ops, b, c, res,
    ans);
d41d8c
530b75    cout << ((long long)(ans + 0.5)) << endl;
cbb184    }
c4c9bd    */
```

Full file hash: 46f321

9 String

9.1 Aho Corasick

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
30562e     Aho-Corasick: O(alpha_size * string_sum)
4e9057     In general, multiple pattern string matching tree/automaton.
d41d8c
a6fcc1     Keep in mind that find_all can be O(N*sqrt(N)) if no duplicate
b16fe5     patterns. (N is total string length)
d41d8c
ca2095     Constraints:
76adb2     chars in the string are all in the interval
66258a     [first, first + alpha_size - 1].
3da079     This will not free some memory on object destruction.
390590     Duplicate patterns are allowed, empty patterns are not.
d41d8c
b95cae     Usage:
df3a72     Set alpha_size and the first char in the alphabet.
e98cb2     Call constructor passing the list of pattern strings.
0f1b09     Use one of find, find_all ... to process a text or do your own
9d306a     thing.
42a2c2     To find the longest words that start at each position, reverse
ac5c99     all input.
3439d1     Bottleneck in this code is memory allocation.
91a84c     For 10^6 total string size, memory usage can be up to 300 Mb.
d41d8c
b34145     You can save time:
3cd07b         list_node, match_list, match_list_last are only needed to
d10915         list all matches.
57e4db         atm automaton table can be cut to reduce memory usage.
018d49         The text processing stuff is also optional.
02e3ad         Node memory can be one big array instead of vector.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
e7f92a struct aho_corasick
f95b70 {
da45ec     enum
f95b70     {
033315         alpha_size = 26, // Number of chars in the alphabet.
b3d02f         first = 'a' // First char.
2145c1     };
d41d8c
fc487b struct list_node // Simple linked list node struct.
f95b70 {
53e65f     int id;

```



```

6ec94b     list_node *next;
ff56a7     explicit list_node(int id, list_node *next) : id(id), next(next)
    {}
2145c1 };
d41d8c
e4accb struct node
f95b70 {
ca8b7e     int fail = -1;          // node failure link (aka suffix link).
d41d8c
2eb620     int nmatches = 0;      // Number of matches ending in this
d41d8c         // node.
d41d8c
9005b9     int next[alpha_size]; // Next node in trie for each letter.
d41d8c         // Replace with unordered_map or list
d41d8c         // if memory is tight.
d41d8c
c0f747     int atm[alpha_size]; // Optional: Automaton state
d41d8c         // transition table. Simpler text
d41d8c         // processing.
d41d8c
d41d8c     // Pointer to first node in linked list of matches.
d41d8c     // List ends with null pointer.
44edb6     list_node *match_list = nullptr;
d41d8c
d41d8c     // Internal: pointer to last node in list of matches
d41d8c     // (before bfs), or null if empty list.
01009c     list_node *match_list_last = nullptr;
d41d8c
d41d8c     // Start with all invalid transitions.
e6fb82     node() { memset(next, -1, sizeof(next)); }
2145c1 };
d41d8c
b9ea22     vector<node> nodes;
d41d8c
9b61f6     aho_corasick(const vector<string> &pats)
f95b70     {
225eb3         nodes.emplace_back(); // Make root node 0.
b5bf96         for (int i = 0; i < sz(pats); i++)
f95b70         {
b3da3c             int cur = 0; // Start from root.
9f5c69             for (int j = 0; j < sz(pats[i]); j++)
f95b70             {
ec0388                 int k = pats[i][j] - first;
d41d8c
10937b                 if (nodes[cur].next[k] <= 0)
f95b70                 {
d41d8c                     // Make new node if needed.
976fa3                     nodes[cur].next[k] = sz(nodes);
225eb3                     nodes.emplace_back();
cbb184                 }

```

```

d41d8c
47b49f         cur = nodes[cur].next[k];
cbb184     }
d41d8c
d41d8c         // Add logic here if additional data is needed on matched
d41d8c         // strings.
4daeea     nodes[cur].nmatches++;
d41d8c         // Add string to node list of matches.
45f177     nodes[cur].match_list = new list_node(i, nodes[cur].match_list)
;
fe38fe         if (nodes[cur].nmatches == 1)
947da5         nodes[cur].match_list_last = nodes[cur].match_list;
cbb184     }
d41d8c
26a528     queue<int> q;
d41d8c         // Define fail for first level.
6733a6     for (int i = 0; i < alpha_size; i++)
f95b70     {
d41d8c         // Invalid transitions from 0 now become valid self
d41d8c         // transitions.
e8dc83         if (nodes[0].next[i] == -1)
fb628f             nodes[0].next[i] = 0;
d41d8c
d41d8c         // Automaton state transition table.
7d3171         nodes[0].atm[i] = nodes[0].next[i];
d41d8c
d41d8c         // Single letter nodes have fail = 0 and go in the queue.
bc34bf         if (nodes[0].next[i] > 0)
f95b70         {
eded92             q.push(nodes[0].next[i]);
9b22e6             nodes[nodes[0].next[i]].fail = 0;
cbb184         }
cbb184     }
d41d8c
ee6bdd     while (!q.empty()) // Use bfs to compute fail for next level.
f95b70     {
69faa7         int cur = q.front();
833270         q.pop();
d41d8c
6733a6         for (int i = 0; i < alpha_size; i++)
af4a6e             if (nodes[cur].next[i] > 0) // Don't use -1 and don't
d41d8c                 // use transition to root.
f95b70             {
d41d8c                 // Unrelated to code below, filling automaton.
3ecdd3                 nodes[cur].atm[i] = nodes[cur].next[i];
d41d8c
d41d8c                 // Computing fail for next node and putting it in
d41d8c                 // the queue.
3ae7da                 int prox = nodes[cur].next[i];
53ef92                 q.push(prox);

```

```

d41d8c
f252cb         int state = nodes[cur].fail;
c66324         while (nodes[state].next[i] == -1)
d712e2             state = nodes[state].fail;
d41d8c
7836db         nodes[prox].fail = nodes[state].next[i];
d41d8c
d41d8c         // Add logic here if additional data is needed on
d41d8c         // matched strings.
2940ed         nodes[prox].nmatches += nodes[nodes[prox].fail].nmatches;
d41d8c
d41d8c         // Add in O(1) list from fail link to next node's
d41d8c         // list.
d41d8c         // Operation: a->b->null c->null to a->b->c->null.
59ed4d         (nodes[prox].match_list_last ? nodes[prox].match_list_last
->next : nodes[prox].match_list) = nodes[nodes[prox].fail].match_list;
cbb184     }
2954e9     else
f95b70     {
a04598         nodes[cur].atm[i] = nodes[nodes[cur].fail].atm[i];
cbb184     }
cbb184 }
cbb184 }
d41d8c
d41d8c // Optional
d41d8c // Returns a vector retv such that, for each text position i:
d41d8c // retv[i] is the index of the largest pattern ending at position
d41d8c // i in the text.
d41d8c // If retv[i] == -1, no pattern ends at position i.
32246d vector<int> find(const string &text)
f95b70 {
107323     vector<int> retv(sz(text));
b3da3c     int cur = 0;
d41d8c
77447e     for (int i = 0; i < sz(text); i++)
f95b70     {
13dae2         cur = nodes[cur].atm[text[i] - first];
29e58f         retv[i] = (nodes[cur].match_list ? nodes[cur].match_list->id :
-1);
cbb184     }
d41d8c
6272cf     return retv;
cbb184 }
d41d8c
d41d8c // Optional
d41d8c // Returns a vector retv such that, for each text position i:
d41d8c // retv[i] is the number of pattern matches ending at position i
d41d8c // in the text.
48d0f2 vector<int> count(const string &text)
f95b70 {

```

```

107323     vector<int> retv(sz(text));
b3da3c     int cur = 0;
d41d8c
77447e     for (int i = 0; i < sz(text); i++)
f95b70     {
13dae2         cur = nodes[cur].atm[text[i] - first];
1a43d3         retv[i] = nodes[cur].nmatches;
cbb184     }
d41d8c
6272cf     return retv;
cbb184 }
d41d8c
d41d8c // Optional
d41d8c // Returns a vector retv such that, for each text position i:
d41d8c // retv[i] is a list of indexes to the patterns ending at position
d41d8c // i in the text.
d41d8c // These lists will be sorted from largest to smallest pattern
d41d8c // length.
d41d8c // Keep in mind that find_all can be O(N*sqrt(N)) if no duplicate
d41d8c // patterns. (N is total string length)
4e5a4c vector<vector<int>> find_all(const string &text)
f95b70 {
77b54a     vector<vector<int>> retv(sz(text));
b3da3c     int cur = 0;
d41d8c
77447e     for (int i = 0; i < sz(text); i++)
f95b70     {
13dae2         cur = nodes[cur].atm[text[i] - first];
d82b0e         for (auto n = nodes[cur].match_list; n != nullptr; n = n->next)
4c4784             retv[i].push_back(n->id);
cbb184     }
d41d8c
6272cf     return retv;
cbb184 }
d41d8c
d41d8c // Optional
d41d8c // Returns a vector retv such that:
d41d8c // retv is a list of indexes to the patterns ending at position
d41d8c // pos in the text.
d41d8c // This list will be sorted from largest to smallest pattern
d41d8c // length.
251c66 vector<int> find_all_at_pos(const string &text, int pos)
f95b70 {
aeb888     vector<int> retv;
b3da3c     int cur = 0;
d41d8c
77447e     for (int i = 0; i < sz(text); i++)
f95b70     {
13dae2         cur = nodes[cur].atm[text[i] - first];
d41d8c

```

```
c57c6f         if (i == pos)
d82b0e         for (auto n = nodes[cur].match_list; n != nullptr; n = n->
    next)
1ad617             retv.push_back(n->id);
cbb184         }
d41d8c
6272cf         return retv;
cbb184     }
2145c1 };
Full file hash: 2ec64b
```

9.2 Hash

```

5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
032d27     String hashing:
4d6ea3         Get polynomial hash for any substring in O(1) after O(n)
76e86b         preprocessing.
d41d8c
b95cae     Usage:
cac48f         Good values c = 137, mod = 10^9 + 7.
92282d         If necessary to check too many pairs of hashes, use two
c7c0c4         different hashes.
d41d8c
10785a         If hashing something other than english characters:
eb7765             - Don't have elements with value 0.
0e7713             - Use c > max element value.
c4c9bd */
d41d8c
164df5 struct hash_interval
f95b70 {
8805cb     ll c, mod;
dcf2cf     vector<ll> h, p;
d41955     hash_interval(const string &s, ll c, ll mod) : c(c), mod(mod), h(sz
(s) + 1), p(sz(s) + 1)
f95b70     {
c4bbd4         p[0] = 1;
cdca7d         h[0] = 0;
11e703         for (int i = 0; i < sz(s); i++)
f95b70             {
909b2c                 h[i + 1] = (c * h[i] + s[i]) % mod;
e69dd9                 p[i + 1] = (c * p[i]) % mod;
cbb184             }
cbb184         }
d41d8c
d41d8c // Returns hash of interval s[a ... b] (where 0 <= a <= b < sz(s))
d12ad5 ll get(int a, int b)
f95b70 {
2c612c     return (h[b + 1] - ((h[a] * p[b - a + 1]) % mod) + mod) % mod;
cbb184 }
2145c1 };
Full file hash: b9525a

```

9.3 KMP

```

5d1131 #include "../..//contest/header.hpp"
d41d8c
d41d8c /*
8dec4f     Prefix Function and KMP:
e45403     Computes prefix function for a given string in O(n).
16bb22     String matching in O(n + m).
37f784     No need to be strings, you can use vector<int> since the
e68a76     algorithms don't depend on the alphabet size, they only
f21b2e     perform equality comparisons.
b5efd9     Usage is explained in each function.
d41d8c
3db72f     Author: Arthur Pratti Dadalto
c4c9bd */
d41d8c
d41d8c // Returns the prefix function for the given string.
d41d8c // pi[i] for 0 <= i <= s.size() (s.size() + 1 elements).
d41d8c // pi[i] considers the prefix of string s having size i.
d41d8c // pi[i] is the size of its (the prefix's) largest proper prefix
d41d8c // which is also a suffix.
d41d8c // For "aabaaab", pi is {0,0,1,0,1,2,2,3}
4fce64 template <class T>
8fa849 vector<int> prefix_function(T s)
f95b70 {
d2c5d5     vector<int> pi(s.size() + 1, 0);
a94e4a     for (int i = 2; i <= s.size(); i++)
f95b70     {
3f878c         int j = pi[i - 1]; // j is the size of the candidate prefix
d41d8c         // to expand.
d41d8c
4b3f35         while (j > 0 && s[j] != s[i - 1]) // While we still have a
d41d8c         // candidate prefix and it
d41d8c         // can't be expanded.
d41d8c
187475         j = pi[j]; // Go to the next candidate prefix.
d41d8c
d41d8c         // If candidate prefix can be expanded, do it. Otherwise,
d41d8c         // there is no prefix that is also a suffix.
f986f8         pi[i] = s[j] == s[i - 1] ? j + 1 : 0;
cbb184     }
d41d8c
81d1a2     return pi;
cbb184 }
d41d8c
d41d8c // Returns a sorted list of all positions in the text string where
d41d8c // begins an occurrence of the key string.
d41d8c // e.g. kmp("aabaaab", "aab") returns {0, 4}.
4fce64 template <class T>
15b377 vector<int> kmp(T text, T key)

```

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

Full file hash: 415801

9.4 Suffix Array

```

d41d8c
5d1131 #include "../contest/header.hpp"
d41d8c
d41d8c /*
1f77e9     Suffix array:
47c303         Build suffix array and LCP array in  $O((n + \text{lim}) \log n)$  using
9ffdc3          $O(n + \text{lim})$  memory, where lim is the alphabet size.
d41d8c
fbd842         sa[i] is the starting index of the suffix which is i-th in the
ec385d         sorted suffix array.
765a5f         The returned vector is of size s.size()+1,
15c36c         and sa[0] == s.size(). The '\0' char at the end is considered
c45f23         part of the string, so sa[0] = "\0", the prefix starting at
af1165         index s.size().
d41d8c
878881         The lcp array contains longest common prefixes for
e7b14e         neighbouring strings in the suffix array:
e419c1         lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0.
d41d8c
81eeab     Example:
981b73         Computing the LCP and the SA of "GATAGACA"
d33e7b         i sa[i] lcp[i] suffix
fd774f         0 8    0    ""
cac682         1 7    0    "A"
430b3a         2 5    1    "ACA"
d30cc0         3 3    1    "AGACA"
c895f5         4 1    1    "ATAGACA"
1a04b3         5 6    0    "CA"
b1b780         6 4    0    "GACA"
2999cd         7 0    2    "GATAGACA"
08e6dc         8 2    0    "TAGACA"
d41d8c
b95cae     Usage:
6a6409         Important: the input string must not contain any zero values.
95cc8f         Must use C++11 or above.
87e7ae         You can use this for strings of integers, just change the
421ad7         alphabet size.
d41d8c
1d1558     Source: https://github.com/kth-competitive-programming/kactl/blob/
3fd52d     master/content/strings/SuffixTree.h
c4c9bd */
d41d8c
15a9b6 struct suffix_array
f95b70 {
71675a     vector<int> sa, lcp;
092958     suffix_array(const string &s, int lim = 256) // or basic_string<int
> for integer strings.
f95b70     {

```

```

e72340     int n = sz(s) + 1, k = 0, a, b;
f6a0db     vector<int> x(s.begin(), s.end() + 1), y(n), ws(max(n, lim)),
rank(n);
85469f     sa = lcp = y;
eb75f9     iota(sa.begin(), sa.end(), 0);
7707f7     for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p)
f95b70     {
8dff9b         p = j;
00aec0         iota(y.begin(), y.end(), n - j);
83008c         for (int i = 0; i < n; i++)
e9b19c             if (sa[i] >= j)
d0873d                 y[p++] = sa[i] - j;
450a8a         fill(ws.begin(), ws.end(), 0);
83008c         for (int i = 0; i < n; i++)
799bb0             ws[x[i]]++;
7d6bd3         for (int i = 1; i < lim; i++)
f256af             ws[i] += ws[i - 1];
5df399         for (int i = n; i--;)
d01b67             sa[--ws[x[y[i]]]] = y[i];
9dd20c         swap(x, y);
017be6         p = 1;
16ab1b         x[sa[0]] = 0;
d41d8c
aa4866         for (int i = 1; i < n; i++)
f95b70         {
fcb940             a = sa[i - 1];
2d820b             b = sa[i];
0cc036             x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
cbb184         }
cbb184     }
d41d8c
aa4866     for (int i = 1; i < n; i++)
2f33c5         rank[sa[i]] = i;
d41d8c
05cb2b     for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
487069         for (k &&k--, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; k++)
9eecb7             ;
cbb184     }
2145c1 };
Full file hash: 87092f

```

9.5 Suffix Tree

```

d41d8c  /*
e39503  Suffix Tree:
5c6262      A compressed trie is a trie where all redundant nodes are
7fd4f3      eliminated by allowing edges to hold substrings.
ce3de2      A Suffix Tree is a Trie containing all the suffixes of a
d6469f      certain string S.
ceb4ec      Using a dummy character in the end of S guarantees that all
4ea5b1      suffixes end in leafs, and vice-vers.
09d5e4      This code builds the Suffix Tree in  $O(|S| \cdot \lg |\text{Alph}|)$  time and
59af55       $O(|S|)$  memory (where  $|\text{Alph}|$  is the size of the alphabet).
d41d8c
81eeab  Example:
c7a2f4      A dfs through the Suffix Tree of "banana$" looks like:
eb213f      Begin in root
fffe57      Enter through "$"
0c19d4      Leave through "$"
028838      Enter through "na"
fffe57      Enter through "$"
0c19d4      Leave through "$"
c49ab4      Enter through "na$"
f0e081      Leave through "na$"
4e157f      Enter through "banana$"
7b0b76      Leave through "banana$"
b30896      Enter through "a"
fffe57      Enter through "$"
0c19d4      Leave through "$"
028838      Enter through "na"
fffe57      Enter through "$"
0a0cce      Leave through "$"
c49ab4      Enter through "na$"
f0e081      Leave through "na$"
13654d      Leave through "na"
b669ad      Leave through "a"
d41d8c
b95cae  Usage:
a8994a      Create an object Suffix Tree st passing the string as
763b5b      argument, and optionally the dummy character as second
00d570      argument.
94795e      "verify_substring(P)" checks in  $O(|P|)$  if P is a substring of
a5cb5c      S.
d41d8c
d41d8c
b602e4  Author: Augusto Damschi Bernardi
063279      Based on: https://bcc.ime.usp.br/tccs/2016/yancouto/tcc.pdf
c4c9bd  */
5d1131  #include "../contest/header.hpp"
d41d8c
3c9ee1  struct node{

```

```

d41d8c // Each node keeps information about the edge arriving into it.
d41d8c
d41d8c // Keeps left and right index of edge's substring in S.
d41d8c // (may not be the same occurrence one, see "aba$");
3aaf96 int left, right;
d41d8c
d41d8c // *parent points to parent node
d41d8c // *suffix points to node corresponding to [left+1...right]
d41d8c // (by the end of the process exists for every node other than
d41d8c // the root)
75e587 node *parent, *suffix;
d41d8c
d41d8c //next_node[c] points to the kid of current node whose edge
d41d8c // begins with character c (only one by character).
0592ac map<char, node*> next_node;
d41d8c
b3c7aa node(int _left, int _right, node *_parent):
ef39bc left(_left), right(_right), parent(_parent){}
d41d8c
b98e83 ~node(){
9f93fe     for(auto child : next_node)
77845c         delete child.second;
cbb184 }
d41d8c
d41d8c //Lenght of current edge
9fdc7c int len(){
1c55c1     return right - left + 1;
cbb184 }
d41d8c
d41d8c //Convinient way to find kid
dff70b node* next(char c){
42567e     if(next_node.count(c))
30b2fa         return next_node[c];
ea9b0a     return NULL;
cbb184 }
d41d8c
2145c1 };
d41d8c
156e1d struct suffix_tree{
bb7be6     node *root;
6f1605     char dummy;
ac0066     string s;
d41d8c
b9109f     suffix_tree(string _s, char _dummy = '$')
f95b70     {
d9d655         s = _s;
ecbbb7         dummy = _dummy;
0bf292         s += dummy;
fd02fe         root = new node(0, -1, NULL);
d41d8c         // In the beginning of iteration i,j, node cur_node in

```

```

d41d8c // [left...cur_dist] represents [i...j-1]
d41d8c // need_suffix points to node that doesn't have a suffix yet
d41d8c // (at most one at a time, for at most one iteration of i)
f85522 node *cur_node = root;
fba8a7 int cur_dist = -1, i = 0;
d41d8c
d41d8c //Invariants:
d41d8c // *At the beginning of step i,j, s[i...j-1] and all of it's
d41d8c // suffixes are inserted in the suffix trie
d41d8c // *At the beginning of step i,j, cur_node[cur_dist] is the
d41d8c // end point representing s[i...j-1]
d41d8c // *At any increment of i, at most one node doesn't have
d41d8c // "suffix" field defined, and it's stored in "need_suffix"
b97766 for(int j = 0; j < (int)s.size(); j++){
5fb750     char c = s[j];
b63b95     node *need_suffix = NULL;
67d6e0     while(i <= j){
d41d8c         // Inserts s[i...j]
d41d8c
d41d8c         // Case 1: s[i...j] already exists in the suffix tree
d41d8c         // If it's in the next node, move to it
815707         if(cur_dist == cur_node->len() - 1 and cur_node->next
(c) != NULL){
5666ba             cur_node = cur_node->next(c);
977ced             cur_dist = -1;
cbb184         }
d41d8c         // If now we have to take one more char from the
d41d8c         // edge, take it
14012b         if(cur_dist < cur_node->len() - 1 and get_char(
cur_node, cur_dist + 1) == c){
cb48a1             cur_dist += 1;
c2bef1             break;
cbb184         }
d41d8c
d41d8c         // Case 2: s[i...j-1] ends in a node
716c0b         if(cur_dist == cur_node->len() - 1){
b95c44             cur_node->next_node[c] = new node(j, s.size() -
1, cur_node);
d41d8c             // Puts cur_node in s[i_1...j-1]
1e8c16             if(cur_node != root){
55acc1                 cur_node = cur_node->suffix;
bb6442                 cur_dist = cur_node->len() - 1;
cbb184             }
cbb184         }
d41d8c
d41d8c         // Caso 3: s[i...j-1] ends in an edge
4e6b83         else{
d41d8c             // Creates a new node and splits the edge
593b20             node *mid = new node(cur_node->left, cur_node->
left + cur_dist, cur_node->parent);

```

```

9e7635         cur_node->parent->next_node[get_char(mid, 0)] =
mid;
d79b0b         mid->next_node[get_char(cur_node, cur_dist + 1)]
= cur_node;
49e656         cur_node->parent = mid;
e8c75a         cur_node->left += cur_dist + 1;
2f15cb         mid->next_node[s[j]] = new node(j, s.size() - 1,
mid);
d41d8c         // Sets any missing suffix link
07a6af         if(need_suffix != NULL)
b9c09e             need_suffix->suffix = mid;
d41d8c         // Tries to find the suffix link for "mid"
37a0fd         cur_node = mid->parent;
6a5351         int g;
1e8c16         if(cur_node != root){
55acc1             cur_node = cur_node->suffix;
62ae69             g = j - cur_dist;
cbb184         }
2954e9         else
26a5eb             g = i + 1;
d41d8c         // Initially cur_node points to node
d41d8c         // [i+1 ... g-1]
7b2f26         while(g < j and g + cur_node->next(s[g])->len()
<= j){
a622cf             cur_node= cur_node->next(s[g]);
97642d             g += cur_node->len();
cbb184         }
d41d8c         // Case where suffix link was found
9adab0         if(g == j){
3febd7             need_suffix = NULL;
63a879             mid->suffix = cur_node;
bb6442             cur_dist = cur_node->len() - 1;
cbb184         }
d41d8c         // Case where suffix link doesnt exists yet
4e6b83         else{
4d6242             need_suffix = mid;
a622cf             cur_node = cur_node->next(s[g]);
fe9ad2             cur_dist = j - g;
cbb184         }
cbb184     }
b2c239     i += 1;
cbb184 }
cbb184 }
cbb184 }
d41d8c ~suffix_tree(){
1b4230     delete root;
cd7ff3 }
cbb184
d41d8c char get_char(node *cur, int ind){
837589

```

```

49bcec         return s[cur->left + ind];
cbb184     }
d41d8c
d41d8c     //Optional
18f27e     bool verify_substring(string sub){
f85522         node *cur_node = root;
1160eb         int cur_dist = -1;
d41d8c
2bf421         for(char c : sub){
38c748             if(cur_dist < cur_node->len() - 1){
d7031d                 if(get_char(cur_node, cur_dist + 1) != c)
d1fe4d                     return false;
0eb3d3                 cur_dist++;
cbb184             }
4e6b83             else{
8cdd64                 if(cur_node->next(c) == NULL)
d1fe4d                     return false;
5666ba                 cur_node = cur_node->next(c);
6c785e                 cur_dist = 0;
cbb184             }
cbb184         }
d41d8c
8a6c14         return true;
cbb184     }
d41d8c
d41d8c     //Onlu for debugging
b29999     void print(node* cur_node = NULL){
684b03         if(cur_node == NULL)
7b39bc             cur_node = root;
a53190         printf("node %d %d [", cur_node->left, cur_node->right);
1d7add         for(int i = cur_node->left; i <= cur_node->right; i++)
e6b012             printf("%c", s[i]);
edddff         printf("] entra\n");
d41d8c
289aef         for(auto el : cur_node->next_node){
316932             print(el.second);
cbb184         }
d41d8c
a53190         printf("node %d %d [", cur_node->left, cur_node->right);
1d7add         for(int i = cur_node->left; i <= cur_node->right; i++)
e6b012             printf("%c", s[i]);
b9a845         printf("] sai\n");
cbb184     }
2145c1 };
d41d8c
Full file hash: 8c7498

```