Definitely Not A Lib

Leonardo Valente Nascimento

12 de setembro de 2025

Índice

1 data_structures **2**
1.1 Bit.hpp . . . 2
1.2 CartesianTree.hpp . . . 3
1.3 DynamicCht.hpp . . . 3
1.4 IndexedSet.hpp . . . 4
1.5 OffsetVector.hpp . . . 5
1.6 Pareto.hpp . . . 5
1.7 SegtreeIterative.hpp . . . 6
1.8 SegtreeLazy.hpp . . . 7
1.9 SegtreePersistent.hpp . . . 8
1.10 SparseTable.hpp . . . 10
1.11 SqrtDecomp.hpp . . . 10

2 utility **11**
2.1 Checker.cpp . . . 11
2.2 Gen.cpp . . . 12
2.3 Hash.sh . . . 12
2.4 HashFile.sh . . . 12
2.5 Pragmas.hpp . . . 13
2.6 Rng.hpp . . . 13
2.7 Stress.sh . . . 13
2.8 StressChecker.sh . . . 13
2.9 Template.cpp . . . 14
2.10 TemplateBig.cpp . . . 14

3 math **15**
3.1 BasicCombi.hpp . . . 15
3.2 BerlekampMassey.hpp . . . 15
3.3 BigInt.hpp . . . 16
3.4 Bigmod.hpp . . . 18
3.5 Binpow.hpp . . . 18
3.6 Division.hpp . . . 19
3.7 ExtendedGcd.hpp . . . 19
3.8 FactoringAndPrimalityTest.hpp . . . 19
3.9 Fft.hpp . . . 20
3.10 Linearrecurrence.hpp . . . 22
3.11 Matrix.hpp . . . 23
3.12 MatrixMulMod.hpp . . . 25

3.13 Modint.hpp 26

3.14 Polynomial.hpp 27

3.15 Sieve.hpp 28

4 string **29**

4.1 AhoCorasik.hpp 29

4.2 HashInterval.hpp 29

4.3 Kmp.hpp 30

4.4 Manacher.hpp 30

4.5 MinRot.hpp 31

4.6 SuffixArray.hpp 31

4.7 SuffixAutomaton.hpp 33

4.8 Trie.hpp 34

4.9 ZFunction.hpp 34

5 geometry **35**

5.1 Point.hpp 35

6 graph **35**

6.1 2sat.hpp 35

6.2 BinaryLift.hpp 36

6.3 BipartiteMatching.hpp 37

6.4 Dinic.hpp 38

6.5 DsuRollback.hpp 39

6.6 DynamicConnectivity.hpp 40

6.7 EdgeColoring.hpp 41

6.8 FunctionalGraph.hpp 42

6.9 Hld.hpp 43

6.10 IncrementalMst.hpp 45

6.11 Isomorphism.hpp 46

6.12 Kthshortestwalk.hpp 48

6.13 Lca.hpp 49

6.14 Lct.hpp 50

6.15 RootedTree.hpp 51

6.16 Scc.hpp 52

6.17 UnionFind.hpp 53

1 data_structures

1.1 Bit.hpp

```
Hash: 321f5b
/*
from https://github.com/defnotmee/definitely-not-a-lib

Usage: BIT(n) -> creates array arr of size n where you can
make point updates and prefix queries (0-indexed!) in O(log(n))

BIT::merge(a, b) -> merges b into element a. By default a+=b.
(must be commutative and associative)

BIT::update(id, x) -> merge(arr[i],x) for every i <= id

BIT::query(id) -> initializes ret = T(), does merge(ret, arr[i])
for every i <= id, returns ret.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

f1 14 template<typename T = ll>
3c 71 struct BIT{
15 67     vector<T> bit;
eb 27     BIT(int n = 0){
ca 0d         bit = vector<T>(n+1);
13 cb     }

fe 4a     static void merge(T& a, T b){
b5 9f         a+=b;
37 cb     }

4e 7e     void update(int id, T x){
9e ab         id++;
21 b8         while(id < bit.size()){
0a 00             merge(bit[id],x);
24 36             id+=id&-id;
a1 cb         }
89 cb     }

98 32     T query(int id){
49 ab         id++;
b6 83         T ret = T();
32 7a         while(id){
ee df             merge(ret,bit[id]);
22 29             id-=id&-id;
1a cb         }
c9 ed         return ret;
87 cb     }
32 21 };
```

1.2 CartesianTree.hpp

```
Hash: 387379
/*
from https://github.com/defnotmee/definitely-not-a-lib

The best cartesian tree.

Given an array v, calculates the following information in O(n):

- fl[i]: biggest j < i such that v[j] <= v[i]. fl[i] = -1 by default
- fr[i]: smallest j > i such that v[j] < v[i]. fr[i] = n by default
- cl[i]: index of the element that minimizes v[j] for fl[i] < j < i. cl
   [i] = i by default
- cr[i]: index of the element that minimizes v[j] for i < j < fr[i]. cr
   [i] = i by default
- pai[i]: parent of i on the cartesian tree, that is, in the tree where
    i has edges to cl[i] and cr[i]. -1 by default.

In case there are repeated elements, the ones with lowest index will be
    closer to the root of the cartesian tree.

Can also take different comparator functions in its template
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

23 bd template<typename T, typename cmp = less<T>>
4e ed struct CarTree{
ac 1a     int n;
58 51     vector<T> v;
7a 4d     vector<T> fl, fr, cl, cr, pai;
5c 88     int root;

79 7c     CarTree(vector<T>& _v) : n(_v.size()), v(_v), fl(n), fr(n),
    cl(n), cr(n), pai(n,-1){
67 60         for(int i = 0; i < n; i++){
16 0c             fl[i] = i-1;
3e 62             cl[i] = cr[i] = i;
33 23             fr[i] = n;

df 2f             int lst = -1;
51 dc             while(fl[i] != -1 && cmp()(v[i], v[fl[i]])){
c3 8e                 lst = fl[i];
18 0d                 fr[fl[i]] = i;
90 ce                 fl[i] = fl[fl[i]];
ed cb             }
f9 7c             if(lst != -1)
53 99                 cl[i] = lst, pai[lst] = i;
3c f7             if(fl[i] != -1)
63 e8                 cr[fl[i]] = i, pai[i] = fl[i];
44 cb         }

07 83         root = min_element(all(pai))-pai.begin();
92 cb     }
38 21 };
```

1.3 DynamicCht.hpp

```
Hash: 09bf62
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * based on https://github.com/kth-competitive-programming/kactl/blob/
    main/content/data-structures/LineContainer.h
 *
 * Implements a data structure where you can insert functions of the
    form
 * f(x) = ax+b and query the maxmimum/minimum value of f(x)
 *
 * Usage: declare CHT<1> if you want to find maximum f(x) queries, and
 * CHT<-1> if you want minimum f(x) queries.
 *
 * O(log(n)) amortized insertion and O(log(n)) queries
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

aa 73 using line = array<ll,2>;

// mult = 1 for maximum, mult = -1 for minimum
27 dc template<ll mult = 1>
f8 c0 class CHT{
b0 d3     struct poss{
70 54         line l;
a4 50         mutable ll maxx;

67 30         bool operator<(ll x) const {
ab 3b             return maxx < x;
6a cb         }
bd d4         bool operator<(poss o) const {
ca 3f             return l < o.l;
c3 cb         }
f2 21     };

          // if x can be double, change this to -INFINITY
c0 fd     static const ll inf = LLONG_MAX;

          // if x can be double, change this to a/b
2a fd     ll div_floor(ll a, ll b){
73 60         return a/b-(a%b!=0 && (a^b)<0);
cf cb     }

57 5d     multiset<poss,less<>> s;
          // assuming l1 <= l2, finds smallest x such that l1(x) <= l2(
             x)
35 e4     ll intersect(line l1, line l2){
69 49         ll da = l2[0]-l1[0], db = l1[1]-l2[1];
96 bf         if(da == 0)
fd 3d             return -inf;
75 14         return div_floor(db,da);
bf cb     }

53 67     public:
          // Inserts f(x) = ax*b in the structure
47 d0     void insert(ll a, ll b){
63 43         line l = {a*mult,b*mult};

53 02         auto it = next(s.insert({l}));
34 7c         while(it != s.end() && intersect(l,it->l) >= it->maxx)
bf f6             it = s.erase(it);
dd 42         prev(it)->maxx = it == s.end() ? inf : intersect(l,it->l)
    ;
b2 04         it--;

74 23         if(it!=s.begin()){
5f 18             auto prv = prev(it);
3a 38             ll in = intersect(prv->l, l);
ba 52             if(in > it->maxx){
29 df                 s.erase(it);
e4 50                 return;
54 cb             }
36 1f             prv->maxx = in;
a8 16             while(prv != s.begin() && prev(prv)->maxx >= prv->
    maxx){
11 3d                 s.erase(prv);
9e f4                 prv = prev(it);
49 2a                 prv->maxx = intersect(prv->l,l);
91 cb             }
8e cb         }
a8 cb     }

          // Finds maximum f(x) in the structure if mult = 1 and
             minimum f(x) if mult = -1
26 4a     ll query(ll x){
5b 71         auto [a,b] = s.lower_bound(x)->l;
f2 66         return mult*(a*x+b);
0d cb     }
09 21 };
```

1.4 IndexedSet.hpp

Hash: 461dc5

```
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * TODO: ADD usage
 */
77 77 #include <ext/pb_ds/assoc_container.hpp>
07 30 #include <ext/pb_ds/tree_policy.hpp>

f7 67 template<typename T>
06 a9 using index_set = __gnu_pbds::tree<T, __gnu_pbds::null_type,less<
    T>,
46 2c __gnu_pbds::rb_tree_tag, __gnu_pbds::
    tree_order_statistics_node_update>;
```

1.5 OffsetVector.hpp

```
Hash: 89f92e
/*
from https://github.com/defnotmee/definitely-not-a-lib

Create a vector that can be accessed with indexes from [-n to n-1].
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

26 67 template<typename T>
b9 40 struct offvec{

a4 51     vector<T> v;
75 b7     int offset;

b7 3d     offvec(int n = 0, T def = T()){
92 db         offset = n;
44 ea         v = vector<T>(2*n, def);
fc cb     }

bb 8d     T& operator[](int id){
a8 c8         return v[id+offset];
87 cb     }
89 21 };
```

1.6 Pareto.hpp

```
Hash: ac250d
/*
from https://github.com/defnotmee/definitely-not-a-lib

Maintains a partially ordered set (or pareto front), that is,
a list of pairs (x[i], y[i]) such that if for i < j:
x[i] < x[j] => y[i] < y[j].

In a practical sense, "increasing x is bad but incresing y
is good". You can edit pareto::item::fix to change that.

Can only do insertions. O(logn) per insert.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

2d 5f struct pareto{
e7 a3     struct item{
fb 0b         ll x, y;

12 e9         bool operator<(item c) const {
3e a6             if(x == c.x)
fd 2d                 return y < c.y;
77 86             return x < c.x;
6c cb         }


25 85         inline void fix(){
                  // In case increasing x is good, uncomment this:
                  // x*=-1;

                  // In case increasing y is bad, uncomment this:
                  // y*=-1;
a2 cb         }
99 21     };

ca cd     set<item> s;

6c a1     void insert(ll x, ll y){
16 97         item cur = {x,y};
37 e5         cur.fix();
b7 b3         auto it = s.lower_bound(cur);

ee 23         if(it != s.begin()){
5b 53             auto it2 = it;
b9 af             it2--;

a3 4b             if(it2->y >= cur.y)
b2 50                 return;
8d cb         }

9f 7b         while(it != s.end() && cur.y >= it->y){
45 f6             it = s.erase(it);
ef cb         }

c5 a1         s.insert(cur);
96 cb     }

          // returns best item with x <= max_x,
c3 66     item bsearch(ll max_x){
a3 16         item cur = {max_x,0};
34 e5         cur.fix();
d3 87         cur.x++;
55 af         cur.y = -INFL;
fe b3         auto it = s.lower_bound(cur);
92 01         if(it == s.begin()){
                  // pretends that there is a really bad item that
                     always exists
da 9b             item ret = {INFL,-INFL};
81 1e             ret.fix();
74 ed             return ret;
a0 cb         }
f1 04         it--;
9f ff         item ret = *it;
1f 1e         ret.fix();
1b ed         return ret;

ff cb     }
ac 21 };
```

1.7 SegtreeIterative.hpp

```
Hash: ca8ced
/*
from https://github.com/defnotmee/definitely-not-a-lib

Segtree that does point updates and range queries (by default, point
   set range sum).
The merge operation can be non-commutative.

Implementation based on https://codeforces.com/blog/entry/18051
Different from the implementation on that blog, the range on query is [
   l,r] instead of
[l,r)

Commonly changed parts will be commented.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

// In case you want nodes to be a custom struct:

// uncomment this
      // struct seg {
      //     ll x = 0; // "identity value" of operation
// };

43 fc template<typename seg = ll> // comment this
39 d8 struct SegPoint{

3c e4     int sz;
9a df     vector<seg> tree;

f7 b6     SegPoint(int n = 0): sz(n), tree(2*n){};

55 fe     SegPoint(vector<seg> v){ // O(n) builder
b1 ea         *this = SegPoint(v.size());

f9 51         for(int i = 0; i < sz; i++)
8b 71             tree[i+sz] = v[i];
f6 bc         for(int i = sz-1; i > 0; i--)
93 db             tree[i] = merge(tree[2*i], tree[2*i+1]);
00 cb     }

ab 2c     static seg merge(seg a, seg b){
58 df         return {a+b}; // here is where 2 nodes are merged
1a cb     }
ea 40     void update(int id, seg val){
c6 92         id+=sz;

50 ae         tree[id] = val; // here is where you update a point

b0 77         id>>=1;

33 7a         while(id){
89 da             tree[id] = merge(tree[2*id], tree[2*id+1]);
e0 77                id>>=1;
06 cb            }
4f cb        }

03 0d        seg query(int l, int r){
b4 ed            l += sz;
81 c0            r += sz+1;

26 86            seg retl = seg(), retr = seg(); // must be identity value
     through merge

b7 40            while(l < r){
d4 1f                if(l&1)
8a 06                    retl = merge(retl, tree[l++]);
77 84                if(r&1)
96 b3                    retr = merge(tree[--r], retr);

2d 45                l>>=1;
b1 e9                r>>=1;
71 cb            }

8c 5a            return merge(retl,retr);
fa cb        }

ca 21 };
```

1.8 SegtreeLazy.hpp

```
Hash: a5b495
/*
from https://github.com/defnotmee/definitely-not-a-lib

Segment tree that allows range updates and queries. By default, it
    supports affine transformation
updates and sum queries, but commonly editted parts will be commented.

If a lazy segtree is not needed I recommend going for an
    segtree_iterative.hpp for
speed.

0-indexed by default.

==============================================================================
Declaration: SegTree<type>(size), where type is the datatype that
    represents a node of the segtree
Update: update(l, r, {mult, add}), for l <= i <= r, v[i] = v[i]*mult+
    add
Query: query(l,r), returns seg object equivalent to the sum of all
    values on range [l,r]

*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

0a 38 struct SegTree{
          // Uncomment if you need a custom struct
          // struct seg{
          //     int x = 0; // identity value of the merge operation
          // }
6b 5f     using seg = ll; // <--- comment this if you need a custom
    struct


4b 07     struct lazy{
f7 7d         ll mult = 1, add = 0; // "identity value" of lazy tag

              // Here is where you edit how to propagate the lazy tag
                  for the children
              // of a segtree node
35 ef         void operator+=(const lazy& a){
a1 76             add*=a.mult;
32 d4             mult*=a.mult;
a2 29             add+=a.add;
cd cb         }
7d 21     };
3b 5a     static inline seg null = seg(); // identity element through
    the merge operation

          // Here is where you change how to merge nodes
52 2c     static seg merge(seg a, seg b){
8a 53         return a+b;
61 cb     }

26 df     vector<seg> tree;
f5 2c     vector<lazy> lz;

4e 40     int sz, ql, qr;
1f a5     lazy val;
0f 23     SegTree(int n = 0){
76 43         sz = n;
72 93         tree = vector<seg>(4*n,null);
3d 73         lz = vector<lazy>(4*n);
8b cb     }

d7 30     void build(int id, int l, int r, vector<seg> & v){
49 89         if(l == r){
63 62             tree[id] = v[l];
51 50             return;
71 cb         }

37 08         const int e = id*2+1, d = id*2+2, m = (l+r)>>1;

01 f7         build(e,l,m,v);
c6 30         build(d,m+1,r,v);
bd 72         tree[id] = merge(tree[e],tree[d]);

8f cb     }

95 2b     SegTree(vector<seg> v){ // O(n) builder
c9 46         *this = SegTree(v.size());
56 49         build(0,0,sz-1,v);
58 cb     }

8d 8d     void refresh(int id, int l, int r){
cd 57         if(l != r){
15 08             const int e = id*2+1, d = id*2+2, m = (l+r)>>1;

49 c0             lz[e]+=lz[id];
6f b6             lz[d]+=lz[id];
4a cb         }

              // Here is where you update the value of the current node
                   based on the lazy tag
f8 d2         tree[id] = tree[id]*lz[id].mult+lz[id].add*(r-l+1);
e7 b0         lz[id] = lazy();
28 cb     }

ab 51     void update(int l, int r, lazy x){
e0 40         ql = l, qr = r, val = x;
54 8b         upd(0,0,sz-1);
d1 cb     }

93 0d     seg query(int l, int r){
ae e2         ql = l, qr = r;
b1 b0         return qry(0,0,sz-1);
6a cb     }

e2 bf     private:

12 bf     void upd(int id, int l, int r){
ce a7         refresh(id,l,r);

76 ce         if(ql <= l && r <= qr){
08 3f             lz[id] += val;
b2 a7             refresh(id,l,r);
c7 50             return;
f3 cb         }
fb 87         if(ql > r || l > qr)
e1 50             return;

3a 08         const int e = id*2+1, d = id*2+2, m = (l+r)>>1;

89 b7         upd(e,l,m);
e5 ad         upd(d,m+1,r);

33 72         tree[id] = merge(tree[e], tree[d]);
76 cb     }

4c 31     seg qry(int id, int l, int r){
f6 a7         refresh(id,l,r);

d1 43         if(ql <= l && r <= qr)
d6 c9             return tree[id];

5e 87         if(ql > r || l > qr)
fc 54             return null;

d1 08         const int e = id*2+1, d = id*2+2, m = (l+r)>>1;
5d c3         return merge(qry(e,l,m), qry(d,m+1,r));
0b cb     }
a5 21 };
```

1.9 SegtreePersistent.hpp

```
Hash: 126ce7
/*
from https://github.com/defnotmee/definitely-not-a-lib

Persistent Segment Tree with point updates. By default, does point set
    and range sum
```

```
To create a segtree use PSegTree<type>(min_coord, max_coord).
You can effectively copy a segtree in O(1) by just copying a PSegTree
    instance.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif


// Bump allocator for extra performance:

// static char buf[450 << 20]; // by default can store 4.7e8 bytes
      // void* operator new(size_t s) {
      //    static size_t i = sizeof buf;
      //    assert(s < i);
      //    return (void*)&buf[i -= s];
// }
// void operator delete(void*) {}

// implementation above from https://github.com/kth-competitive-
    programming/kactl/blob/main/content/various/BumpAllocator.h


// Uncomment if you need a custom struct.
      // struct seg{

// };

43 fc template<typename seg = ll>
80 bf struct Node{
92 c5     Node(seg x = null) : x(x){}

          // identity value of element through merge operation
46 5a     static inline seg null = seg();

1b c7     seg x = null;
78 f4     Node* e = nullptr, *d = nullptr;

1a 2c     static seg merge(seg a, seg b){
8a 53         return a+b;
52 cb     }

1e fb     void refresh(){
ce ae         if(!e)
cf 9a             e = new Node(), d = new Node();
5a cb     }

37 2e     void update(ll l, ll r, ll q, seg val){
36 89         if(l == r){
91 d1             x = val;
0a 50             return;
9d cb         }

dd 13         refresh();

d8 0f         ll m = (l+r)>>1;

a1 27         if(q <= m)
80 37             (e = new Node(*e))->update(l,m,q,val);
1b a9         else (d = new Node(*d))->update(m+1,r,q,val);

81 6a         x = merge(e->x, d->x);
25 cb     }

06 04     seg query(ll l, ll r, ll ql, ll qr){


40 ce         if(ql <= l && r <= qr){
81 ea             return x;
58 cb         }
ba 87         if(ql > r || l > qr)
db 54             return null;

ac 13         refresh();

d3 0f         ll m = (l+r)>>1;
ca cc         return merge(e->query(l,m,ql,qr), d->query(m+1,r,ql,qr));
ca cb     }

21 21 };

e5 fc template<typename seg = ll>
92 e0 struct PSegTree{

34 f8     ll l, r;
ed c1     Node<seg>* head;

91 cd     PSegTree(ll l = 0, ll r = 0) : l(l), r(r), head(new Node<seg
    >()){}

be 00     seg query(ll ql, ll qr){
61 57         return head->query(l,r,ql,qr);
a1 cb     }

ad 65     void update(ll q, seg val){
8b 6c         (head = new Node<seg>(*head))->update(l,r,q,val);
ba cb     }
12 21 };
```

1.10 SparseTable.hpp

```
Hash: dec367
/*
from https://github.com/defnotmee/definitely-not-a-lib

With O(nlog(n)) pre-processing, creates a data structure that
answers minimum range queries (RMQ) in O(1). Can be modified
to work with any indempotent function.


*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

95 93 template<typename T = int>
a1 76 struct RMQ{
a8 1a     int n;
79 21     vector<vector<T>> sp;

e4 96     RMQ(vector<T> v) : n(v.size()), sp(int(log2(n))+1, vector<T>(
   n)){
fc f7         sp[0] = v;

96 a7         for(int i = 1; i < sp.size(); i++)
ee 06             for(int j = 0; j + (1<<i) <= n; j++)
c5 7d                 sp[i][j] = merge(sp[i-1][j], sp[i-1][j+(1<<i-1)])
    ;

1d cb     }

6e b6     static T merge(T a, T b){
1a 23         return min(a,b);
86 cb     }

76 b7     T query(int l, int r){ // must be called with l <= r
fd 1e         int logg = log2(r-l+1);
b8 e9         return merge(sp[logg][l], sp[logg][r-(1<<logg)+1]);
de cb     }
de 21 };
```

1.11 SqrtDecomp.hpp

```
Hash: 3d0d8b
/*
from https://github.com/defnotmee/definitely-not-a-lib

Divides an array into blocks of sqrt. In this case,
its doing range addition update and range maximum query.

TODO: clean code, make it more general
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

76 51 const int LEN = 400;

7c 14 template<typename T = ll>
41 ff struct decomp{
11 3d     vector<T> elem;
de af     vector<T> block, lz;

69 99     decomp(int n = 0){
fc 56         elem = vector<T>(n);
a1 af         block = vector<T>((n+LEN-1)/LEN);
9e 4e         lz = vector<T>((n+LEN-1)/LEN);
e1 cb     }

d1 57     void reconstruct(int bid){
82 e0         block[bid] = 0;
c4 a0         for(int i = bid*LEN; i < min(int(elem.size()), (bid+1)*
   LEN); i++){
72 6d             block[bid] = max(block[bid], elem[i]);
32 cb         }
a3 e6         block[bid]+=lz[bid];
06 cb     }

88 41     void update(int l, int r, T x){
32 9a         int bl = l/LEN+1, br = r/LEN;

18 16         if(bl >= br){
```

```
42 24           for(int i = l; i <= r; i++)
89 0f               elem[i]+=x;

13 76           reconstruct(br);
45 5c           if(bl-1 != br)
63 06               reconstruct(bl-1);
08 9d       } else {
7b 50           for(int i = l; i < bl*LEN; i++)
ac 0f               elem[i]+=x;
cc 37           for(int i = bl; i < br; i++)
3c bb               lz[i]+=x, block[i]+=x;
d4 69           for(int i = br*LEN; i <= r; i++)
e7 0f               elem[i]+=x;

21 06           reconstruct(bl-1);
62 76           reconstruct(br);
ee cb       }
d7 cb   }

45 b7   T query(int l, int r){
87 9a       int bl = l/LEN+1, br = r/LEN;
84 83       T ret = T();

35 16       if(bl >= br){
f8 24           for(int i = l; i <= r; i++)
06 13               ret = max(ret,elem[i]+lz[i/LEN]);
d5 9d       } else {
f3 50           for(int i = l; i < bl*LEN; i++)
b8 1f               ret = max(ret,elem[i]+lz[bl-1]);
fa 37           for(int i = bl; i < br; i++)
cc cb               ret = max(ret,block[i]);
70 69           for(int i = br*LEN; i <= r; i++)
56 66               ret = max(ret,elem[i]+lz[br]);
62 cb       }
11 ed       return ret;
57 cb   }
3d 21 };
```

2 utility

2.1 Checker.cpp

```
Hash: 8101f6
/*
from https://github.com/defnotmee/definitely-not-a-lib

Example checker for stress_checker.sh
*/

2b 2b #include<bits/stdc++.h>
64 01 #define all(x) begin(x), end(x)
0d df #define ff first
d9 a9 #define ss second
80 92 #define O_O
6d ca using namespace std;
af 67 template <typename T>
a3 7f using bstring = basic_string<T>;
ba 67 template <typename T>
d9 f2 using matrix = vector<vector<T>>;
df 34 typedef unsigned int uint;
78 f4 typedef unsigned long long ull;
2e ad typedef long long ll;
96 ff typedef pair<int,int> pii;
0f 0d typedef pair<ll,ll> pll;
91 6d typedef double dbl;
fd 68 typedef long double dbll;
ec 5a const ll INFL = 4e18+25;
e0 dc const int INF = 1e9+42;
3f 2a const double EPS = 1e-7;
22 f2 const int MOD = (1<<23)*17*7 + 1; // 998244353
93 d1 const int RANDOM = chrono::high_resolution_clock::now().
   time_since_epoch().count();
cb fc const int MAXN = 1e6+1;


77 e8 int main(){

2c 8b     ios_base::sync_with_stdio(false);
4c 00     cin.tie(nullptr);

b7 2b     ifstream ccin("input");

6a ba     int a, b;
1b 0a     cin >> a >> b;

34 c7     assert(a < b);

20 0a     int c;
ed d1     ccin >> c;

91 6f     if(a*b != c){
```

```
43 2d           cout << "a*b is not c\n";
c6 6a           return 1;
4d cb       }

1f f3       cout << "ok\n";

35 bb       return 0;

81 cb }
```

2.2 Gen.cpp

```
Hash: 59c40c
/*
from https://github.com/defnotmee/definitely-not-a-lib

Example of a generator for stress.sh
*/

2b 2b #include<bits/stdc++.h>
64 01 #define all(x) begin(x), end(x)
0d df #define ff first
d9 a9 #define ss second
80 92 #define O_O
6d ca using namespace std;
af 67 template <typename T>
a3 7f using bstring = basic_string<T>;
ba 67 template <typename T>
d9 f2 using matrix = vector<vector<T>>;
df 34 typedef unsigned int uint;
78 f4 typedef unsigned long long ull;
2e ad typedef long long ll;
96 ff typedef pair<int,int> pii;
0f 0d typedef pair<ll,ll> pll;
91 6d typedef double dbl;
fd 68 typedef long double dbll;
ec 5a const ll INFL = 4e18+25;
e0 dc const int INF = 1e9+42;
3f 2a const double EPS = 1e-7;
22 f2 const int MOD = (1<<23)*17*7 + 1; // 998244353
93 d1 const int RANDOM = chrono::high_resolution_clock::now().
    time_since_epoch().count();
cb fc const int MAXN = 1e6+1;

dc 01 mt19937 rng;

54 6b int range(int l, int r){
df 0d     return uniform_int_distribution<>(l,r)(rng);
d4 cb }

4b 6f int main(int argc, char ** argv){

e8 8b     ios_base::sync_with_stdio(false);
aa 00     cin.tie(nullptr);

9e 83     rng.seed(atoi(argv[1]));
42 d8     int n = range(1,5), m = range(1,5), k = range(0,n*m);

d7 18     cout << n << ' ' << m << ' ' << k << endl;

6c bb     return 0;

59 cb }
```

2.3 Hash.sh

```
Hash: d78ff6
d4 d4 # From https://github.com/tdas0/lib/blob/master/library/contest/
    hash.sh

d4 d4 # Usage: bash hash.sh arquivo.cpp l1 l2
d4 d4 # Finds hash of file from line l1 to line l2

d7 d7 sed -n $2','$3' p' $1 | sed '/^#w/d' | cpp -dD -P -fpreprocessed
    | tr -d '[:space:]' | md5sum | cut -c-6
```

2.4 HashFile.sh

```
Hash: c85258

d4 d4 # From https://github.com/tdas0/lib/blob/master/library/contest/
    gethash.sh

d4 d4 # Gets hash of file to compare to the pdf of the library
d4 d4 # Usage: bash gethash.sh arquivo.cpp

f5 f5 echo "" > pref.txt
5e 95 while IFS= read -r l; do
ca e8   echo "$l" >> pref.txt
db 65   echo "$l" > line.txt
```

```
22 d4   hp=$(echo $(bash hash.sh pref.txt 1 1000) | cut -c-2)
b5 fd   hl=$(echo $(bash hash.sh line.txt 1 1000) | cut -c-2)
d9 ae   echo -e "$hp $hl $1"
c8 65 done < "$1"
```

2.5 Pragmas.hpp

```
Hash: 5e11de
/*
from https://github.com/defnotmee/definitely-not-a-lib

Useful pragmas from nor's blog: https://codeforces.com/blog/entry/96344
*/

88 88 #pragma GCC optimize("O3,unroll-loops")
5a 82 #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

// Pragma for randomized solutions by magnus.hegdahl

5e a0 #pragma VODOO magic("Please work this time")
```

2.6 Rng.hpp

```
Hash: 263a2f
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Basic RNG functionality. High quality.
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

a5 d5 const int SEED = chrono::high_resolution_clock::now().
    time_since_epoch().count();
74 57 mt19937 rng(SEED);

// Returns random integer from l to r.
d4 6b int range(int l, int r){
25 d2     return uniform_int_distribution<int>(l,r)(rng);
26 cb }
```

2.7 Stress.sh

```
Hash: 687d34
d4 d4 #!/usr/bin/env bash

d4 d4 # Based on tyrowhiz's template.
d4 d4 # Usage: bash stress.sh wrong_sol bruteforce generator
    test_case_count

d4 d4 # wrong_sol, bruteforce and generator must be WITHOUT extensions

07 07 make $1
ab d3 make $2
ee 49 make $3

42 07 for ((testNum=0;testNum<$4;testNum++))
45 d4 do
08 2c     ./$3 $testNum > input
a0 7e     ./$2 < input > outSlow
17 a2     ./$1 < input > outWrong
d0 ac     if !(diff -b "outWrong" "outSlow")
fd 0e     then
7e 75         echo "Error found!"
3a 62         echo "Input:"
7b c5         cat input
49 98         echo "Wrong Output:"
60 a2         cat outWrong
59 97         echo "Slow Output:"
94 a8         cat outSlow
cf f2         exit
16 75     fi
1c d6     echo Passed Test:$testNum
7e 6b done
68 1b echo Passed $4 tests
```

2.8 StressChecker.sh

```
Hash: 55d9cc
d4 d4 #!/usr/bin/env bash

d4 d4 # Based on tyrowhiz's template.
d4 d4 # Usage: bash stress.sh wrong_sol checker generator
    test_case_count
d4 d4 # - checker should return 0 if the solution is correct and
    anything else otherwise
```

```
d4 d4 # - if the checker needs the original input, it will be on a file
     named input and
d4 d4 # you could use something like "ifstream ccin("input"); ccin >>
    something" to read it

d4 d4 # wrong_sol, checker and generator must be WITHOUT extensions

07 07 make $1
ab d3 make $2
ee 49 make $3

42 07 for ((testNum=0;testNum<$4;testNum++))
45 d4 do
08 2c     ./$3 $testNum > input
d3 fd     ./$1 < input > out

5b 5e     if !(./$2 < out > veredict)
f7 0e     then
1d 75         echo "Error found!"
cb 62         echo "Input:"
7d c5         cat input
63 37         echo "Output:"
94 6b         cat out
bc 0b         echo "Checker Veredict:"
af 56         cat veredict
4a f2         exit
ee 75     fi
87 d6     echo Passed Test:$testNum
82 6b done
55 1b echo Passed $4 tests
```

2.9 Template.cpp

```
Hash: 617e5f
/*
by Leonardo Valente Nascimento
*/

2b 2b #include<bits/stdc++.h>
64 01 #define all(x) begin(x), end(x)
0d df #define ff first
d9 a9 #define ss second
80 92 #define O_O
6d ca using namespace std;
a1 90 using ll = long long;
56 67 template<typename T>
e0 f2 using matrix = vector<vector<T>>;
37 a0 using pii = pair<int,int>;
e2 f3 using ull = unsigned long long;
1a 4e const int MOD = 998244353;
5e dc const int INF = 1e9+42;
47 e3 const ll INFL = 2e18;
4a 9c const int MAXN = 1e6+10;

81 e8 int main(){

94 21     cin.tie(0)->sync_with_stdio(0);


ec bb     return 0;

61 cb }
```

2.10 TemplateBig.cpp

```
Hash: 55ce33
/*
by Leonardo Valente Nascimento

Big version of template.cpp (used for when copypasting is allowed)
*/

2b 2b #include<bits/stdc++.h>
64 01 #define all(x) begin(x), end(x)
0d df #define ff first
d9 a9 #define ss second
80 92 #define O_O
6d ca using namespace std;
af 67 template <typename T>
a3 7f using bstring = basic_string<T>;
ba 67 template <typename T>
d9 f2 using matrix = vector<vector<T>>;
2f 90 using ll = long long;
f9 ae using uint = unsigned int;
e4 f3 using ull = unsigned long long;
5c e5 using dbl = double;
ed 93 using dbll = long double;
c9 79 using ci = complex<int>;
18 db using cl = complex<ll>;
94 a0 using pii = pair<int,int>;
e8 bb using pll = pair<ll,ll>;
25 5a const ll INFL = 4e18+25;
```

```
a2 dc const int INF = 1e9+42;
2c 2a const double EPS = 1e-7;
80 f2 const int MOD = (1<<23)*17*7 + 1; // 998244353
d1 d5 const int SEED = chrono::high_resolution_clock::now().
    time_since_epoch().count();
d4 fc const int MAXN = 1e6+1;
df 57 mt19937 rng(SEED);

fc e8 int main(){

5c 8b     ios_base::sync_with_stdio(false);
cc 00     cin.tie(nullptr);



ed bb     return 0;

55 cb }
```

3 math

3.1 BasicCombi.hpp

```
Hash: 6fe4d8
/*
from https://github.com/defnotmee/definitely-not-a-lib

Calculates factorials and binomials modulo p for all
numbers from 0 to n-1. By default creates the struct
for n = MAXN and names it combi.

Idea for O(n) inverse of each number from this blog:
https://codeforces.com/blog/entry/83075
*/


d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
91 53 #include"modint.hpp"
25 f2 #endif

22 86 template<ull M>
b6 d8 struct Combi{
          // note that inv[0] = 1 in this impl
bb a8     vector<ll> fac, inv, invfac;

f2 bc     Combi(int n = MAXN){
bf 6b         fac = inv = invfac = vector<ll>(n,1);

4b 9f         for(int i = 2; i < n; i++){
58 e0             fac[i] = fac[i-1]*i%M;
77 ea             inv[i] = inv[M%i]*(M-M/i)%M;
bc 3b             invfac[i] = invfac[i-1]*inv[i]%M;
5f cb         }
ac cb     }

17 ab     ll choose(int n, int k){
de 37         if(n < k)
28 bb             return 0;
00 76         return fac[n]*invfac[k]%M*invfac[n-k]%M;
f5 cb     }
02 21 };

6f b7 Combi<MOD> c;
```

3.2 BerlekampMassey.hpp

```
Hash: 616591
/*
from https://github.com/defnotmee/definitely-not-a-lib

Based on https://mzhang2021.github.io/cp-blog/berlekamp-massey/

Finds coefficients of the shortest linear recurrence that
describes a given sequence in O(n^2). If the original linear recurrence
is of order k, 2k terms will be necessary to pinpoint it exactly.

Returns a sequence c0c1c2...ck where if the sequence is s0s1s2...sn
it will hold that si = sum(c(j)*s(i-j-1)) for i > k.
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
91 f2 #endif

42 67 template<typename T>
55 f7 vector<T> berlekamp_massey(vector<T> s){
de 59     vector<T> c, oc;
d4 e8     T ldelta = 0;
4d d2     int f = 0;
ec 1c     for(int i = 0; i < s.size(); i++){
62 fa         T delta = s[i];

52 22         for(int j = 0; j < c.size(); j++){
4f d7             delta-=c[j]*s[i-j-1];
80 cb         }

fb 84         if(delta == 0)
cc 5e             continue;
28 22         if(ldelta == 0){
4b b3             c = vector<T>(i+1);
34 ab             f = i;
7c a4             ldelta = delta;
96 5e             continue;
cf cb         }

4f 4e         vector<T> maybe = c;
85 fd         vector<T> d = oc;

10 82         for(auto& i : d)
d1 37             i*=-1;

35 af         d.insert(d.begin(),T(1));

70 7b         c.resize(max(c.size(), d.size()+i-f-1));

00 71         T mult = delta/ldelta;
f3 1b         for(int j = 0; j < d.size(); j++)
62 d1             c[j+i-f-1]+=d[j]*mult;

3d c3         if(i+oc.size() > f+maybe.size()){
a5 60             oc = maybe;
62 ab             f = i;
ab a4             ldelta = delta;
17 cb         }
46 cb     }

14 80     return c;

61 cb }
```

3.3 BigInt.hpp

```
Hash: 5afbdf
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Implements arithmetic operations with arbitrary precision integers.
 *
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

78 45 template<ll B = 10>
51 97 struct Big{
d3 e9     static_assert(B >= 1);

fa 6b     vector<ll> v;
c3 75     bool minus = 0;

cb f3     Big(ll x = 0) {
6d 30         if(x < 0)
2b 4a             minus = 1, x*=-1;
96 f4         while(x){
24 c0             v.push_back(x%B);
6a 57             x/=B;
88 cb         }
20 95         if(v.empty())
73 20             v.push_back(0);
67 cb     }

7b a0     void trim(){
81 0c         while(v.size() > 1 && v.back() == 0)
57 de             v.pop_back();
aa cb     }

a2 3d     Big(string s){
4b 00         static_assert(B <= 10);
65 73         reverse(all(s));
8f 33         if(s.back() == '-')
5d cb             minus = 1, s.pop_back();
8b 72         for(int i = 0; i < s.size(); i++)
d9 cb             v.push_back(s[i]-'0');
11 0e         trim();
c4 cb     }

          /**
           * Converts an Big Integer from base A to base B
           *
           * ASSUMES THAT A IS A POWER OF B OR VICE VERSA
           *
```

```
         * Useful for speeding up operations (for example,
         * doing multiplication in base 10^9 is 81x faster than
         * base 10)
         * */
46 66   template<ll A>
8a d9   Big(Big<A> x){
8b db       minus = x.minus;
bc be       if(A == B){
30 44           v = x.v;
fc 50           return;
78 cb       }
01 7f       if(A < B){
c5 20           v.push_back(0);
f7 01           ll cur = 1;
15 26           for(ll i = 0; i < x.v.size(); i++){
47 85               v.back()+=cur*x.v[i];
7b 72               cur*=A;
1f d3               if(cur == B)
79 33                   cur = 1, v.push_back(0);
5e cb           }
51 0e           trim();
6e 9d       } else {
e8 31           for(int i = 0; i < x.v.size(); i++){
22 01               ll cur = 1;
4c fc               while(cur != A){
95 3d                   v.push_back(x.v[i]%B);
6e 35                   x.v[i]/=B;
23 63                   cur*=B;
89 cb               }
13 cb           }
8c 0e           trim();
cc cb       }
00 cb   }

7c cd   void extend(int sz){
74 82       if(v.size() < sz)
e7 ea           v.resize(sz);
8d cb   }

32 4b   Big& operator+=(Big& o){
4b 8f       if(minus == o.minus){
79 c9           extend(o.v.size());
72 73           o.extend(v.size());
c4 20           v.push_back(0);
d2 ae           for(int i = 0; i < o.v.size(); i++){
c0 f2               v[i]+=o.v[i];
0c 82               v[i+1]+=v[i]/B;
a2 f0               v[i]%=B;
8d cb           }
f2 0e           trim();
e1 cb       }
cd 35       return *this;
3c 21   };

c1 7b   Big& operator-=(Big o){
51 c4       o.minus^=1;
ac 64       return *this += o;
c3 cb   }

4a bc   Big operator+(Big o){
59 59       Big x;
1d 9e       return x+=o;
a8 cb   }

95 64   Big operator-(Big o){
b6 59       Big x;
52 45       return x-=o;
0e cb   }

9b de   Big operator*(Big o) const {
40 6a       Big ret;
42 e1       ret.extend(v.size()+o.v.size());
ad f1       for(int i = 0; i < v.size(); i++){
e4 1e           for(int j = 0; j < o.v.size(); j++){
2a b0               ret.v[i+j]+=v[i]*o.v[j];
b7 84               ret.v[i+j+1]+=ret.v[i+j]/B;
15 02               ret.v[i+j]%=B;
5f cb           }
44 cb       }
32 9b       ret.trim();
b4 ed       return ret;
4f cb   }

56 fe   friend istream& operator>>(istream& in, Big& x){
2e ac       string s;
c5 bd       in >> s;
85 53       x = Big(s);
fa 09       return in;
43 cb   }

a2 ce   friend ostream& operator<< (ostream& out, Big& x){
ff 03       x.trim();
f6 0d       if(x.minus)
ef 82           out << '-';
2b 13         for(int i = int(x.v.size())-1; i >= 0; i--){
65 90             out << x.v[i];
01 cb         }
b1 fe         return out;
09 cb     }
5a 21 };
```

3.4 Bigmod.hpp

```
Hash: 5902df
/*
from https://github.com/defnotmee/definitely-not-a-lib

Implements modulo operations for big MOD. Important for
number theory stuff.
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
91 f2 #endif

bf 55 inline ull modadd(ull a, ull b, ull m){
2b 47     return min(a+b,a+b-m);
26 cb }

34 8f inline ull modsub(ull a, ull b, ull m){
9f ec     return min(a-b,a-b+m);
94 cb }

// stolen from https://github.com/kth-competitive-programming/kactl/
   blob/main/content/number-theory/ModMulLL.h
// works for a,b,m < 7.2e18
e7 f8 inline ull modmul(ull a, ull b, ull m){
2d 8d     ull ret = a*b - m*ull((long double)(a)*b/m);
af 9d     return min({ret,ret+m,ret-m});
e4 cb }

1f b3 ull inverse(ull a, ull m){
4c 5c     ull x = a, y = m;
85 be     complex<ull> cx = {1,0}, cy = {0,1};

63 f4     while(x){
bf 0e         ull curdiv = y/x;
97 61         y-=curdiv*x;
bc eb         cy-=curdiv*cx;
c0 0e         swap(cx, cy);
de 9d         swap(x, y);
71 cb     }

2c 50     return min(cy.real()+m, cy.real());
ad cb }

0b 0d ull divmul(ull a, ull b, ull m){
1e a5     return modmul(a,inverse(b,m),m);
d6 cb }

e6 5b ull power(ull in, ull exp, ull m){
34 cc     ull ret = 1;
89 fb     while(exp){
03 87         if(exp&1)
27 a0             ret = modmul(ret,in,m);
17 3d         in = modmul(in,in,m);
30 ef         exp>>=1;
60 cb     }
53 ed     return ret;
59 cb }
```

3.5 Binpow.hpp

```
Hash: 984c7c
/*
from https://github.com/defnotmee/definitely-not-a-lib

Does binary exponentation. By default can handle exponents
< 2^63, for more you just edit the constants in the function.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

26 67 template<typename T>
38 0d T power(T cur, ll exp){
aa 7b     T ret = T(1); // works for modint.cpp by default

8d fb     while(exp){
07 87         if(exp&1)
2b 27             ret*=cur;
d7 73         cur*=cur;
c8 ef         exp>>=1;
6f cb     }
fa ed     return ret;
98 cb }
```

3.6 Division.hpp

```
Hash: 82cbc8
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Integer division with ceil and floor that works for
 * potentially negative numbers
 */

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

3e fd ll div_floor(ll a, ll b){
fd 60     return a/b-(a%b!=0 && (a^b)<0);
2a cb }
27 ab ll div_ceil(ll a, ll b){
a0 61     return a/b+(a%b!=0 && (a^b)>0);
82 cb }
```

3.7 ExtendedGcd.hpp

```
Hash: d571a8
/*
from https://github.com/defnotmee/definitely-not-a-lib

based on https://cp-algorithms.com/algebra/extended-euclid-algorithm.
    html

Given 2 numbers x, y, returns {gcd(x,y), alpha, beta} such that alpha*x
     + beta*y = gcd(x,y)
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
91 f2 #endif

e4 99 auto gcd_ex(ll x, ll y){
c6 ca     complex<ll> cx = {1,0}, cy = {0,1};

49 f4     while(x){
73 41         ll curdiv = y/x;

58 61         y-=curdiv*x;
8d 4e         cy-=cx*curdiv;

b5 0e         swap(cx, cy);
d2 9d         swap(x,y);
70 cb     }

88 bf     struct res{ll gcd, alpha, beta;};
5d ed     return res{y,cy.real(),cy.imag()};
d5 cb }
```

3.8 FactoringAndPrimalityTest.hpp

```
Hash: faecfd
/*
from https://github.com/defnotmee/definitely-not-a-lib
Implements primality check with miller-rabin in O(7logn) and
prime factorization in O(n^(1/4)) with pollard-rho.
Primality checking is [supposedly] deterministic but factoring
is a monte carlo algorithm.
Pollard-rho impl is heavily based on:
https://github.com/kth-competitive-programming/kactl/blob/main/content/
    number-theory/Factor.h
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
4f a7 #include"bigmod.hpp"
b1 f2 #endif

73 87 bool is_prime(ull n){
45 c9     if(n <= 1)
96 d1         return false;

f2 74     ull ctz = __builtin_ctz(n-1);
e4 70     ull d = n>>ctz;

ae 0e     auto primes = {2, 3, 5, 13, 19, 73, 193, 407521, 299210837};
          // all primes up to 37 is a reasonable option too
0b a7     auto bases = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022};

a7 ce     for(ull p : primes)
98 e7         if(n == p)
17 6a             return 1;

aa e4     for(ull base : bases){
c2 0c         ull cur = power(base,d,n);
8d 66         if(cur == 1)
4c 5e             continue;
79 2b         for(int i = 0; i < ctz; i++){
03 56             if(cur == n-1)
da 20                 goto NEXT;
72 1f             cur = modmul(cur,cur,n);
59 cb         }
02 d1         return false;
fb 8f         NEXT:;
4b cb     }

05 8a     return true;

b5 cb }

02 67 template<typename T>
04 cc void pollard(T n, vector<T>& v){
97 e7     if(n == 1)
bc 50         return;
5e a7     if(is_prime(n)){
de 3b         v.push_back(n);
ba 50         return;
1f cb     }

04 dd     static mt19937_64 rng(RANDOM);
bf dc     uniform_int_distribution<T> rg(0,n-1);
bd fb     T c = rg(rng);
ae 64     T x, y;
f5 e8     x = y = rg(rng);

ea ce     auto next = [&](T x){
9d fd         return modadd(modmul(x,x,n),c,n);
a8 21     };

1e a4     T prod = 2;
41 aa     T g = 1;
e3 8f     while((g = gcd(prod,n)) == 1){
43 ac         for(int i = 0; i < 50; i++){
1d 2a             if(x == y)
da 1f                 x = y = rg(rng), c = rg(rng);
b3 53             x = next(x);
b1 1d             y = next(next(y));
41 80             ll cur = modmul(abs(x-y),prod,n);
71 69             if(cur)
ac 27                 prod = cur;
ec cb         }
41 cb     }

57 36     pollard(g,v);
71 6c     pollard(n/g,v);
91 cb }

d7 67 template<typename T>
b2 4f vector<T> factorize(T n, bool sorted = 0){
c9 5a     vector<T> ret;

80 0e     pollard(n,ret);

db 64     if(sorted)
6c 2f         sort(all(ret));

3b ed     return ret;
fa cb }
```

3.9 Fft.hpp

```
Hash: 40a300
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Thanks -is-this-fft- for your blog https://codeforces.com/blog/entry
   /111371
 *
 * References for implementation:
 *
 * https://cp-algorithms.com/algebra/fft.html
 * http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf
 * https://github.com/kth-competitive-programming/kactl/blob/main/
   content/numerical/FastFourierTransform.h
 */
d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
91 f2 #endif

74 06 using cdl = complex<long double>;
4f 23 using cd = complex<double>; // if WA, change this to long double
   and pray
```

```
17 ec void fft(vector<cd>& v, bool inverse = 0){
8d 3d     int n = v.size();
81 77     int lg = log2(n);

24 8a     static vector<cdl> loots;
42 dc     static vector<cd> roots;

1b 27     if(loots.size() < n){
dc cb         loots.resize(n,1);
55 7b         roots.resize(n,1);
03 cb     }


a0 89     for(static int len = 2; len < n; len<<=1){
41 d8         cdl z = polar(1.0l, acos(-1.0l)/len);
fc d8         for(int i = len; i < 2*len; i++){
96 00             roots[i] = loots[i] = loots[i/2] * ((i&1) ? z : 1);
61 cb         }
0d cb     }

4c 80     vector<int> rev(n);

1e 6f     for(int i = 1; i < n; i++){
31 56         rev[i] = (rev[i>>1]>>1)+((i&1)<<lg-1);
28 fc         if(rev[i] > i)
ba 60             swap(v[i],v[rev[i]]);
13 cb     }

07 9b     for(int len = 1; len < n; len<<=1){
1a 27         for(int block = 0; block < n; block+=2*len){
b4 5d             for(int l = block; l < block+len; l++){
23 c4                 cd cur = roots[l-block+len]*v[l+len];
2b d8                 tie(v[l], v[l+len]) =
63 f6                     make_pair(v[l]+cur, v[l]-cur);
e4 cb             }
82 cb         }
e8 cb     }

8d 1f     if(inverse){
9f 83         reverse(1+all(v));
f3 2d         for(auto& i : v)
9b c4             i/=n;
44 cb     }
b6 cb }


dd f1 vector<ll> convolution(vector<ll>& a, vector<ll>& b){
90 43     int n = 1;
35 0c     while(n+1 < a.size()+b.size())
af c1         n<<=1;

39 b0     vector<cd> in(n);

c1 43     for(int i = 0; i < a.size(); i++)
a4 0a         in[i].real(a[i]);
a9 c0     for(int i = 0; i < b.size(); i++)
ae f4         in[i].imag(b[i]);

b0 21     fft(in);

f0 4d     vector<cd> newin(n);

cf 60     for(int i = 0; i < n; i++){
86 d6         int opos = (n-i)&(n-1);
b5 2c         newin[i] = (in[opos]+conj(in[i]))
d7 24         *(in[opos]-conj(in[i]))*cd(0, -0.25/n);
de cb     }

a5 1e     fft(newin);

01 8a     vector<ll> ret(a.size()+b.size()-1);
55 2a     for(int i = 0; i < a.size()+b.size()-1; i++){
74 f6         ret[i] = round(newin[i].real());
8f cb     }

9f ed     return ret;

62 cb }

a2 5e vector<cd> convolution(vector<cd> a, vector<cd> b){
42 f7     int rets = a.size()+b.size()-1;
ac 43     int n = 1;
9c 0c     while(n+1 < a.size()+b.size())
bd c1         n<<=1;

81 ca     a.resize(n), b.resize(n);

c1 0f     fft(a), fft(b);

c0 60     for(int i = 0; i < n; i++){
3e db         a[i]*=b[i];
40 cb     }

2e c3     fft(a,1);
55 68     a.resize(rets);

5f 3f     return a;
36 cb }


bd e3 template<ull M = MOD>
46 b9 vector<ll> convolutionmod(vector<ll>& a, vector<ll>& b){
ab 57     const int len = sqrt(M);
41 43     int n = 1;
3e 0c     while(n+1 < a.size()+b.size())
85 c1         n<<=1;

09 53     vector<cd> ca(n), cb(n);

64 43     for(int i = 0; i < a.size(); i++)
d3 67         ca[i] = cd(a[i]%len, a[i]/len);

57 c0     for(int i = 0; i < b.size(); i++)
4f 61         cb[i] = cd(b[i]%len, b[i]/len);

f0 ec     fft(ca), fft(cb);

25 52     vector<cd> p1(n), p2(n);

5e 60     for(int i = 0; i < n; i++){
39 d6         int opos = (n-i)&(n-1);

              // also inverting for fft inverse
ea b7         p1[i] = (ca[opos]+conj(ca[i]))*cb[opos]*cd(0.5/n);
16 79         p2[i] = (ca[opos]-conj(ca[i]))*cb[opos]*cd(0,-0.5/n);
28 cb     }

3f bb     fft(p1), fft(p2);

88 8a     vector<ll> ret(a.size()+b.size()-1);

29 9c     for(int i = 0; i < ret.size(); i++){
cf df         ll r1 = round(p1[i].real()), i1 = round(p1[i].imag());
0b aa         ll r2 = round(p2[i].real()), i2 = round(p2[i].imag());

4b 0a         ll small = r1%MOD, mid = (i1+r2)%MOD, big = i2%MOD;
01 fd         (ret[i] = small + mid*len + big*len%MOD*len)%=MOD;
38 cb     }

7f ed     return ret;
40 cb }
```

3.10 Linearrecurrence.hpp

```
Hash: d73748
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
a3 97 #include"polynomial.hpp"
6d f2 #endif

1d 67 template<typename T>
c6 08 T get_kth_term(vector<T> s, vector<T> c, ll k){

56 85     Poly<T> charac(c.size()+1);
6a 97     for(int i = 0; i < c.size(); i++)
b2 cc         charac[i] = c[c.size()-i-1]*-1;
5e 5f     charac.p.back() = 1;

7a 95     Poly<T> retp(c.size());
3f f1     retp[0] = 1;
6d 75     Poly<T> mul(c.size());
9c f3     if(c.size() == 1)
a0 17         mul[0] = c[0];
92 7f     else mul[1] = 1;

27 95     while(k){
59 33         if(k&1){
26 13             retp*=mul;
0d ab             retp%=charac;
57 cb         }

5d 64         mul*=mul;
44 d3         mul%=charac;

4c b4         k>>=1;
85 cb     }

c8 ce     T ret = 0;

bd 01     for(int i = 0; i < c.size(); i++){
9b 88         ret+=s[i]*retp[i];
55 cb     }

f6 ed     return ret;
d7 cb }
```

3.11 Matrix.hpp

```
Hash: 48af65
/*
from https://github.com/defnotmee/definitely-not-a-lib

Implements matrices and linear algebra stuff for them.

Includes multiplication, addition, solving system of equation,
finding ranks, etc
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
91 f2 #endif

42 67 template<typename T>
65 bf struct Matrix{

9f 14     int n, m;
f9 e2     valarray<valarray<T>> v;

0a 73     Matrix(int _n, int _m, int id = 0) : n(_n), m(_m), v(valarray
    <T>(m),n) {
7b 9e         if(id){
15 af             for(int i = 0; i < min(n,m); i++)
b9 62                 v[i][i] = 1;
12 cb         }
09 cb     }

37 97     valarray<T>& operator[] (int x){
56 7b         return v[x];
cc cb     }


35 4e     Matrix transpose(){
5b bc         Matrix newv(m,n);

cb 83         for(int i = 0; i < n; i++)
4f a7             for(int j = 0; j < m; j++)
49 06                 newv[j][i] = (*this)[i][j];

b9 b0         return newv;
3d cb     }

f6 58     Matrix operator+(Matrix& b){
a3 50         Matrix ret(*this);
df 2c         return ret.v+=b.v;
3a cb     }

db c6     Matrix& operator+=(Matrix& b){
11 8c         return v += b.v;
7e cb     }

69 7b     Matrix operator*(Matrix b){
35 5b         Matrix ret(n, b.m);

64 83         for(int i = 0; i < n; i++)
b4 a7             for(int j = 0; j < m; j++)
fa bc                 for(int k = 0; k < b.m; k++)
0a 66                     ret[i][k] += v[i][j]*b.v[j][k];

4e ed         return ret;
7a cb     }

b4 80     Matrix& operator*=(Matrix b){
d7 0a         return *this = *this*b;
81 cb     }

ec d6     Matrix power(ll exp){
d2 7b         Matrix in = *this;
2a 01         Matrix ret(n, n, 1);

e2 fb         while(exp){
88 87             if(exp&1)
c3 6c                 ret*=in;
25 f5             in*=in;
23 ef             exp>>=1;
98 cb         }
b4 ed         return ret;
06 cb     }

          /*
          Alters current matrix.

          Does gaussian elimination and puts matrix in
          upper echelon form (possibly reduced).

          Returns the determinant of the square matrix with side equal
              to the number
          of rows of the original matrix.
          */

a1 50     T gaussjordanize(int reduced = 0){
```

```
08 f0           T det = T(1);

ae bd           int line = 0;
b1 6f           for(int col = 0; col < m; col++){

57 e7               int pivot = line;
8f 94               while(pivot < n && v[pivot][col] == T(0))
db 05                   pivot++;

40 ae               if(pivot >= n)
a3 5e                   continue;

15 84               swap(v[line], v[pivot]);

94 b4               if(line != pivot)
43 0f                   det *= T(-1);

d7 01               det*=v[line][line];

d6 a6               v[line]/=T(v[line][col]);

20 0e               if(reduced)
fd 6d                   for(int i = 0; i < line; i++){
2a 7f                       v[i] -= T(v[i][col])*v[line];
89 cb                   }

ee bd               for(int i = line+1; i < n; i++){
e7 7f                   v[i] -= T(v[i][col])*v[line];
ab cb               }

de 64               line++;
20 cb           }

c0 41           return det * (line == n);
7a cb       }

            /*
            When called on any matrix, puts it in reduced row echelon
                form and solves the system of equations
            it represents. In particular, if called on matrix A, finds a
                vector x such that Ax = y

            Returns {possible x, number of solutions (2 if there are
                infinite solutions)}

            In case theres no solution, returns {{},0}
            */
```

```
21 ab       pair<vector<T>,int> solve_system(vector<T> y){

0a 10           Matrix aug(n, m+1);

95 60           for(int i = 0; i < n; i++){
a6 a7               for(int j = 0; j < m; j++)
05 78                   aug[i][j] = v[i][j];
4d 77               aug[i][m] = y[i];
b1 cb           }

b8 b0           aug.gaussjordanize(1);

b1 18           int solcount = n < m ? 2 : 1;

cb 72           vector<T> x(m);

2a 45           for(int i = n-1; i >= 0; i--){
c7 1e               if(i < m && aug[i][i] == T(0))
10 e5                   solcount = 2;

13 e8               int pivot = 0;
95 ca               while(pivot < m && aug[i][pivot] == T(0))
c8 05                   pivot++;

19 41               if(pivot == m){
b4 ff                   if(aug[i][m] != T(0)){
c6 14                       return {{},0};
08 cb                   }
c0 5e                   continue;
0f cb               }

a3 98               x[pivot] = aug[i][m];

a8 c6               for(int j = pivot+1; j < m; j++){
a6 39                   x[pivot]-=x[j]*aug[i][j];
99 cb               }
d6 cb           }

d2 60           for(int i = 0; i < n; i++){
70 a7               for(int j = 0; j < m; j++)
a2 ab                   v[i][j] = aug[i][j];
9c cb           }

5f d8           return {x, solcount};

42 cb       }
```

```
        /*
        Finds a possible solution for the system of linear equations,
             as well as a
        basis for the solution. The set of solutions will be a linear
             combination of
        the basis, added to the initial answer provided.

        First return value is the initial solution, and the second is
             the basis of the solution.
        If there is no solution, both return values will be empty
           vectors.
        */
d8 cb   pair<vector<T>, vector<vector<T>>> basis_solution(vector<T> y
   ){
54 af       auto [x0, solcount] = solve_system(y);

09 57       if(solcount == 0){
b0 21           return {};
d3 cb       }

94 e3       vector<int> pivot(n);
73 35       vector<int> pivoted(m);
26 60       for(int i = 0; i < n; i++){
38 10           while(pivot[i] < m && v[i][pivot[i]] == T(0))
3d 8f               pivot[i]++;
b3 9a           if(pivot[i] < m)
5e ed               pivoted[pivot[i]] = 1;
06 cb       }

ba be       vector<vector<T>> basis;
79 dd       for(int i = 0; i < m; i++){
3e e8           if(pivoted[i])
2c 5e               continue;
b8 04           vector<T> cbasis(m);
af e0           cbasis[i] = 1;
37 57           for(int j = 0; j < n; j++){
d9 35               if(pivot[j] != m)
88 8e                   cbasis[pivot[j]] += T(-1)*v[j][i];
61 cb           }
51 90           basis.push_back(cbasis);
a2 cb       }
7b 71       assert(bool(solcount > 1) == bool(basis.size()));

27 8d       return {x0,basis};
6a cb   }

         /*
        Does not alter current matrix.
        Returns {inverse matrix, is curent matrix invertable}
        */
e5 10   pair<Matrix<T>, bool> find_inverse(){
45 3d       int n = v.size();
fc 02       Matrix<T> aug(n, 2*n);

a2 83       for(int i = 0; i < n; i++)
94 f9           for(int j = 0; j < n; j++)
c7 78               aug[i][j] = v[i][j];

34 83       for(int i = 0; i < n; i++)
0c 4c           aug[i][n+i] = 1;

28 90       T det = aug.gaussjordanize(1);

a3 18       Matrix<T> ret(n,n);
78 60       for(int i = 0; i < n; i++){
30 16           ret[i] = valarray<T>(aug[i][slice(n,n,1)]);
a2 cb       }

04 59       return {ret, det != T(0)};
68 cb   }

        // Returns rank of matrix. Does not alter it.
66 2c   int get_rank() const {
a0 09       if(m == 0)
bf bb           return 0;

fe 34       Matrix<T> aux(*this);

25 c9       aux.gaussjordanize();

44 3b       int resp = 0;

b0 83       for(int i = 0; i < n; i++)
b0 9a           resp += (aux[i] != valarray<mint>(m)).sum();

56 68       return resp;
75 cb   }

48 21 };
```

3.12 MatrixMulMod.hpp

```
Hash: 378c2d
```

```
/*
from https://github.com/defnotmee/definitely-not-a-lib

Fast matrix multiplication with modulo. Useful for matrix
exponentiation problems and such.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

6b e3 template<ull M = MOD>
a2 cb void mat_mul(matrix<ll> a, matrix<ll> b){
55 50     matrix<ll> ret(a.size(), vector<ll>(b[0].size()));

d1 bb     for(int i = 0; i < a.size(); i++){
be 12         for(int j = 0; j < b[0].size(); j++){
74 9b             int ct = LONG_LONG_MAX/(M*M);
91 58             for(int k = 0; k < b.size(); k++, ct--){
91 34                 ret[i][j] += a[i][k]*b[k][i];
03 1a                 if(ct)
fc 73                     ret[i][j]%=M;
1d cb             }
b7 73             ret[i][j]%=M;
f5 cb         }
c8 cb     }

ab ed     return ret;
37 cb }
```

3.13 Modint.hpp

```
Hash: b90b52
/*
from https://github.com/defnotmee/definitely-not-a-lib

Implements integers in Z_MOD.
At all points it is assumed that 0 <= x < MOD and that MOD*MOD + MOD
    fits unsigned long long

If you want non-const MOD, use beegmod.cpp

*** If you only want to one value of MOD, check the "mint" alias at the
     bottom of the code. ***
*/
```

```
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

ac 86 template<ull M>
b1 1b struct modint{
34 0e     ull x;

          // It is assumed -M <= v.
ba e7     constexpr modint(ll v = 0) : x(v >= M ? (v+M)%M : v){};

2d c3     bool operator==(const modint& o){
56 d2         return x == o.x;
47 cb     }

          // Example on how to implement operators if youre lazy:
          // modint operator+(modint b){
          //     return x+b.x;
          // }

4a 1c     modint operator+(modint b) const{
16 dc         return min(x+b.x, x+b.x-M);
2b cb     }

00 d7     modint operator-(modint b) const{
de 6b         return min(x-b.x, x-b.x+M);
84 cb     }

c0 ac     modint operator*(modint b) const {
b1 dc         return x*b.x%M;
c0 21     };

a8 2f     modint inverse(){
7d 26         ll x = this->x, y = M;

68 ca         complex<ll> cx = {1,0}, cy = {0,1};

8c f4         while(x){
bd 41             ll curdiv = y/x;
a9 61             y-=curdiv*x;
34 eb             cy-=curdiv*cx;
31 0e             swap(cx, cy);
7b 9d             swap(x, y);
4e cb         }

a3 77         return cy.real();
83 cb     }
f5 e3     modint operator/(modint b) const {
12 78         return *this*b.inverse();
10 cb     }

d5 34     void operator+=(modint b){
6c 4f         x = min(x+b.x, x+b.x-M);
62 cb     }

41 cc     void operator-=(modint b){
b2 60         x = min(x-b.x, x-b.x+M);
3f cb     }

66 41     void operator*=(modint b){
72 76         (x*=b.x)%=M;
85 cb     }

7c 92     void operator/=(modint b){
74 7d         *this = *this/b;
5d cb     }

bf 66     friend istream& operator>> (istream& in, modint& v){
48 4a         ll x;
6b 14         in >> x;
69 6c         v = modint(x);
a8 09         return in;
bc cb     }

cf e8     friend ostream& operator<< (ostream& out, modint v){
3d d5         return out << v.x;
4f cb     }
4f 21 };

b9 9a using mint = modint<MOD>;
```

3.14 Polynomial.hpp

```
Hash: 00f872
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif
```

```
26 67 template<typename T>
c7 21 struct Poly{
59 1a     int n;
0e b8     vector<T> p;

d7 03     Poly(int n) : n(n), p(n){}
a4 0c     Poly(const vector<T>& v) : n(v.size()), p(v){}

bd 8d     T& operator[](int id) {
ab 77         return p[id];
bf cb     }

44 c8     Poly operator+(Poly b) const {
29 f7         Poly ret(max(n, b.n));

0a a8         for(int i = 0; i < ret.n; i++)
f0 10             ret[i] = p[i]+b[i];
e0 ed         return ret;
6c cb     }

75 bd     Poly operator-(Poly b) const {
30 f7         Poly ret(max(n, b.n));

51 a8         for(int i = 0; i < ret.n; i++)
ef 09             ret[i] = p[i]-b[i];
e6 ed         return ret;
c6 cb     }

6e 88     Poly operator*(Poly b) const {
13 3e         Poly ret(n+b.n-1);

9c 83         for(int i = 0; i < n; i++)
ce 11             for(int j = 0; j < b.n; j++)
8f 75                 ret[i+j] += p[i]*b[j];

18 ed         return ret;
c4 cb     }

71 aa     Poly operator*(T b) const {
df 69         Poly ret = *this;
eb 83         for(int i = 0; i < n; i++)
a8 09             ret[i]*=b;
21 ed         return ret;
05 cb     }

0a ca     Poly operator%(Poly b) const {
d1 7a         Poly ret(*this);
24 36         b*=T(1)/b.p.back();

7b 66         for(int i = n-b.n; i >= 0; i--){
d3 ef             T scale = ret[i+b.n-1];
35 11             for(int j = 0; j < b.n; j++)
19 c9                 ret[i+j]-=b[j]*scale;
eb cb         }

8e 66         ret.p.resize(b.n-1);
d3 04         ret.n = b.n-1;

98 ed         return ret;
85 cb     }

d0 e2     void operator%=(Poly b) {
93 7b         (*this) = (*this) % b;
e4 cb     }

f4 21     void operator+=(Poly b){
cd 17         (*this) = (*this) + b;
9e cb     }

a4 11     void operator-=(Poly b){
ff 46         (*this) = (*this) - b;
62 cb     }

62 5b     void operator*=(Poly b){
e2 b3         (*this) = (*this) * b;
fa cb     }

b3 e2     void operator*=(T b){
3e b3         (*this) = (*this) * b;
7e cb     }
00 21 };
```

3.15 Sieve.hpp

```
Hash: bfe02d
/*
from https://github.com/defnotmee/definitely-not-a-lib

Calculates smallest prime that divides each number for
all x < n and also maintains a list of all primes up to that
in O(n)

By default creates a sieve named sieve of size MAXN.
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
91 f2 #endif

78 3a struct Sieve{
66 fd     vector<int> primes;
38 89     vector<int> next;

20 8b     Sieve(int n){
84 8c         next = vector<int>(n);

0f 9f         for(int i = 2; i < n; i++){
d3 72             if(!next[i])
22 20                 next[i] = i, primes.push_back(i);

17 7c             for(ll j : primes){
0d a1                 if(j*i >= n)
2b c2                     break;
b8 da                 next[j*i] = j;
e0 4f                 if(j == next[i])
0d c2                     break;
e1 cb             }
96 cb         }
a5 cb     }

be 2a     inline bool is_prime(int n){
22 74         return next[n] == n;
55 cb     }

          // returns pairs in form {prime, exponent}
          // will always return them in ascending order
e0 bb     vector<pii> factorize(int n){
3a a5         vector<pii> ret;

a7 02         while(n != 1){
24 f6             int p = next[n];
ad d9             int ct = 0;
73 bf             while(n%p == 0)
42 31                 ct++, n/=p;
e6 fd             ret.push_back({p,ct});
88 cb         }
65 ed         return ret;
11 cb     }
bf 93 } sieve(1e6+10);
```

4 string

4.1 AhoCorasik.hpp

```
Hash: 207c79
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
ac 30 #include"trie.hpp"
8d 6d #include"../utility/template.cpp"
c4 f2 #endif

2b e8 template<int ALPHA = 26, int INI = 'a'>
2f 83 struct SuperTrie : Trie<ALPHA, INI>{
59 02     vector<int> in_suffix, slink, pai, paic, match;
a1 53     using Trie<ALPHA,INI>::trie;
2e 09     vector<bstring<int>> rslink;

92 1f     SuperTrie(int expected = MAXN) : Trie<ALPHA, INI>(MAXN){}

7b a4     int next(int id, int c){
78 fe         while(id && trie[id].ptr[c] == -1)
b7 3a             id = slink[id];
8c 11         if(trie[id].ptr[c] != -1)
ed 90             id = trie[id].ptr[c];
a3 64         return id;
63 cb     }

34 a2     void calc_link(){
fb 5a         in_suffix = slink = pai = paic = match = vector<int>(trie
    .size());
87 c4         rslink = vector<bstring<int>>(trie.size());
96 26         queue<int> q;

bb 53         q.push(0);

93 14         while(!q.empty()){
03 69             int cur = q.front();
7a 83             q.pop();
12 6b             for(int c = 0; c < ALPHA; c++){
ca f8                 int viz = trie[cur].ptr[c];
92 60                 if(viz == -1)
f6 5e                     continue;
ed aa                 pai[viz] = cur;
58 71                 paic[viz] = c;
7a 84                 q.push(viz);
e8 cb             }
d0 b3             if(!cur)
97 5e                 continue;

2b bb             slink[cur] = next(slink[pai[cur]], paic[cur]);
9e 59             slink[cur] = (slink[cur] != cur)*slink[cur];
ba bd             rslink[slink[cur]].push_back(cur);

46 c5             in_suffix[cur] = in_suffix[slink[cur]]+trie[cur].term
    ;
e2 cb         }
8c cb     }

73 84     void add_str(string& s, int ct = 1){
c5 04         int id = 0;
9b 0a         int sid = 0;

48 d5         while(sid < s.size()){
73 ba             int c = s[sid] - INI;
91 f0             id = next(id,c);
7e b7             match[id] += ct;
d5 be             sid++;
a3 cb         }
d2 cb     }

a6 fb     void calc_match(int id = 0){
e3 67         for(int i : rslink[id]){
4e a7             calc_match(i);
8f b8             match[id]+=match[i];
22 cb         }
44 cb     }
20 21 };
```

4.2 HashInterval.hpp

```
Hash: 3b59e4
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif
```

```
6b e3 template<ull M = MOD>
e8 a2 struct Hasher{
d0 ce     vector<ull> psum, power;

4a 0d     Hasher(string& s, ull c = 123){
7e 77         psum = vector<ull>(s.size()+1);
e0 f5         power = vector<ull>(s.size()+1,1);
ea 63         for(int i = 1; i < power.size(); i++)
26 7c             power[i] = power[i-1]*c%M;
ad 01         for(int i = 1; i < psum.size(); i++)
27 a5             (psum[i] = psum[i-1]*c+s[i-1])%=M;
a6 cb     }

f6 47     ull sub_hash(int l, int r){
66 79         return (psum[r+1]-psum[l]*power[r-l+1]%M+M)%M;
e6 cb     }
84 bf     ull hash(){
3d 08         return psum.back();
0a cb     }
3b 21 };
```

4.3 Kmp.hpp

```
Hash: f1429a
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Let pi(s) := size of the biggest proper prefix of s that is
 * also a suffix of s.
 *
 * kmp(s) calculates pi(s[0..i]) for all i from 0 to n-1 in O(n)
 *
 * Useful for multiple applications, for example string matching.
 *
 * Also has a function that finds the automaton corresponding
 * to how the prefix function will change. Formally:
 *
 * kmp_automaton<ALPHA,INI>(s)[p][c-INI] := pi(s+"#"+t+"c") given that
 * pi(s+"#"+t) = p, the alphabet size is ALPHA and the smallest
    character
 * is INI.
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

26 67 template<typename T>
18 33 vector<int> kmp(const T s){
a5 27     vector<int> pi(s.size());
6b 88     for(int i = 1; i < s.size(); i++){
7c 8d         pi[i] = pi[i-1];
0f e3         while(pi[i] != 0 && s[pi[i]] != s[i]){
10 77             pi[i] = pi[pi[i]-1];
2d cb         }
5f 18         pi[i]+=s[i]==s[pi[i]];
b5 cb     }
a5 81     return pi;
9e cb }

26 e8 template<int ALPHA = 26, int INI = 'a'>
4d 08 vector<array<int,ALPHA>> kmp_automaton(const string s){
0b 6b     vector<int> pi = kmp(s);
ce 5b     vector<array<int,ALPHA>> ret(s.size()+1);

52 ff     ret[0][s[0]-INI] = 1;
a9 88     for(int i = 1; i < s.size(); i++){
24 12         ret[i] = ret[pi[i-1]];
26 b1         ret[i][s[i]-INI] = i+1;
b3 cb     }
05 69     ret.back() = ret[pi.back()];

3e ed     return ret;
f1 cb }
```

4.4 Manacher.hpp

```
Hash: 9f4181
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

1c fc vector<int> manacher(string in){
92 9b     string s = "#";

d3 31     for(char i : in)
2d 94         s+=i + string("#");

fd 16     int n = s.size();
12 82     vector<int> ret(n);

e4 39     int mid = 0, r = 0;

de a4     for(int i = 1; i < n-1; i++){
da 88         if(i < r)
fc e9             ret[i] = min(r-i, ret[mid*2-i]);

bb 3d         while(ret[i] <= i && i+ret[i] < n && s[i-ret[i]] == s[i+
    ret[i]])
55 84             ret[i]++;

06 c6         if(i+ret[i] > r)
cd 59             mid = i, r = i+ret[i];
58 cb     }

c4 6c     return vector<int>(1+all(ret)-1);
9f cb }
```

4.5 MinRot.hpp

```
Hash: 2aac66
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Given a string/vector s, finds all the lexicographically minimum
 * rotations of s in O(nlogn)
 */

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

26 67 template<typename T>
fd 14 vector<int> min_rot(T v){
2c 3d     int n = v.size();
09 a9     vector<int> cand;
ce b2     auto mn = *min_element(all(v));
8a 87     for(auto i : v)
87 4c         if(i == mn)
c0 5b             cand.push_back(i);

a2 f9     vector<int> is_cand(n);
78 ea     for(int i : cand)
41 7f         is_cand[i] = 1;
5a bf     int k = 1;
84 66     while(true){
0b 7a         auto mn = v[(cand[0]+k)%n];

e2 98         for(int i : cand){
17 b6             is_cand[(i+k)%n] = 0;
6b 1b             if(v[(i+k)%n] != mn)
ee 28                 is_cand[i] = 0;
89 cb         }
91 00         vector<int> newcand;

92 ea         for(int i : cand)
66 d4             if(is_cand[i])
05 53                 newcand.push_back(i);

55 75         if(newcand.empty())
31 9c             return cand;

91 06         swap(cand,newcand);

3f ac         k++;
f1 cb     }
2a cb }
```

4.6 SuffixArray.hpp

```
Hash: bcbfc1
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

13 3f struct SuffixArray{
a6 1a     int n;
8b ac     string s;
d9 74     vector<int> sa, rnk;

03 19     SuffixArray(string& s) : s(s), n(s.size()), sa(n), rnk(n
    +1,-1){
95 83         for(int i = 0; i < n; i++)
4e 16             rnk[i] = s[i];
17 b9           iota(all(sa),0);

24 c3           for(int k = -1; k == -1 || (1<<k) <= n; k++){
be ea               int off = k == -1 ? 0 : (1<<k);

8c 1e               vector<pii> lookup(n);
71 54               vector<int> ct(max(256, n));
17 ee               vector<int> nsa(n);

19 60               for(int i = 0; i < n; i++){
60 30                   ct[rnk[i]]++;
d6 6a                   lookup[i] = {rnk[i], rnk[min(n,i+off)]};
e0 cb               }

d8 ee               vector<int> ps = ct;

9c ee               for(int i = 1; i < ps.size(); i++)
a9 36                   ps[i]+=ps[i-1];

91 ea               auto aux =[&](int id){
45 1e                   nsa[ps[rnk[id]] - (ct[rnk[id]]--)] = id;
56 21               };

e9 7c               for(int i = n-off; i < n; i++)
1f 63                   aux(i);

ca 83               for(int i = 0; i < n; i++)
3a 52                   if(sa[i] >= off)
87 3b                       aux(sa[i]-off);


a9 43               swap(sa,nsa);

6b f3               rnk[sa[0]] = 0;

4d aa               for(int i = 1; i < n; i++)
9d b8                   rnk[sa[i]] = rnk[sa[i-1]]+(lookup[sa[i]] !=
    lookup[sa[i-1]]);

ec cb           }

61 75           rnk.pop_back();

66 cb       }
69 21 };

bd 6a struct LCP : SuffixArray{
28 a5     vector<int> lcp;

bf 77     matrix<int> sparse;

8b c0     LCP(string& s) : SuffixArray(s), lcp(n), sparse(int(log2(n)
    +1), vector<int>(n)) {

d9 60         for(int i = 0; i < n; i++){
96 27             int& clcp = lcp[rnk[i]];
39 15             if(rnk[i]+1 == n){
33 11                 clcp = 0;
e4 5e                 continue;
fc cb             }
46 a7             int nxt = sa[rnk[i]+1];

68 59             while(i+clcp < n && nxt+clcp < n && s[i+clcp] == s[
    nxt+clcp]){
f4 9c                 clcp++;
bc cb             }

f2 9a             if(i+1 < n)
6d 2a                 lcp[rnk[i+1]] = max(0,clcp-1);
7d cb         }

c0 2d         sparse[0] = lcp;

88 61         for(int i = 1; i < sparse.size(); i++){
df 8b             for(int j = 0; j + (1<<i) <= n; j++){
92 61                 sparse[i][j] = min(sparse[i-1][j], sparse[i-1][j
    +(1<<i-1)]);
81 cb             }
49 cb         }
9a cb     }

          // returns the lcp between s[sa[l]..n] and s[sa[r]..n]
5c 9e     int get_lcp_sa(int l, int r){
3b c2         if(l > r)
eb e4             swap(l,r);
46 61         r--;
06 1e         int logg = log2(r-l+1);
42 d3         return min(sparse[logg][l], sparse[logg][r-(1<<logg)+1]);
6e cb     }

          // returns lcp between s[l..n] and s[r..n]
ed f9     int get_lcp(int l, int r){
42 29         return get_lcp_sa(rnk[l], rnk[r]);
ff cb     }
```

```
c6 e6     void debug(){
4e 1c         for(int i = 0; i < s.size(); i++){
cb 68             cerr << i << ": " << "sa[i] = " <<sa[i] << ", suffix
    = " << s.substr(sa[i]) << ", lcp = " << lcp[i] << '\n';
75 cb         }
af cb     }
bc 21 };
```

4.7 SuffixAutomaton.hpp

```
Hash: c79348
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

92 e8 template<int ALPHA = 26, int INI = 'a'>
e6 92 struct SuffixAutomaton {
02 2d     struct state{
b4 01         int len, firstpos, link = -1, is_clone = 0;
4c be         array<int, ALPHA> ptr;

8b fc         state(int len = 0, int firstpos = -1)
1d ae             : len(len), firstpos(firstpos){
36 b0             fill(all(ptr),-1);
2d cb         }
47 21     };

fa 76     vector<state> sa;
10 5c     int last = 0;
3b d9     int ct = 0;
c3 a1     SuffixAutomaton(){
4f 97         sa = {{}};
29 cb     }

fb 62     void extend(int c){
5c a6         c-=INI;
25 d9         int nxt = sa.size();
89 c0         sa.push_back({sa[last].len+1, ct++});

cf 94         int pivot = last;

11 3b         while(pivot != -1 && sa[pivot].ptr[c] == -1)
14 78             sa[pivot].ptr[c] = nxt, pivot = sa[pivot].link;

4c b9         if(pivot == -1){
7d da             sa[nxt].link = 0;
bd 9d         } else {
49 52             int tmp = sa[nxt].link = sa[pivot].ptr[c];
e4 7f             if(sa[tmp].len != sa[pivot].len+1){
ab f7                 int nlink = sa[nxt].link = sa.size();
ad 0b                 sa.push_back(sa[tmp]);
1d d9                 sa.back().len = sa[pivot].len+1;
3b a3                 sa.back().is_clone = 1;
0c 82                 int tmp = sa[pivot].ptr[c];
c8 58                 sa[tmp].link = nlink;

a5 9d                 while(pivot != -1 && sa[pivot].ptr[c] == tmp){
83 29                     sa[pivot].ptr[c] = nlink;
25 33                     pivot = sa[pivot].link;
34 cb                 }
42 cb             }
32 cb         }
f0 a7         last = nxt;
fa cb     }

be 84     SuffixAutomaton(string s){
d7 97         sa = {{}};
f2 3b         sa.reserve(2*s.size());
0a 29         for(char i : s)
6f 89             extend(i);

30 cb     }

7c a4     bool accept(string s){
53 04         int id = 0;

33 54         for(char i : s){
30 b8             id = sa[id].ptr[i-INI];
06 a5             if(id == -1)
4a bb                 return 0;
38 cb         }
3a 6a         return 1;
37 cb     }

b7 30     vector<int> get_ct(){
52 8f         vector<int> ct(sa.size());
55 88         vector<bstring<int>> rslink(sa.size());

3e b5         for(int i = 1; i < sa.size(); i++){
```

```
90 50             rslink[sa[i].link].push_back(i);
4c cb         }

d7 82         auto dfs =[&](int id, auto && dfs) -> void {
db 50             ct[id] = !sa[id].is_clone;

82 4d             for(int i : rslink[id])
29 49                 dfs(i,dfs), ct[id]+=ct[i];
81 21         };

04 c2         dfs(0,dfs);

55 b6         return ct;
98 cb     }

c1 81     state& operator[](int id){
ea 39         return sa[id];
a7 cb     }

9f 28     int size(){
a1 bd         return sa.size();
ad cb     }
c7 21 };
```

4.8 Trie.hpp

```
Hash: 136607
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

92 e8 template<int ALPHA = 26, int INI = 'a'>
19 71 struct Trie {
1a 67     public:
b6 3c     struct node{
39 be         array<int,ALPHA> ptr;
50 f7         int term; // number of strings that terminate on the node
9b bf         int sub;  // number of strings in the subtree of the node

62 a7         constexpr node() : term(0), sub(0){
44 b5             for(int i = 0; i < ALPHA; i++)
f8 99                 ptr[i] = -1;
a5 cb         }
74 21     };
6d 95     vector<node> trie;

b5 99     Trie(int expected = MAXN) : trie(1) {
41 48         trie.reserve(expected);
cb cb     }

38 cd     void insert(const string& s, int ct = 1){
e2 04         int id = 0;
30 be         int pos = 0;
ff 51         while(pos < s.size()){
3d 72             char cur = s[pos]-INI;
53 42             if(trie[id].ptr[cur] == -1)
09 a3                 trie[id].ptr[cur] = trie.size(), trie.push_back
    ({});
6e c2             trie[id].sub+=ct;
97 8a             id = trie[id].ptr[cur];
31 65             pos++;
f6 cb         }
1c c2         trie[id].sub += ct;
18 9a         trie[id].term += ct;
2f cb     }

99 e4     int find(const string& s){
09 43         int id = 0, pos = 0;
d5 51         while(pos < s.size()){
60 72             char cur = s[pos]-INI;
3a 42             if(trie[id].ptr[cur] == -1)
c0 da                 return -1;
b1 8a             id = trie[id].ptr[cur];
e4 65             pos++;
1d cb         }
7a 64         return id;
6d cb     }
13 21 };
```

4.9 ZFunction.hpp

```
Hash: d2a27d
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif
```

```
3a ba vector<int> z_function(auto s){
00 16     int n = s.size();
69 fa     int l = 0, r = 0;
fd 2e     vector<int> z(n);
a3 6f     for(int i = 1; i < n; i++){
76 88         if(i < r)
88 14             z[i] = min(r-i, z[i-l]);
a2 81         while(i + z[i] < n && s[i+z[i]] == s[z[i]])
5a 9f             z[i]++;
4b fe         if(i+z[i] > r)
d0 a1             l = i, r = i+z[i];
fd cb     }
58 07     return z;
d2 cb }
```

5 geometry

5.1 Point.hpp

```
Hash: df8967
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
77 12 #include"template.cpp"
9d f2 #endif

3a 14 template<typename T = ll>
c9 be struct point{
96 64     T x, y;

20 ab     inline point operator+(point b){
75 4f         return {x+b.x, y+b.y};
34 cb     }

c1 5d     inline point operator-(point b){
cb 53         return {x-b.x, y-b.y};
d9 cb     }

e9 92     inline point operator*(T scale){
02 1a         return {x*scale, y*scale};
81 cb     }
```

```
57 92     inline T cross(point b){
44 a9         return x*b.y-b.x*y;
67 cb     }

47 27     inline T dot(point b){
f3 e0         return x*b.x + y*b.y;
75 cb     }

a8 fd     inline T dist2(){
cd 2b         return x*x+y*y;
e0 cb     }

2b fe     inline double dist(){
0a d4         return sqrt(dist2());
cc cb     }
df 21 };
```

6 graph

6.1 2sat.hpp

```
Hash: 0f603e
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
df 3e #include"scc.hpp"
2f f2 #endif

19 d9 struct TwoSat{
29 1a     int n;
f6 3c     SCC scc;

61 e2     TwoSat(int n = 0) : n(n), scc(2*n){}

7f b1     static constexpr int no(int x){
7a 61         return 2*x;
d4 cb     }
68 50     static constexpr int yes(int x){
0d 46         return 2*x+1;
e0 cb     }
c9 b5     void add_or(int a, int b){
dd 56         scc.add_edge(a^1, b);
6f 18         scc.add_edge(b^1, a);
7b cb     }

f3 d9     void add_xor(int a, int b){
3a 23         add_or(a,b);
56 77         add_or(a^1,b^1);
3c cb     }

          // If impossible, returns an empty vector
          // If possible, returns a possible construction where
          // ret[i] = 1 <=> i is true
2a 6e     vector<int> get_sat(){
0b 41         scc.kosaraju();
08 82         vector<int> ret(n);

12 60         for(int i = 0; i < n; i++){
95 32             if(scc.scc[no(i)] == scc.scc[yes(i)])
53 21                 return {};
16 60             ret[i] = scc.scc[no(i)] < scc.scc[yes(i)];
22 cb         }

1d ed         return ret;
d1 cb     }

0f 21 };
```

6.2 BinaryLift.hpp

```
Hash: 017f86
/*
from https://github.com/defnotmee/definitely-not-a-lib

Given an array of ancestors (par), is able to get information
about starting on a certain node and going to the ancestor of the
current node k steps in a row in O(log(k)) per query. Is able to work
    with
any functional graph, but the lca function just works for trees.

Usage:

- BinLift(par): constructs the structure. par is assumed to be 0-
    indexed
- lift: an auxiliary class that stores information about the path (for
    example
what is the maximum edge on the path). By default only stores the
    vertex you will end
up in after going up a certain number of times.
- k_up(id,k): returns a lift structure of starting on id and going to
    the ancestor
k times in a row.
- lca(a,b,h): assuming the functional graph given is a tree, if h is a
    vector representing
the height of the nodes in a tree, returns the lift structure of the
    path between a and b.
The .to member of the return value will be the lca between a and b. If
    you are storing more
information about the path, it needs to be commutative (for example,
    you can store max).
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
91 f2 #endif

5f 6b struct lift{
18 70     int to = 0;
44 aa     int mn = 1e9; // Example of path agregate, must be identity
    value through merge
bd 21 };

e1 71 struct BinLift{

5c 8b     int n, lg;

          // what happens when you go through a, and then go through b?
6f 4e     static lift merge(lift a, lift b){
bb 97         return {b.to, min(a.mn, b.mn)};
f6 cb     }

b0 50     matrix<lift> jmp;

c9 be     BinLift(vector<lift> par) : n(par.size()), lg(log2(n)+1){
7f 38         jmp = matrix<lift>(lg,par);

60 82         for(int i = 1; i < lg; i++){
a1 27             for(int j = 0; j < par.size(); j++){
55 52                 jmp[i][j] = merge(jmp[i-1][j], jmp[i-1][jmp[i-1][
    j].to]);
d6 cb             }
d0 cb         }
7c cb     }

80 fe     lift k_up(int id, int k){
9d 51         lift ret{id}; // needs to be an identity element through merge
ca 95         while(k){
1c 3e             ret = merge(ret, jmp[__builtin_ctz(k)][ret.to]);
f6 ab             k-=k&-k;
73 cb         }
9a ed         return ret;
98 cb     }

58 b2     lift lca(int a, int b, vector<int>& h){
03 be         if(h[a] < h[b])
c8 25             swap(a,b);

28 fe         int d = h[a]-h[b];
4a 91         lift la = k_up(a,d), lb = {b}; // needs to be an identity element through merge

b0 97         if(la.to == lb.to)
60 c9             return la;

77 35         for(int i = lg-1; i >= 0; i--){
12 7e             if(jmp[i][la.to].to != jmp[i][lb.to].to)
82 4c                 la = merge(la,jmp[i][la.to]), lb = merge(lb,jmp[i][lb.to]);
03 cb         }

c1 d4         la = merge(la, jmp[0][la.to]);
af 04         lb = merge(lb, jmp[0][lb.to]);


fd 91         return merge(la,lb);
a8 cb     }

01 21 };
```

6.3 BipartiteMatching.hpp

```
Hash: 915763
/*
from https://github.com/defnotmee/definitely-not-a-lib

Uses hopcroft-karp's algorithm to find the maximum matching on a bipartite graph. Runs in time O(E*sqrt(V)) on worst case, and time O(E*log(V)) on random graphs.

Depending on the aplication, the dinic.hpp interface may be more convenient.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

86 53 struct BiGraph{ // bipartite graph of sizes n and m

83 14     int n, m;
fc 3b     vector<basic_string<int>> g;
0f d3     vector<int> matched, match;

4e b3     BiGraph(int _n, int _m) : n(_n), m(_m), g(n), matched(n), match(m,-1){}

42 01     void add_edge(int a, int b){
02 02         g[a].push_back(b);
10 cb     }

00 0b     vector<pii> max_matching(){

17 4b         while(augment());

03 14         vector<pii> resp;
af 94         for(int i = 0; i < m; i++)
13 e3             if(match[i] != -1)
ca 76                 resp.push_back({match[i], i});

56 68         return resp;
52 cb     }

ac bf     private:
2a cb     bool augment(){
b5 ee         vector<int> dist(n, -1);

9c 26         queue<int> q;
d6 60         for(int i = 0; i < n; i++){
b8 89             if(!matched[i])
14 4b                 q.push(i), dist[i] = 0;
2b cb         }

19 28         bool fail = 1;
44 9d         while(!q.empty() && fail){
2a 69             int cur = q.front();
c1 83             q.pop();
85 95             for(int i : g[cur]){
4c 56                 if(match[i] == -1){
72 1e                     fail = 0;
83 c2                     break;
78 cb                 }
7f 56                 if(dist[match[i]] == -1){
5c 65                     dist[match[i]] = dist[cur]+1;
25 c5                     q.push(match[i]);
57 cb                 }
d1 cb             }
71 cb         }

88 59         if(fail)
5d d1             return false;

78 f9         vector<int> check(n);

9f 44         auto dfs =[&](int id, auto && dfs) -> bool {
5e e8             check[id] = 1;
4b a6             for(int i : g[id]){
8d 97                 int& mi = match[i];
15 98                 if(mi == -1 ||
8b bf                 (!check[mi] && dist[mi] == dist[id]+1 && dfs(mi,
   dfs))){
47 90                     mi = id;
fa c5                     matched[id] = 1;
4a 8a                     return true;
8b cb                 }
81 cb             }
76 d1             return false;
45 21         };

8a 60         for(int i = 0; i < n; i++){
ed 3a             if(!check[i] && !matched[i])
c3 e6                 dfs(i,dfs);
a1 cb         }

d0 8a         return true;
c5 cb     }

91 21 };
```

6.4 Dinic.hpp

```
Hash: e411bf
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 * Based on https://github.com/kth-competitive-programming/kactl/blob/
   main/content/graph/Dinic.h
 *
 * Uses Dinic's algorithm to find maximum flow between two vertices.
 *
 * O(VElog(U)), where U is max capacity. Faster in practice. On unit
   networks
 * (graphs where capacities not connected to source or sink are 1),
   complexity
 * improves to O(sqrt(V)E).
 *
 * After calling max_flow, the corresponding flow on edges is
   recoverable
 * with Edge::flow() and left_of_mincut becomes well defined.
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif


27 14 struct Dinic{
d6 e9     struct Edge{
8c e4         ll to, cap, ocap, rev;
93 c2         ll flow(){
eb f2             return max(ocap-cap, 0ll);
bf cb         }
c6 21     };

af ed     vector<vector<Edge>> g;

31 80     void add_edge(int u, int v, ll cap){
1d 60         g[u].push_back({v,cap,cap,(ll)g[v].size()});
f2 ee         g[v].push_back({u, 0, 0, (ll)g[u].size()-1});
4f cb     }

          // Returns if v is in the same side of the min_cut as s
88 4b     bool left_of_mincut(int v){
59 96         return dist[v] != -1;
da cb     }

8c ff     ll max_flow(int s, int t){
91 7a         ll flow = 0;
e4 98         for(int k = 30; k >= 0; k--)
12 ed             while(bfs(s,t,k)) while (ll it = dfs(s,t,LLONG_MAX))
    flow += it;
d5 99         return flow;
dd cb     }

89 2a     Dinic(int n) : g(n), ptr(n), dist(n){}

16 bf     private:
3c c7     vector<int> ptr, dist;

eb 03     ll dfs(int id, int t, ll x){
75 f1         if(id == t || !x)
f0 ea             return x;

28 75         for(int & i =ptr[id]; i < g[id].size(); i++){
c5 28             Edge& e = g[id][i];
f4 6c             if(dist[e.to] != dist[id]+1)
a0 5e                 continue;
54 b4             if(ll filled = dfs(e.to, t, min(x, e.cap))){
38 06                 e.cap-=filled;
be 8c                 g[e.to][e.rev].cap+=filled;
91 2e                 return filled;
63 cb             }
78 cb         }

13 bb         return 0;
95 cb     }

65 c1     bool bfs(int s, int t, int k){
2f 4c         fill(all(ptr),0), fill(all(dist),-1);
33 ef         vector<int> q({s});
c8 66         q.reserve(g.size());
a4 a9         dist[s] = 0;
47 2a         for(int i = 0; i < q.size(); i++){
80 5a             int id = q[i];
95 37             for(auto i : g[id]){
33 5f                 if(dist[i.to] == -1 && (i.cap>>k)){
78 11                     dist[i.to] = dist[id]+1;
37 e0                     q.push_back(i.to);
a6 cb                 }
db cb             }
de cb         }
56 69         return dist[t]+1;
64 cb     }
e4 21 };
```

6.5 DsuRollback.hpp

```
Hash: 084442
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

47 d8 struct DSU_Rollback{
a8 61     struct log{
59 4b         int node1, node2;
9e a3         int prev1, prev2;
5b 21     };
97 bf     private:
f8 99     vector<int> v; // Either parent (if v[i] >= 0) or size (if v[
    i] < 0 and i is a root) of the component
17 2f     vector<log> history;
f7 67     public:
b3 2a     int comp_ct;

0f 37     DSU_Rollback(int n = 0) : v(n,-1), comp_ct(n){}

a1 a6     constexpr int size(int id){ // returns size of the component
    id belongs to
76 93         return -v[find(id)];
24 cb     }

f6 96     constexpr int pai(int id){ // Returns parent of id
ff 0c         return v[id] < 0 ? id : v[id];
9f cb     }


67 13     int find(int id){ // removing path compression
fa a4         return v[id] < 0 ? id : find(v[id]);
c9 cb     }

fb c8     bool onion(int a, int b){
04 bc         a = find(a);
8e b8         b = find(b);

1d ae         if(a == b)
c8 bb             return 0;

22 ad         if(size(a) > size(b)) // union by size
a0 25               swap(a,b);

1f 4c           comp_ct--;
69 69           history.push_back({a,b,v[a],v[b]});
93 72           v[b] += v[a];
e9 4c           v[a] = b;
42 6a           return 1;
14 cb       }

05 5c       void rollback(){
78 d5           auto [a,b,va,vb] = history.back();
67 8b           v[a] = va;
fd 99           v[b] = vb;
d2 2c           comp_ct++;
e5 7d           history.pop_back();
58 cb       }

60 3d       bool same(int a, int b){
5e c0           return find(a) == find(b);
5e cb       }

95 cd       constexpr int snapshot(){
0a 53           return history.size();
9e cb       }

08 21 };
```

6.6 DynamicConnectivity.hpp

```
Hash: d1c2a4
/*
from https://github.com/defnotmee/definitely-not-a-lib

Offline Dynamic Connectivity in O(nlog^2(n)). Allows for duplicate
    edges.
If an edge that doesn't exist is deleted, it is just ignored.

By default answers how many connected components were in the graph at
a given point.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
ea 4c #include"dsu_rollback.hpp"
09 f2 #endif
```

```
a4 fd struct Dynamic_Connectivity{
04 1a     int n;
d7 13     DSU_Rollback uf;
c8 e1     vector<pii> edges;
e2 1d     vector<int> ponta;
2c ce     map<pii, basic_string<int>> st;

4f 3d     Dynamic_Connectivity(int n = 0, int expected = 0) : n(n), uf(
   n){
64 c7         ponta.reserve(expected);
7f 81         edges.reserve(expected);
b8 cb     }

86 01     void add_edge(int a, int b){
4e f7         if(a > b)
b6 25             swap(a,b);
06 1e         st[{a,b}].push_back(edges.size());
f2 9b         edges.push_back({a,b});
18 e8         ponta.push_back(-2);
02 cb     }

ac 05     void rem_edge(int a, int b){
7e f7         if(a > b)
f0 25             swap(a,b);
3c 1f         if(st[{a,b}].empty()) // removing edge that is not there
e8 50             return;
00 62         int removed = st[{a,b}].back();
99 7d         st[{a,b}].pop_back();

33 87         ponta[removed] = edges.size();
aa b0         ponta.push_back(removed);
7a 9b         edges.push_back({a,b});
3d cb     }

0e e3     void add_query(){
d7 40         edges.push_back({-1,-1});
22 a4         ponta.push_back(-1);
65 cb     }


0a 9c     vector<int> solve(){
20 1e         for(int& i : ponta)
c8 28             if(i == -2) i = ponta.size();

16 07         vector<int> resp;

44 54         solve(0, int(ponta.size())-1,resp);
8e 68           return resp;
10 cb       }

51 bf       private:

a8 cb       void solve(int l, int r, vector<int>& resp){

05 89           if(l == r){
a4 93               if(ponta[l] == -1){
21 10                   resp.push_back(uf.comp_ct);
b3 cb               }
c3 50               return;
e7 cb           }

3d 77           int version = uf.snapshot();

0f 27           int m = (l+r)>>1;

e0 11           for(int i = m+1; i <= r; i++){
01 27               if(ponta[i] < l){
32 78                   uf.onion(edges[i].ff, edges[i].ss);
ef cb               }
20 cb           }

38 de           solve(l,m,resp);

b8 ea           while(uf.snapshot() != version)
95 c1               uf.rollback();

b7 e9           for(int i = l; i <= m; i++){
87 3d               if(ponta[i] > r){
23 78                   uf.onion(edges[i].ff,edges[i].ss);
e5 cb               }
25 cb           }

a1 12           solve(m+1,r,resp);

d1 ea           while(uf.snapshot() != version)
aa c1               uf.rollback();
a4 cb       }
d1 21 };
```

6.7 EdgeColoring.hpp

Hash: 603344

```
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Copied from https://github.com/koosaga/olympiad/blob/master/Library/
   codes/graph/edgecolor_bipartite.cpp
 *
 * I have no idea how to use this, but why not put it here.
 *
 * O(nm)
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

47 69 const int MAXM = 1000;

4f 53 struct edge_color{ // must use 1-based
03 fb   int deg[2][MAXN];
15 ae   pii has[2][MAXN][MAXN];
7b de   int color[MAXM];
ae 95   int c[2];
8c c4   void clear(int n){
a9 2e       for(int t=0; t<2; t++){
3d cc           for(int i=0; i<=n; i++){
ac c0               deg[t][i] = 0;
34 fe               for(int j=0; j<=n; j++){
c5 55                   has[t][i][j] = pii(0, 0);
d0 cb               }
3e cb           }
d3 cb       }
ea cb   }
d6 82   void dfs(int x, int p) {
5d e7       auto i = has[p][x][c[!p]];
46 6f       if (has[!p][i.first][c[p]].second) dfs(i.first,!p);
d6 8d       else has[!p][i.first][c[!p]] = pii(0,0);
4d 8f       has[p][x][c[p]] = i;
9b 22       has[!p][i.first][c[p]] = pii(x,i.second);
1a bd       color[i.second] = c[p];
3e cb   }
87 85   int solve(vector<pii> v, vector<int> &cv){
4e ef       int m = v.size();
5d 1a       int ans = 0;
bc c2       for (int i=1;i<=m;i++) {
fe 99           int x[2];
65 72           x[0] = v[i-1].first;
36 05           x[1] = v[i-1].second;
1e ec           for (int d=0;d<2;d++) {
```

```
e3 cf                 deg[d][x[d]]+=1;
cb 08                 ans = max(ans,deg[d][x[d]]);
c2 4b                 for (c[d]=1;has[d][x[d]][c[d]].second;c[d]++);
a3 cb             }
c7 d9             if (c[0]!=c[1]) dfs(x[1],1);
47 f7             for (int d=0;d<2;d++) has[d][x[d]][c[0]] = pii(x[!d],i)
    ;
3e 16             color[i] = c[0];
d2 cb         }
ef f5         cv.resize(m);
e8 c2         for(int i=1; i<=m; i++){
4d a0             cv[i-1] = color[i];
c6 bb             color[i] = 0;
7a cb         }
3f ba         return ans;
49 cb     }
60 b8 }EC;
```

6.8 FunctionalGraph.hpp

```
Hash: ab49e6
/*
from https://github.com/defnotmee/definitely-not-a-lib

Constructs a functional graph. Is able to answer distance directed
    distance
queries in O(1).

For each vertex stores the following information

- pai[v]: parent of a vertex
- height[v]: ammount of steps necessary to reach a vertex on a cycle
- cycleid[v]: which cycle v ends up in. If cycleid[v] != cycleid[u],
    they are on different components
- cyclepos[v]: index of the first vertex from the cycle that v touches
    on clist[cycleid[v]]
- tin[v]: preorder of v on its corresponding tree (rooted on clist[
    cycleid[v]][cyclepos[v]])
- tout[v]: preorder of v on its corresponding tree (rooted on clist[
    cycleid[v]][cyclepos[v]])

In addition, for each cycle, stores a list of the vertices in the cycle
     on clist[v]

All of this is O(n) preprocessing.
*/
```

```
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

a1 7c struct FuncGraph{
73 1a     int n;
8b f8     vector<int> pai, height, cycleid, cyclepos, is_cycle, tin,
    tout;
b6 0f     vector<basic_string<int>> rev, clist;

dc f6     FuncGraph(vector<int> v) : n(v.size()), pai(v), height(n),
f3 5a     rev(n), cycleid(n,-1), cyclepos(n), clist(n), is_cycle(n),
    tin(n), tout(n){
be 83         for(int i = 0; i < n; i++)
d2 3a             rev[pai[i]].push_back(i);

8e 60         for(int i = 0; i < n; i++){
70 f4             if(cycleid[i] == -1)
34 bc                 get_cycle(i);
ef cb         }
88 cb     }

c4 d0     void get_cycle(int id){
b0 5b         int a = id, b = id;

ad 01         do{
b6 5a             a = pai[a];
9f 57             b = pai[pai[b]];
d6 54         } while(a != b);

06 5f         process_cycle(a);
01 cb     }

30 97     void process_cycle(int id){
f5 e9         int cid = cycleid[id] = id;


ad 02         int v = id;
8a 01         do{
f2 b5             cyclepos[v] = clist[cid].size();
62 89             clist[cid].push_back(v);
2b 15             is_cycle[v] = 1;
a2 90             v = pai[v];
20 5a             cycleid[v] = cid;
d9 81         } while(v != id);
85 01           do{
24 6b               dfs(v);
13 90               v = pai[v];
2b 81           } while(v != id);

d0 cb       }

93 26       void dfs(int id){
30 36           tout[id] = tin[id];
89 c6           for(int i : rev[id]){
13 75               if(cycleid[i] == -1){
dd 24                   cycleid[i] = cycleid[id];
74 68                   cyclepos[i] = cyclepos[id];
cc db                   height[i] = height[id]+1;
21 7b                   tin[i] = ++tout[id];
2b 1e                   dfs(i);
d2 e6                   tout[id] = tout[i];
99 cb               }
b1 cb           }
18 cb       }

            // returns directed distance from a to b, or INF if its not
                possible to go from a to b
78 b5       int dist(int a, int b){
cd f4           if(cycleid[a] != cycleid[b])
16 cd               return INF;
2b 5f           if(is_cycle[a] && !is_cycle[b])
8f cd               return INF;
bf e7           if(!is_cycle[a] && !is_cycle[b]){
45 e4               if(height[a] < height[b] || cyclepos[a] != cyclepos[b
    ])
f4 cd                   return INF;
aa 17               if(tin[b] <= tin[a] && tin[a] <= tout[b]){
96 91                   return height[a]-height[b];
60 cb               }
c9 cd               return INF;
fd cb           }

8f 53           return height[a]+dist_in_cycle(cyclepos[a], cyclepos[b],
    clist[cycleid[a]].size());
6b cb       }

0e bf       private:

da 9b       int dist_in_cycle(int a, int b, int csize){
d9 7e           if(b >= a)
09 49               return b-a;
7f 03           return csize+b-a;
59 cb       }

ab 21 };
```

6.9 Hld.hpp

```
Hash: 02a01c
/*
from https://github.com/defnotmee/definitely-not-a-lib

Heavy-Light Decomposition. Does path queries in O(log^2(n)).
Requires a SegTree.
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
f8 1f #include"../../data_structures/segtree_lazy.hpp"
a3 f2 #endif

13 40 struct HLD {
3f bf     private:
11 bd     int n, root;
df ae     vector<int> tin, tout, sub, pai, height;
0e 3b     vector<basic_string<int>> g;
3b cb     int m = 0;
7f e0     SegTree st;
5f 31     vector<int> head;
75 82     using lazy = SegTree::lazy;
01 e9     using seg = SegTree::seg;

c8 67     public:
b5 d3     HLD(int n, int root = 0) : n(n), root(root), g(n),
af bb     tin(n), tout(n), sub(n,1), pai(n), height(n),
12 f7     st(n), head(n) {}

89 01     void add_edge(int a, int b){
91 02         g[a].push_back(b);
fb 3e         g[b].push_back(a);
bc 7b         m++;
a7 cb     }

58 11     void calc_tree(){
0e 9b         assert(m == n-1);
a9 00         prec(root);
8d 7d         hld(root,root);
9c cb     }

fc 76     void calc_tree(vector<seg>& v){
de 6e         calc_tree();
41 26         vector<seg> v2(n);
8f 83         for(int i = 0; i < n; i++)
94 16             v2[tin[i]] = v[i];
b4 c9         st = SegTree(v2);
ce cb     }

c6 7b     int lca(int a, int b){
5f 2d         while(head[a] != head[b]){
e3 06             if(tin[a] < tin[b])
6d 25                 swap(a,b);
e5 1f             a = pai[head[a]];
de cb         }
b8 9b         return min(a,b,[&](int a, int b){
23 db             return tin[a] < tin[b];
6d c0         });
c1 cb     }

7b b5     int dist(int a, int b){
36 c5         return height[a] + height[b] - 2*height[lca(a,b)];
9f cb     }

76 f5     void update_point(int id, SegTree::lazy upd){
7a 9c         st.update(tin[id], tin[id], upd);
5a cb     }

          // if no_root = 1, the root won't be included in the update;
4e d4     void update_subtree(int id, SegTree::lazy upd, int no_root = 0){
d9 58         st.update(tin[id]+no_root, tout[id], upd);
d6 cb     }

          // if no_root = 1, the root won't be included in the update;
72 6c     void update_path(int a, int b, SegTree::lazy upd, int no_root = 0){
b0 2d         while(head[a] != head[b]){
61 06             if(tin[a] < tin[b])
ec 25                 swap(a,b);
05 eb             st.update(tin[head[a]], tin[a], upd);
f4 1f             a = pai[head[a]];
a5 cb         }
fe a0         if(tin[a] > tin[b])
02 25             swap(a,b);
ae b2         st.update(tin[a]+no_root, tin[b], upd);
40 cb     }

a3 e6     seg query_point(int id){
f2 6f         return st.query(tin[id],tin[id]);
8a cb     }

          // if no_root = 1, the root won't be included in the query;
6f 30     seg query_subtree(int id, int no_root = 0){
85 82         return st.query(tin[id]+no_root,tout[id]);
56 cb     }

          // if no_root = 1, the root won't be included in the query;
          // this query will work even if the query is non commutative
ee 33     seg query_path(int a, int b, int no_root = 0){
ce 86         seg retl = seg(), retr = seg();

e4 2d         while(head[a] != head[b]){
62 4c             seg& ret = tin[a] > tin[b] ? retl : retr;
3a 33             int& v = tin[a] > tin[b] ? a : b;
90 6b             ret = st.merge(ret,st.query(tin[head[v]], tin[v]));
0c 58             v = pai[head[v]];
c6 cb         }

bd a0         if(tin[a] > tin[b])
1e 25             swap(a,b);

45 37         return st.merge(st.merge(retl,st.query(tin[a]+no_root,tin[b])), retr);

86 cb     }

02 bf     private:

9c c7     void prec(int id){
d6 5a         if(g[id].size() && g[id][0] == pai[id])
10 a8             swap(g[id][0], g[id].back());
76 20         for(int& v : g[id]){
e0 85             if(v == pai[id])
ab 5e                 continue;
eb 21             pai[v] = id;
4a 09             height[v] = height[id]+1;
33 f9             prec(v);
48 b0             sub[id]+=sub[v];
af df             if(sub[v] > sub[g[id][0]])
57 00                 swap(v,g[id][0]);
ec cb         }
62 cb     }
```

```
e8 a2     void hld(int id, int hd){
69 36         tout[id] = tin[id];
18 a6         head[id] = hd;
ad 5c         if(g[id].size() && g[id][0] != pai[id]){
e7 38             tin[g[id][0]] = tout[id]+1;
10 e2             hld(g[id][0],hd);
36 8a             tout[id] = tout[g[id][0]];
0d cb         }
69 8f         for(int i = 1; i < g[id].size(); i++){
3a 85             int v = g[id][i];
a0 85             if(v == pai[id])
29 5e                 continue;
1d bd             tin[v] = tout[id]+1;
e0 97             hld(v, v);
83 b1             tout[id] = tout[v];
7b cb         }
4b cb     }
02 21 };
```

6.10 IncrementalMst.hpp

```
Hash: be6a3a
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
9e ed #include"../utility/rng.hpp"
a8 f2 #endif

78 a6 struct IncrementalMST{
73 bc     vector<int> par, sz, prio;
80 3d     vector<pii> parw;


b2 0c     IncrementalMST(int n) : par(n), sz(n,1), prio(n), parw(n,{INF
    ,INF}){
54 6a         iota(all(prio),0);
28 bd         par = prio;
41 1e         shuffle(all(prio),rng);
37 cb     }
bf bf     private:

da 98     int find(int v, pii w = {INF-1,INF}){
a4 f9         while(parw[v] <= w){
3f e9             while(parw[v] > parw[par[v]]){
32 07                 sz[par[v]]-=sz[v];
54 43                 par[v] = par[par[v]];
24 cb             }
b9 c3             v = par[v];
33 cb         }
6f 6d         return v;
c3 cb     }

54 17     void disconnect(int v){
7e 71         if(par[v] == v)
ba 50             return;
36 0e         disconnect(par[v]);
45 07         sz[par[v]]-=sz[v];
04 cb     }

c1 cc     int connect(int v, pii w = {INF-1, INF}){
fe f9         while(parw[v] <= w){
da 3e             sz[par[v]]+=sz[v];
b1 c3             v = par[v];
80 cb         }
9d 6d         return v;
76 cb     }

64 0f     void consider_edge(int a, int b, pii w){
97 0a         disconnect(a), disconnect(b);

a2 98         while(a != b){
59 c8             a = connect(a,w);
08 eb             b = connect(b,w);
97 46             if(prio[a] > prio[b])
59 25                 swap(a,b);
8a de             swap(par[a],b);
a2 40             swap(parw[a],w);
70 cb         }

df 69         connect(a);
be cb     }

31 67     public:

          /**
           * Finds maximum edge in the path from a to b
           * @return weight of maximum edge from a to b (or {INF,-1} if
           * they are disconnected)
           */
29 4f     pii max_edge(int a, int b){
```

```
74 82         int ra = find(a), rb = find(b);
0d 7b         if(ra != rb)
9a 56             return {INF,-1};

43 6e         if(parw[a] > parw[b])
83 25             swap(a,b);
fb 02         while(par[a] != b){
9d 3a             a = par[a];
ff 6e             if(parw[a] > parw[b])
93 25                 swap(a,b);
85 cb         }
9a d6         return parw[a];
68 cb     }

          /**
           * Deletes maximum edge of the path from a to b
           * from the MST
           * @return weight of the edge removed from the MST (or {INF
               ,-1} if
           * they are disconnected)
           */
27 16     pii delete_maximum(int a, int b){
60 82         int ra = find(a), rb = find(b);
10 7b         if(ra != rb)
35 56             return {INF,-1};

43 6e         if(parw[a] > parw[b])
78 25             swap(a,b);
34 02         while(par[a] != b){
eb 3a             a = par[a];
86 6e             if(parw[a] > parw[b])
d8 25                 swap(a,b);
5b cb         }

02 d9         b = a;
6e ad         while(par[b] != b){
7d ac             sz[par[b]]-=sz[a];
89 08             b = par[b];
ca cb         }
b1 21         par[a] = a;
46 25         pii ret = {INF,INF};
01 fa         swap(parw[a],ret);
47 ed         return ret;
55 cb     }

          /**
           * Adds edge between a and b with weight w to the graph.
           * @return weight of the edge removed from the MST (or {INF
               ,-1} if
           * there was none)
           */
e4 27     pii add_edge(int a, int b, pii w){
b4 ae         if(a == b)
9c 56             return {INF,-1};

7b 67         pii ret = delete_maximum(a,b);
33 b6         if(ret <= w)
e7 1c             swap(w,ret);

ae cf         consider_edge(a,b,w);

6b ed         return ret;
01 cb     }
be 21 };
```

6.11 Isomorphism.hpp

```
Hash: 71fe5b
/*
from https://github.com/defnotmee/definitely-not-a-lib

Gives a way to hash a tree, either considering it rooted or not.
(choose the corresponding struct depending on the case)

Usage:

Rooted_Isomorphism(n, root) initializes the structure for a
tree of size n (0 indexed) rooted at root.

add_edge(a,b) is self explanatory

After adding all edges, call calc_tree() to get the hash of the tree.

After calling calc_tree(), hashsub[i] will contain the hash of subtree
    i.

For Unrooted_Isomorphism, the biggest difference is that the hashub
    array will
be meaningless.
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
```

```
91 f2 #endif

2b d5 const int SEED = chrono::high_resolution_clock::now().
    time_since_epoch().count();

05 84 struct Rooted_Isomorphism{

fa bd     int n, root;
65 ae     vector<int> tin, tout, sub, pai, height;
d6 3b     vector<basic_string<int>> g;
75 cb     int m = 0;
88 99     ull seed;
93 81     vector<ll> hashsub;

95 79     ull hasher(ull x){
              // http://xorshift.di.unimi.it/splitmix64.c
2a 6e         x+=0x9e3779b97f4a7c15;
c2 3e         x = (x^(x>>30)) * 0xbf58476d1ce4e5b9;
74 31         x = (x^(x>>27)) * 0x94d049bb133111eb;
19 10         return x^(x>>31)^seed;
05 cb     }

71 b1     Rooted_Isomorphism(int n = 0, int root = 0, ull seed = SEED)
      : n(n), root(root),
db 8d     tin(n), tout(n), sub(n,1), pai(n,root), height(n), g(n),
19 1f     seed(seed), hashsub(n) {}

          // use this if you want the same graph for a different root,
              otherwise important info wont be reset
f8 1e     Rooted_Isomorphism(Rooted_Isomorphism& r, int root) :
    Rooted_Isomorphism(r.n, root){
9f c9         m = r.m;
33 69         g = r.g;
3c cb     }

6a 01     void add_edge(int a, int b){
24 02         g[a].push_back(b);
28 3e         g[b].push_back(a);
27 7b         m++;
fc cb     }

          // returns hash of the whole tree
8c d9     ull calc_tree(){
97 9b         assert(m == n-1);
e2 00         prec(root);
fb 0d         return hashsub[root];
b3 cb     }
```

```
1c bf     private:

9a c7     void prec(int id){
fb 36         tout[id] = tin[id];
b7 81         for(int v : g[id]){
7f 85             if(v == pai[id])
04 5e                 continue;
38 21             pai[v] = id;
ff 09             height[v] = height[id]+1;
2f bd             tin[v] = tout[id]+1;
fb f9             prec(v);
d3 b1             tout[id] = tout[v];
37 b0             sub[id]+=sub[v];
e9 ff             hashsub[id]+=hashsub[v];
4d cb         }
50 06         hashsub[id] = hasher(hashsub[id]);
35 cb     }

9f 21 };

d7 50 struct Unrooted_Isomorphism{
15 40     Rooted_Isomorphism tree;

31 b6     Unrooted_Isomorphism(int n) : tree(n){}

59 01     void add_edge(int a, int b){
6d 3b         tree.add_edge(a,b);
59 cb     }

57 37     pii find_centroids(){
cc 96         int id = tree.root;

75 66         while(true){
2c a0             for(int v : tree.g[id]){
95 a3                 if(tree.pai[id] != v && tree.sub[v]*2 >= tree.n){
69 c4                     id = v;
44 20                     goto NEXT;
64 cb                 }
6a cb             }
cd c2             break;
36 8f             NEXT:;
0e cb         }
68 02         if(tree.sub[id]*2 == tree.n)
b0 83             return {tree.pai[id], id};
4a 70         return {id,id};
1e cb     }
```

```
a8 d9     ull calc_tree(){
df e2         tree.calc_tree();
37 9d         auto [c1, c2] = find_centroids();

28 99         tree = Rooted_Isomorphism(tree,c1);
dc 0f         ull tmp = tree.calc_tree();

38 f9         tree = Rooted_Isomorphism(tree,c2);

15 b6         return min(tmp, tree.calc_tree());
76 cb     }
71 21 };
```

6.12 Kthshortestwalk.hpp

```
Hash: b77c3c
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Copied from https://judge.yosupo.jp/submission/246093
 * Thanks nor!!
 */
d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

78 4f template <class T>
6d b2 using min_heap = priority_queue<T, vector<T>, greater<T>>;

d5 65 template <class Distance, class Graph>
f3 7b auto dijkstra(const Graph& g, int s) {
db ee     vector<Distance> d(g.size(), numeric_limits<Distance>::max())
   ;
69 d5     vector<Distance> prv(g.size(), -1);
09 3e     min_heap<pair<Distance, int>> heap;
70 1c     heap.emplace(d[s] = 0, s);
61 74     while (!heap.empty()) {
50 41         auto [dv, v] = heap.top();
08 77         heap.pop();
24 05         if (dv != d[v]) continue;
5b e1         for (auto&& [to, w] : g[v]) {
e5 a3             if (d[to] > dv + w) {
51 bc                 d[to] = dv + w;
5b 12                 heap.emplace(d[to], to);
e8 ac                 prv[to] = v;
96 cb             }
7f cb         }
a6 cb     }
47 97     return make_pair(d, prv);
91 cb }

0f 6b template <typename Key, typename Value>
11 a6 struct LeftistHeap {
2e 2a     using self_t = LeftistHeap<Key, Value>;
ba 59     int rank;
18 e5     Key key;
cb 1a     Value value;
f0 88     self_t *left, *right;
e5 52     constexpr LeftistHeap(int rank_, Key key_, Value value_,
    self_t* left_,
0c 40                           self_t* right_)
1e d0         : rank{rank_}, key{key_}, value{value_}, left{left_},
    right{right_} {}
67 3a     inline static deque<LeftistHeap> alloc;
52 23     static self_t* insert(self_t* const a, const Key k, const
    Value v) {
08 ca         if (not a or k <= a->key) {
9b 55             alloc.emplace_back(1, k, v, a, nullptr);
d1 14             return &alloc.back();
28 cb         }
0f a0         auto l = a->left, r = insert(a->right, k, v);
7a 46         if (not l or r->rank > l->rank) swap(l, r);
46 2e         alloc.emplace_back(r ? r->rank + 1 : 0, a->key, a->value,
     l, r);
fb 14         return &alloc.back();
65 cb     }
ea 21 };

1a 52 template <typename Distance, typename Graph>
9e 2d auto kth_shortest_paths(int n, const Graph& g, int source, int
    sink, int k) {
56 6f     Graph g_rev(n);
d1 5b     for (int u = 0; u < n; ++u)
9a d2         for (auto [v, w] : g[u]) g_rev[v].push_back({u, w});
fb 77     auto [d, prv] = dijkstra<Distance>(g_rev, sink);
97 60     if (d[source] == numeric_limits<Distance>::max())
42 0c         return vector<Distance>{};
1b ca     vector<basic_string<int>> tree(n);
b9 5b     for (int u = 0; u < n; ++u)
24 6e         if (prv[u] != -1) tree[prv[u]].push_back(u);
ff 83     using heap_t = LeftistHeap<Distance, int>;
5b d4     vector<heap_t*> h(n, nullptr);
```

```cpp
71 f9     {
f7 94         queue<int> q({sink});
00 99         while (not q.empty()) {
3c c0             const int u = q.front();
ba 83             q.pop();
66 c1             bool seen_p = false;
80 1d             for (const auto [v, w] : g[u]) {
ee fa                 if (d[v] == numeric_limits<Distance>::max()) continue;
30 dd                 const auto c = w + d[v] - d[u];
df 24                 if (not seen_p and v == prv[u] and c == 0) {
ab 63                     seen_p = true;
6a 5e                     continue;
c4 cb                 }
14 fa                 h[u] = heap_t::insert(h[u], c, v);
95 cb             }
08 8f             for (auto v : tree[u]) h[v] = h[u], q.push(v);
69 cb         }
d2 cb     }
93 61     vector<Distance> ans{d[source]};
2a 05     ans.reserve(k);
8f 5b     if (not h[source]) return ans;
03 f9     {
38 2f         min_heap<pair<Distance, heap_t*>> q;
1b 29         q.push({d[source] + h[source]->key, h[source]});
2a 33         while (not q.empty() and (int) ans.size() < k) {
58 35             auto [cd, ch] = q.top();
32 83             q.pop();
a2 d8             ans.push_back(cd);
7a 30             if (h[ch->value]) q.push({cd + h[ch->value]->key, h[ch->value]});
c8 95             if (ch->left) q.push({cd + ch->left->key - ch->key, ch->left});
87 52             if (ch->right) q.push({cd + ch->right->key - ch->key, ch->right});
8c cb         }
94 cb     }
f3 ba     return ans;
b7 cb }
```

6.13 Lca.hpp

Hash: f5e683

```cpp
/*
from https://github.com/defnotmee/definitely-not-a-lib

Extension of tree_rooted.hpp that calculates lca in
O(nlogn) precomputation and O(1) per query.

Isnt able to calculate things on the path to the LCA.
(see binlift.hpp for that)
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
4f f4 #include"rooted_tree.hpp"
6d f2 #endif

f8 ae struct LCATree : Tree {
5d cf     vector<int> euler, eid;
88 77     matrix<int> sparse;

4e 9b     LCATree(int n = 0, int root = 0) : Tree(n, root), eid(n) {
7f ed         euler.reserve(2*n);
93 cb     }

a2 08     int get_lower(int a, int b){
98 d0         return height[a] < height[b] ? a : b;
79 cb     }

b7 11     void calc_tree(){
ad 9b         assert(m == n-1);
bb 00         prec(root);

              // not on rooted_tree.hpp
16 d4         int lg = log2(euler.size())+1;
44 18         sparse = matrix<int>(lg, euler);
c3 82         for(int i = 1; i < lg; i++){
a3 84             for(int j = 0; j + (1<<i) <= euler.size(); j++)
a3 ed                 sparse[i][j] = get_lower(sparse[i-1][j], sparse[i-1][j+(1<<i-1)]);
e6 cb         }
6d cb     }

56 7b     int lca(int a, int b){
e7 a0         a = eid[a], b = eid[b];
12 f7         if(a > b)
a7 25             swap(a,b);
b3 33         int logg = log2(b-a+1);
af 1e         return get_lower(sparse[logg][a], sparse[logg][b-(1<<logg)+1]);
9f cb     }
1b b5     int dist(int a, int b){
f7 c5         return height[a]+height[b]-2*height[lca(a,b)];
d4 cb     }

ef bf     private:

42 c7     void prec(int id){
8d 36         tout[id] = tin[id];
b8 43         eid[id] = euler.size(); // not on rooted_tree.hpp
e9 09         euler.push_back(id); // not on rooted_tree.hpp
1e 81         for(int v : g[id]){
c5 85             if(v == pai[id])
71 5e                 continue;
cb 21             pai[v] = id;
6c 09             height[v] = height[id]+1;
97 bd             tin[v] = tout[id]+1;
ef f9             prec(v);
97 b1             tout[id] = tout[v];
69 b0             sub[id]+=sub[v];
79 09             euler.push_back(id); // not on rooted_tree.hpp
ec cb         }
43 cb     }
f5 21 };
```

6.14 Lct.hpp

```
Hash: cf66bb
/**
 * from https://github.com/defnotmee/definitely-not-a-lib
 *
 * Implementation from https://codeforces.com/blog/entry/75885
 * Will implement it myself eventually but will just put it here
 * until I do.
 *
 * Different from other algos on this library, it is 1-INDEXED!!!
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

44 b8 struct SplayTree {
7c bf     struct Node {
0e e0       int ch[2] = {0, 0}, p = 0;
64 bf       ll self = 0, path = 0;        // Path aggregates
5e 6d       ll sub = 0, vir = 0;          // Subtree aggregates
ab d2       bool flip = 0;                       // Lazy tags
36 21     };
26 3e     vector<Node> T;

6a 98     SplayTree(int n) : T(n + 1) {}

7e 6c     void push(int x) {
76 e9       if (!x || !T[x].flip) return;
80 57       int l = T[x].ch[0], r = T[x].ch[1];

1c cd       T[l].flip ^= 1, T[r].flip ^= 1;
0d a1       swap(T[x].ch[0], T[x].ch[1]);
bf 4d       T[x].flip = 0;
76 cb     }

f0 42     void pull(int x) {
59 5b       int l = T[x].ch[0], r = T[x].ch[1]; push(l); push(r);

81 63       T[x].path = T[l].path + T[x].self + T[r].path;
50 0d       T[x].sub = T[x].vir + T[l].sub + T[r].sub + T[x].self;
de cb     }

0d 21     void set(int x, int d, int y) {
72 1a       T[x].ch[d] = y; T[y].p = x; pull(x);
6f cb     }

3d 07     void splay(int x) {
40 d0       auto dir = [&](int x) {
3c 06         int p = T[x].p; if (!p) return -1;
df 8d         return T[p].ch[0] == x ? 0 : T[p].ch[1] == x ? 1 : -1;
2a 21       };
43 0a       auto rotate = [&](int x) {
f9 07         int y = T[x].p, z = T[y].p, dx = dir(x), dy = dir(y);
28 47         set(y, dx, T[x].ch[!dx]);
28 52         set(x, !dx, y);
8e 75         if (~dy) set(z, dy, x);
f8 3f         T[x].p = z;
74 21       };
bb 22       for (push(x); ~dir(x); ) {
fa 02         int y = T[x].p, z = T[y].p;
71 5d         push(z); push(y); push(x);
55 8c         int dx = dir(x), dy = dir(y);
48 30         if (~dy) rotate(dx != dy ? x : y);
eb 64         rotate(x);
0e cb       }
88 cb     }
e2 21   };
```

```
37 6f   struct LinkCut : SplayTree {
ee e5     LinkCut(int n) : SplayTree(n) {}

b7 ac     int access(int x) {
16 16       int u = x, v = 0;
10 3a       for (; u; v = u, u = T[u].p) {
89 6d         splay(u);
eb 11         int& ov = T[u].ch[1];
4a 8d         T[u].vir += T[ov].sub;
33 4f         T[u].vir -= T[v].sub;
db e1         ov = v; pull(u);
72 cb       }
d2 97       return splay(x), v;
8f cb     }

5a d9     void reroot(int x) {
be ef       access(x); T[x].flip ^= 1; push(x);
2d cb     }

f7 c0     void Link(int u, int v) {
22 a2       reroot(u); access(v);
9d a8       T[v].vir += T[u].sub;
77 fb       T[u].p = v; pull(v);
b0 cb     }

38 ad     void Cut(int u, int v) {
0a a2       reroot(u); access(v);
59 5c       T[v].ch[0] = T[u].p = 0; pull(v);
52 cb     }

          // Rooted tree LCA. Returns 0 if u and v arent connected.
32 4c     int LCA(int u, int v) {
55 60       if (u == v) return u;
cd 84       access(u); int ret = access(v);
11 d0       return T[u].p ? ret : 0;
62 cb     }

          // Query subtree of u where v is outside the subtree.
6c a0     ll Subtree(int u, int v) {
e5 89       reroot(v); access(u); return T[u].vir + T[u].self;
83 cb     }

          // Query path [u..v]
cc 5e     ll Path(int u, int v) {
0b b0       reroot(u); access(v); return T[v].path;
1f cb     }

          // Update vertex u with value v
b8 41     void Update(int u, ll v) {
52 ec       access(u); T[u].self += v; pull(u);
a7 cb     }
cf 21 };
```

6.15 RootedTree.hpp

```
Hash: 221a00
/*
from https://github.com/defnotmee/definitely-not-a-lib

Stores a rooted tree with relevant information like height,
dfs order (tin and tout), height, the parent (pai) the size of the
subtrees (sub).

Intended to be inherited or composed for other algos.

Usage:

Tree(n,root): prepares tree of size n with vertices from 0 to n-1

add_edge(a,b): adds edge between a and b

After adding all edges, call calc_tree().
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
91 f2 #endif

46 5a struct Tree{
97 bd     int n, root;
49 ae     vector<int> tin, tout, sub, pai, height;
95 3b     vector<basic_string<int>> g;
35 cb     int m = 0;

a4 3d     Tree(int n = 0, int root = 0) : n(n), root(root),
35 1d     tin(n), tout(n), sub(n,1), pai(n,root), height(n), g(n){}

ce 01     void add_edge(int a, int b){
6e 02         g[a].push_back(b);
94 3e         g[b].push_back(a);
34 7b         m++;
35 cb     }
2b 11     void calc_tree(){
07 9b         assert(m == n-1);
d6 00         prec(root);
21 cb     }

          // call only after calc_tree
3b 37     pii find_centroids(){
45 8e         int id = root;

47 66         while(true){
bc 81             for(int v : g[id]){
52 e2                 if(pai[id] != v && sub[v]*2 >= n){
67 c4                     id = v;
d9 20                     goto NEXT;
ef cb                 }
7c cb             }
50 c2             break;
0c 8f             NEXT:;
5e cb         }
d7 f3         if(sub[id]*2 == n)
5f b4             return {pai[id], id};
4e 70         return {id,id};
bf cb     }

f9 d9     protected:
0c c7     void prec(int id){
44 36         tout[id] = tin[id];
0d 81         for(int v : g[id]){
44 85             if(v == pai[id])
47 5e                 continue;
b9 21             pai[v] = id;
de 09             height[v] = height[id]+1;
33 bd             tin[v] = tout[id]+1;
19 f9             prec(v);
21 b1             tout[id] = tout[v];
03 b0             sub[id]+=sub[v];
14 cb         }
b9 cb     }
22 21 };
```

6.16 Scc.hpp

```
Hash: 6fd52b
/*
from https://github.com/defnotmee/definitely-not-a-lib

Implements kosaraju's algorithm for finding strongly connected
components.

Usage:
SCC(n) : prepares graph of size n with vertices from 0 to n-1
add_edge(a,b) : adds directed edge from a to b

After adding all the edges, call kosaraju().

This call will make SCC::scc have information
on the strongly connected components:

(I) 0 <= scc[i] < scc_count
(II) scc[i] = scc[j] <=> there is a path from i to j and from j to i.
(III) scc[i] < scc[j] => there is no path from j to i. [bonus from
    kosaraju!]

get_condensation() will return a graph of the scc's (condensation graph
    ).
It will be a DAG!

fun fact: if you want to dp in the condensation graph you don't need to
     dfs,
you can just process the sccs in **descending** order because of
    property (III)!
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif

28 bf struct SCC{

e1 1a     int n;
70 47     vector<basic_string<int>> g, r;

af 1b     vector<int> scc;
ef 0b     int scc_count = 0;

ce 20     SCC(int n = 0) : n(n), g(n), r(n), scc(n,-1){}

76 01     void add_edge(int a, int b){
51 02         g[a].push_back(b);
d8 7c         r[b].push_back(a);
83 cb     }
8d db     void kosaraju(){
15 f9         vector<int> check(n);

3f 51         vector<int> euler;
3b 06         euler.reserve(n);

a1 83         for(int i = 0; i < n; i++)
4d 9f             if(!check[i]) dfs(i,check,euler);

d0 6b         reverse(all(euler));

4f f1         for(int i : euler)
c2 0e             if(check[i] == 1) rdfs(i,check), scc_count++;
d5 cb     }

6c 36     struct Condensation{
ed 1a         int n; // number of nodes
ad 43         int sn; // number of sccs
21 3b         vector<basic_string<int>> g; // Edges going out of the
    scc
ea 18         vector<basic_string<int>> in_scc; // List of vertices in
    scc[i]

08 e4         Condensation(int n, int sn) : n(n), sn(sn), g(sn), in_scc
    (sn){};
75 21     };

8e c5     Condensation get_condensation(){
a8 1a         if(scc.back() == -1)
cc 75             kosaraju();

d3 10         Condensation ret(n,scc_count);

ec 60         for(int i = 0; i < n; i++){
82 a1             ret.in_scc[scc[i]].push_back(i);
2d 48             for(int j : g[i]){
39 95                 if(scc[j] != scc[i])
59 f0                     ret.g[scc[i]].push_back(scc[j]);
cc cb             }
52 cb         }

              // comment if you dont care about repeated edges
11 a6         for(int i = 0; i < scc_count; i++){
d9 31             sort(all(ret.g[i]));
b8 26             ret.g[i].erase(unique(all(ret.g[i])),ret.g[i].end());
4f cb         }
e7 ed         return ret;
83 cb     }

89 bf     private:

0d 4f     void dfs(int id, vector<int>& check, vector<int>& euler){
85 e8         check[id] = 1;
bb 54         for(int i : g[id])
4c 34             if(!check[i])
64 c3                 dfs(i,check,euler);
1c 09         euler.push_back(id);
7f cb     }

82 ed     void rdfs(int id, vector<int>& check){
3a d1         scc[id] = scc_count;
9b a1         check[id] = 2;
0d d0         for(int i : r[id])
32 9a             if(check[i] == 1)
de 17                 rdfs(i,check);
8c cb     }

6f 21 };
```

6.17 UnionFind.hpp

```
Hash: 5c4f1c
/*
from https://github.com/defnotmee/definitely-not-a-lib

Disjoint Set Union with union by size and path compression. Complexity
    is O(n*inverse_ackermann(n)), where n is the number of updates.

Use the "size" and "pai" functions to get the size of the group and the
    parent of the current vertex.
*/

d7 d7 #ifndef O_O
99 6d #include"../utility/template.cpp"
e9 f2 #endif


de 60 class UnionFind{
4d 99     vector<int> v; // Either parent (if v[i] >= 0) or size (if v[
    i] < 0 and i is a root) of the component

d5 67     public:
4a 92     UnionFind(int n = 0) : v(n,-1){}
0a 13     int find(int id){
fd e1         return v[id] < 0 ? id : v[id] = find(v[id]);
db cb     }

73 34     int size(int id){ // Returns size of the component id belongs
     to
bc 93         return -v[find(id)];
c4 cb     }

04 f1     int pai(int id){ // Returns parent of id
01 0c         return v[id] < 0 ? id : v[id];
59 cb     }

          // Returns 1 if a and b were in different groups.
          // Useful for Kruskal.
e4 c8     bool onion(int a, int b){
6e bc         a = find(a);
4a b8         b = find(b);

59 ae         if(a == b)
eb bb             return 0;

83 ad         if(size(a) > size(b)) // union by size
e8 25             swap(a,b);

              // b will now be the parent of a

cb 72         v[b] += v[a];
58 4c         v[a] = b;
6e 6a         return 1;
01 cb     }

b7 3d     bool same(int a, int b){
68 c0         return find(a) == find(b);
00 cb     }
5c 21 };
```