# Definitely Not A Lib

Leonardo Valente Nascimento

29 de agosto de 2024

## Índice

# 1 utility

## 1.1 Hash.sh

```
Hash: 3692ba

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
3e 8f   hp=$(echo $(bash hash_file.sh pref.txt 1 1000) | cut -c-2)
ed 48   hl=$(echo $(bash hash_file.sh line.txt 1 1000) | cut -c-2)
58 ae   echo -e "$hp $hl $l"
36 65 done < "$1"
```

## 1.2 HashFile.sh

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

d4 d4 # Para usar (hash das linhas [l1, l2]):
d4 d4 # bash hash.sh arquivo.cpp l1 l2
d7 d7 sed -n $2','$3' p' $1 | sed '/^#w/d' | cpp -dD -P -fpreprocessed
   | tr -d '[:space:]' | md5sum | cut -c-6
```

## 1.3 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")
```

## 1.4 Stress.sh

```
Hash: 43ceff
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     ./$1 < input > outWrong
7a d2     if !(cmp -s "outWrong" "outSlow")
61 0e     then
85 75         echo "Error found!"
66 62         echo "Input:"
7d c5         cat input
4c 98         echo "Wrong Output:"
04 a2         cat outWrong
92 97         echo "Slow Output:"
4e a8         cat outSlow
d3 f2         exit
02 75     fi
bb d6     echo Passed Test:$testNum
a7 6b done
43 1b echo Passed $4 tests
```

## 1.5 Template.cpp

```
Hash: 78440e
/*
by Leonardo Valente Nascimento

My beautiful template :D
*/

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);



97 bb     return 0;

78 cb }
```

# 2 geometry

## 2.1 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 };
```

# 3 data structures

## 3.1 Bit.hpp

```
Hash: 1ca18a
/*
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     }

03 5f     inline void merge(T& a, T b){
ec 9f         a+=b;
5a cb     }

7e 7e     void update(int id, T x){
04 ab         id++;
17 b8         while(id < bit.size()){
82 00             merge(bit[id],x);
84 36             id+=id&-id;
21 cb         }
16 cb     }

50 32     T query(int id){
3b ab         id++;
e6 83         T ret = T();
2a 7a         while(id){
0d df             merge(ret,bit[id]);
6e 29             id-=id&-id;
55 cb         }
1f ed         return ret;
82 cb     }
1c 21 };
```

## 3.2 CartesianTree.hpp

```
Hash: 39e403
/*
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         }
02 cb     }
39 21 };
```

### 3.3 Hashmap.hpp

```
Hash: 80a779
/*
from https://github.com/defnotmee/definitely-not-a-lib

Unordered map with strong hash.
*/

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


5e a2 struct Hasher{
87 bc     ull operator()(ull x){
              // http://xorshift.di.unimi.it/splitmix64.c
78 6e         x+=0x9e3779b97f4a7c15;
88 3e         x = (x^(x>>30)) * 0xbf58476d1ce4e5b9;
31 31         x = (x^(x>>27)) * 0x94d049bb133111eb;
78 e3         return x^(x>>31)^RANDOM; // for random seed, delete if
    lazy
fa cb     }
96 21 };

80 da using hashmap = unordered_map<ull,Hasher>;
```

### 3.4 IndexedSet.hpp

```
Hash: 461dc5
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>;
```

### 3.5 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 };
```

## 3.6 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], then 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 last 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()){
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 };
```

## 3.7 SegtreeIterative.hpp

```
Hash: 342a6a
/*
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     }

f4 d5     inline seg merge(seg a, seg b){
20 df         return {a+b}; // here is where 2 nodes are merged
cd cb     }
9e 40     void update(int id, seg val){
96 92         id+=sz;

3b ae         tree[id] = val; // here is where you update a point

1d 77         id>>=1;

a5 7a         while(id){
f8 da             tree[id] = merge(tree[2*id], tree[2*id+1]);
20 77             id>>=1;
e1 cb         }
2e cb     }

70 0d     seg query(int l, int r){
1d ed         l += sz;
93 c0         r += sz+1;

2e 86         seg retl = seg(), retr = seg();

b5 40         while(l < r){
96 1f             if(l&1)
df 06                 retl = merge(retl, tree[l++]);
dc 84             if(r&1)
7d b3                 retr = merge(tree[--r], retr);

31 45             l>>=1;
a1 e9             r>>=1;
16 cb         }

4a 5a         return merge(retl,retr);
e0 cb     }

34 21 };
```

## 3.8 SegtreeLazy.hpp

```
Hash: 5eab6e
// TODO: Make build accept elements of type seg like
    iterativesegtree.hpp

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

Declaration: SegTree(size)
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]

==============================================================================

If a lazy segtree is not needed I recommend going for an
    iterativesegtree.hpp .
You can erase the parts where it does lazy propagation also.

Segtree for affine transformations and range sums in O(log(n)).
Made to be as customizable and copy-pasteable as possible, speed
and code size is not a concern.

0-indexed by default.

The parts you'll commonly edit will be commented.
*/

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

0a 38 struct SegTree{
de 8b     struct seg{
d6 00         ll x = 0; // "identity value" of the operation
98 21     };

3b 07     struct lazy{
be 7d         ll mult = 1, add = 0; // "identity value" of lazy tag

              // only for C++20, get fucked (implement your own ==)
                  otherwise
4b ba         auto operator<=>(const lazy& a) const = default;

              // Here is where you edit how to propagate the lazy tag
                  for the children
              // of a segtree node
20 ef         void operator+=(const lazy& a){
39 76             add*=a.mult;
2a d4             mult*=a.mult;
31 29             add+=a.add;
7d cb         }
00 21     };

cb df     vector<seg> tree; // yes, this is all for the pun
ac 2c     vector<lazy> lz;

3b 40     int sz, ql, qr;
51 a5     lazy val;

          // Here is where you change how to merge nodes
35 d5     inline seg merge(seg a, seg b){
43 6b         return {a.x+b.x};
23 cb     }

c8 23     SegTree(int n = 0){
68 43         sz = n;
f2 a7         tree = vector<seg>(4*n);
97 73         lz = vector<lazy>(4*n);
a7 cb     }


          // Comment the next two functions if you dont need a O(n)
              builder

be 7e     void build(int id, int l, int r, vector<ll> & v){
77 89         if(l == r){
7a 5f             tree[id].x = {v[l]};
d8 50             return;
ce cb         }

57 08         const int e = id*2+1, d = id*2+2, m = (l+r)>>1;

91 f7         build(e,l,m,v);
a2 30         build(d,m+1,r,v);
33 72         tree[id] = merge(tree[e],tree[d]);

36 cb     }

63 85     SegTree(vector<ll> v){
6b 46         *this = SegTree(v.size());
ab 49         build(0,0,sz-1,v);
cb cb     }

ca 8d     void refresh(int id, int l, int r){
b6 64         if(lz[id] == lazy())
d9 50             return;

e4 57         if(l != r){
b5 08             const int e = id*2+1, d = id*2+2, m = (l+r)>>1;

ff c0             lz[e]+=lz[id];
f4 b6             lz[d]+=lz[id];
16 cb         }

              // Here is where you update the value of the current
                 node based on the lazy tag
74 ae         tree[id] = {tree[id].x*lz[id].mult+lz[id].add*(r-l+1)};
25 b0         lz[id] = lazy();
d6 cb     }

83 51     void update(int l, int r, lazy x){
a4 40         ql = l, qr = r, val = x;

22 8b         upd(0,0,sz-1);
77 cb     }

3c 0d     seg query(int l, int r){
73 e2         ql = l, qr = r;

01 b0         return qry(0,0,sz-1);
ab cb     }

ec bf     private:

01 bf     void upd(int id, int l, int r){
73 a7         refresh(id,l,r);

fa ce         if(ql <= l && r <= qr){
b4 3f             lz[id] += val;
fe a7             refresh(id,l,r);
5a 50             return;
c0 cb         }
15 87         if(ql > r || l > qr)
db 50             return;

55 08         const int e = id*2+1, d = id*2+2, m = (l+r)>>1;

06 b7         upd(e,l,m);
dd ad         upd(d,m+1,r);

99 72         tree[id] = merge(tree[e], tree[d]);
74 cb     }

cb 31     seg qry(int id, int l, int r){
7f a7         refresh(id,l,r);

e4 43         if(ql <= l && r <= qr)
2b c9             return tree[id];

42 87         if(ql > r || l > qr)
3c 88             return seg();

d3 08         const int e = id*2+1, d = id*2+2, m = (l+r)>>1;
10 c3         return merge(qry(e,l,m), qry(d,m+1,r));
76 cb     }
5e 21 };
```

### 3.9 SegtreePersistent.hpp

```
Hash: eddea9
/*
from https://github.com/defnotmee/definitely-not-a-lib

Quite slow but at least it doesn't have memory leaks :D

To make 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

// Uncomment if you need a custom struct. Also construct a segtree
   using PSegTree<seg>
      // struct seg{

// };

43 fc template<typename seg = ll>
57 8a struct SegNode {
90 32     using sp = shared_ptr<SegNode<seg>>;
11 1e     auto make(SegNode<seg> cur = {}){
```

```
8d ff         return make_shared<SegNode<seg>>(cur);
33 cb     }

68 fc     sp e = nullptr, d = nullptr;
d8 52     seg x = 0;

dc 2e     SegNode(seg _x = 0) : x(_x){};

9a 67     SegNode(SegNode<seg>& b) : e(b.e), d(b.d), x(b.x){};

d5 85     static seg merge(seg e, seg d){
a1 48         return e+d;
92 cb     }

e2 99     constexpr seg get(){
d2 88         return this ? x : seg();
0b cb     }

dd 40     void update(ll l, ll r, ll target, seg val){
e2 89         if(l == r){
5a d1             x = val;
b3 50             return;
83 cb         }

cf ae         if(!e)
e3 ab             e = make(), d = make();

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

17 5d         if(target <= m)
d1 17             (e = make(*e))->update(l,m,target,val);
85 e4         else (d = make(*d))->update(m+1,r,target,val);

f7 5e         x = merge(e->get(), d->get());
b5 cb     }

b0 04     seg query(ll l, ll r, ll ql, ll qr){
67 ce         if(ql <= l && r <= qr){
fa ea             return x;
e8 cb         }
90 87         if(ql > r || l > qr)
6d 88             return seg();

59 ae         if(!e)
dd ab             e = make(), d = make();

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

85 32         return merge(e->query(l,m,ql,qr),
50 80             d->query(m+1,r,ql,qr));

e5 cb     }

28 21 };

66 fc template<typename seg = ll>
3f e0 struct PSegTree{
3d 1e     auto make(SegNode<seg> cur = {}){
86 ff         return make_shared<SegNode<seg>>(cur);
7c cb     }

de bb     shared_ptr<SegNode<seg>> head;
36 f8     ll l, r;

8f 79     PSegTree(ll _l, ll _r) : l(_l), r(_r), head(new
    SegNode<seg>){};

29 78     void update(ll id, seg x){
fa 2a         (head = make(*head))->update(l,r,id,x);
bc cb     }

97 00     seg query(ll ql, ll qr){
db 57         return head->query(l,r,ql,qr);
e9 cb     }
ed 21 };
```

## 3.10 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 };
```

# 4 math

## 4.1 BasicCombi.hpp

```
Hash: 5e7c0f
/*
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

40 e3 template<ull M = MOD>
e7 d8 struct Combi{
c4 a7     using mint = modint<M>;

          // note that inv[0] = 1 in this impl
0f b5     vector<mint> fac, inv, invfac;
8e bc     Combi(int n = MAXN){
fc 47         fac = inv = invfac = vector<mint>(n,1);

d7 9f         for(int i = 2; i < n; i++){
e8 ba             fac[i] = fac[i-1]*i;
07 6f             inv[i] = inv[M%i]*(M-M/i);
8c 15             invfac[i] = invfac[i-1]*inv[i];
5c cb         }
2a cb     }

55 e6     mint choose(int n, int k){
a4 37         if(n < k)
12 bb             return 0;
c9 07         return fac[n]*invfac[k]*invfac[n-k];
36 cb     }
23 21 };

5e fa Combi c;
```

## 4.2 Beegmod.hpp

```
Hash: e8e14a
/*
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){
7a 5a     ull ret = a*b - m*ull(dbll(a)*b/m);
f4 9d     return min({ret,ret+m,ret-m});
89 cb }

67 b3 ull inverse(ull a, ull m){
a3 00     complex<ull> ca{1,0}, cb{0,1};

11 39     while(a){
71 1d        ull curdiv = a/m;
a8 ba         ca-=cb*curdiv;
d9 2e         a-=m*curdiv;
fa 6a         swap(a,m);
7e c3         swap(ca,cb);
8c cb     }

02 7e     return min(cb.real(), -cb.real());
35 cb }

d4 0d ull divmul(ull a, ull b, ull m){
c4 a5     return modmul(a,inverse(b,m),m);
07 cb }

27 5b ull power(ull in, ull exp, ull m){
b1 cc     ull ret = 1;
97 fb     while(exp){
22 87         if(exp&1)
7f a0             ret = modmul(ret,in,m);
44 3d         in = modmul(in,in,m);
be ef         exp>>=1;
b8 cb     }
27 ed     return ret;
e8 cb }
```

## 4.3 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 }
```

## 4.4 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 }
```

## 4.5 FactoringAndPrimalityTest.hpp

```
Hash: 007569
/*
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/num
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
5b 7c #include"beegmod.hpp"
b0 f2 #endif

6f 87 bool is_prime(ull n){
7d c9     if(n <= 1)
8f d1         return false;

21 b1     ull ctz = countr_zero(n-1);
25 70     ull d = n>>ctz;

0d 0e     auto primes = {2, 3, 5, 13, 19, 73, 193, 407521, 299210837};
          // all primes up to 37 is a reasonable option too
b1 a7     auto bases = {2, 325, 9375, 28178, 450775, 9780504,
   1795265022};

ce ce     for(ull p : primes)
9f e7         if(n == p)
a8 6a             return 1;

15 e4     for(ull base : bases){
08 0c         ull cur = power(base,d,n);
e9 66         if(cur == 1)
da 5e             continue;
02 2b         for(int i = 0; i < ctz; i++){
68 56             if(cur == n-1)
b7 20                 goto NEXT;
5b 1f             cur = modmul(cur,cur,n);
d1 cb         }
c6 d1         return false;
d2 8f         NEXT:;
93 cb     }

bb 8a     return true;

e6 cb }

13 67 template<typename T>
a8 cc void pollard(T n, vector<T>& v){
c9 e7     if(n == 1)
9d 50         return;
da a7     if(is_prime(n)){
75 3b         v.push_back(n);
88 50         return;
19 cb     }

d7 dd     static mt19937_64 rng(RANDOM);
3f dc     uniform_int_distribution<T> rg(0,n-1);
f2 fb     T c = rg(rng);
56 64     T x, y;
67 e8     x = y = rg(rng);

22 ce     auto next = [&](T x){
d6 fd         return modadd(modmul(x,x,n),c,n);
71 21     };

d4 a4     T prod = 2;
09 aa     T g = 1;
0a 8f     while((g = gcd(prod,n)) == 1){
3a ac         for(int i = 0; i < 50; i++){
db 2a             if(x == y)
4d 1f                 x = y = rg(rng), c = rg(rng);
b8 53             x = next(x);
49 1d             y = next(next(y));
20 80             ll cur = modmul(abs(x-y),prod,n);
f4 69             if(cur)
08 27                 prod = cur;
0a cb         }
66 cb     }

80 36     pollard(g,v);
25 6c     pollard(n/g,v);
a1 cb }

63 67 template<typename T>
43 4f vector<T> factorize(T n, bool sorted = 0){
7c 5a     vector<T> ret;
```

```
98 0e     pollard(n,ret);

64 64     if(sorted)
96 2f         sort(all(ret));

55 ed     return ret;
00 cb }
```

## 4.6 Fft.hpp

```
Hash: 5ed994
/*
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/num
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
5a 0d #include"../modint.hpp"
38 f2 #endif

85 d5 using cdl = complex<dbll>;
a7 23 using cd = complex<double>; // change this to long double if WA
   and pray

8c ec void fft(vector<cd>& v, bool inverse = 0){
b3 3d     int n = v.size();
fb 77     int lg = log2(n);

ef 8a     static vector<cdl> loots;
27 dc     static vector<cd> roots;

2b 27     if(loots.size() < n){
07 cb         loots.resize(n,1);
2e 7b         roots.resize(n,1);
26 cb     }
```

```
65 89     for(static int len = 2; len < n; len<<=1){
52 d8         cdl z = polar(1.0l, acos(-1.0l)/len);
4a d8         for(int i = len; i < 2*len; i++){
23 00             roots[i] = loots[i] = loots[i/2] * ((i&1) ? z : 1);
7b cb         }
d8 cb     }

3a 80     vector<int> rev(n);

07 6f     for(int i = 1; i < n; i++){
8a 56         rev[i] = (rev[i>>1]>>1)+((i&1)<<lg-1);
bb fc         if(rev[i] > i)
c8 60             swap(v[i],v[rev[i]]);
2a cb     }

4d 9b     for(int len = 1; len < n; len<<=1){
02 27         for(int block = 0; block < n; block+=2*len){
d6 5d             for(int l = block; l < block+len; l++){
db c4                 cd cur = roots[l-block+len]*v[l+len];
a5 d8                 tie(v[l], v[l+len]) =
a2 f6                     make_pair(v[l]+cur, v[l]-cur);
cc cb             }
cc cb         }
1c cb     }

f4 1f     if(inverse){
a0 83         reverse(1+all(v));
b1 2d         for(auto& i : v)
9e c4             i/=n;
dd cb     }
c2 cb }


4e f1 vector<ll> convolution(vector<ll>& a, vector<ll>& b){
ae 59     int mx = max(a.size(),b.size());
21 43     int n = 1;

aa 0c     while(n+1 < a.size()+b.size())
79 c1         n<<=1;

29 b0     vector<cd> in(n);

8b 43     for(int i = 0; i < a.size(); i++)
62 0a         in[i].real(a[i]);
2e c0     for(int i = 0; i < b.size(); i++)
eb f4         in[i].imag(b[i]);
```

```
6c 21     fft(in);

15 4d     vector<cd> newin(n);

2c 60     for(int i = 0; i < n; i++){
7f d6         int opos = (n-i)&(n-1);
52 2c         newin[i] = (in[opos]+conj(in[i]))
db 24         *(in[opos]-conj(in[i]))*cd(0, -0.25/n);
22 cb     }

72 1e     fft(newin);

c9 8a     vector<ll> ret(a.size()+b.size()-1);
cd 2a     for(int i = 0; i < a.size()+b.size()-1; i++){
21 f6         ret[i] = round(newin[i].real());
7b cb     }

b7 ed     return ret;

c0 cb }

8e 5e vector<cd> convolution(vector<cd> a, vector<cd> b){
f9 59     int mx = max(a.size(),b.size());
2b f7     int rets = a.size()+b.size()-1;
4b 43     int n = 1;

bd 0c     while(n+1 < a.size()+b.size())
69 c1         n<<=1;

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

63 0f     fft(a), fft(b);

15 60     for(int i = 0; i < n; i++){
97 db         a[i]*=b[i];
53 cb     }

8a c3     fft(a,1);
65 68     a.resize(rets);

ee 3f     return a;
7e cb }


04 e3 template<ull M = MOD>
97 e2 vector<modint<M>> convolutionmod(vector<modint<M>>& a,
    vector<modint<M>>& b){
```

```
36 57     const int len = sqrt(M);
f8 43     int n = 1;
94 0c     while(n+1 < a.size()+b.size())
1b c1         n<<=1;

4a 53     vector<cd> ca(n), cb(n);

17 43     for(int i = 0; i < a.size(); i++)
4d 76         ca[i] = cd(a[i].x%len, a[i].x/len);

ea c0     for(int i = 0; i < b.size(); i++)
c0 ed         cb[i] = cd(b[i].x%len, b[i].x/len);

0f ec     fft(ca), fft(cb);

5c 52     vector<cd> p1(n), p2(n);

b7 60     for(int i = 0; i < n; i++){
b4 d6         int opos = (n-i)&(n-1);

              // also inverting for fft inverse
4e b7         p1[i] = (ca[opos]+conj(ca[i]))*cb[opos]*cd(0.5/n);
ca 79         p2[i] = (ca[opos]-conj(ca[i]))*cb[opos]*cd(0,-0.5/n);
58 cb     }

a4 bb     fft(p1), fft(p2);

01 ee     vector<modint<M>> ret(a.size()+b.size()-1);

a2 9c     for(int i = 0; i < ret.size(); i++){
1e c5         modint<M> small = round(p1[i].real()),
0f 4d             mid = (ll)round(p1[i].imag()) +
   (ll)round(p2[i].real()),
1e 71             big = round(p2[i].imag());

29 5a         ret[i] = small + mid*len + big*len*len;

20 cb     }

2d ed     return ret;
5e cb }
```

## 4.7 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 };
```

## 4.8 Modint.hpp

Hash: 5ff539

```
/*
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{
c2 43     const static ull MOD = M; // in case we need to use it
   somewhere else (for example, combi.cpp)

4a 0e     ull x;

          // It is assumed -M <= v. Extra mod is taken for safety.
f7 ab     constexpr modint(ll v = 0) : x((v+M)%M){};

2b 0f     constexpr modint(ll v, ll raw) : x(v){};

          // only on C++20
b5 8b     bool operator<=>(const modint&) const = default;

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

f0 1c     modint operator+(modint b) const{
26 eb         return modint(min(x+b.x, x+b.x-M),1);
7d cb     }

7f d7     modint operator-(modint b) const{
d0 f1         return modint(min(x-b.x, x-b.x+M),1);
7e cb     }

15 ac     modint operator*(modint b) const {
28 20         return modint((x*b.x%M),1);
80 21     };

5b 2f     modint inverse(){
```

```
c1 26         ll x = this->x, y = M;

7c ca         complex<ll> cx = {1,0}, cy = {0,1};

21 f4         while(x){
04 41             ll curdiv = y/x;
ee 61             y-=curdiv*x;
96 eb             cy-=curdiv*cx;
e1 0e             swap(cx, cy);
e8 9d             swap(x, y);
29 cb         }

11 8c         return modint(cy.real());
a3 cb     }

e5 e3     modint operator/(modint b) const {
fd 78         return *this*b.inverse();
cc cb     }

ef 34     void operator+=(modint b){
5a 4f         x = min(x+b.x, x+b.x-M);
80 cb     }

e0 cc     void operator-=(modint b){
3c 60         x = min(x-b.x, x-b.x+M);
23 cb     }

3e 41     void operator*=(modint b){
98 76         (x*=b.x)%=M;
50 cb     }

d2 92     void operator/=(modint b){
1b 7d         *this = *this/b;
91 cb     }
5d 21 };

5f 9a using mint = modint<MOD>;
```

## 4.9 Sieve.hpp

```
Hash: b72835
/*
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     }
b7 c0 } sieve(MAXN);
```

# 5 strings

## 5.1 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 };
```

## 5.2 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 };
```

## 5.3 Kmp.hpp

```
Hash: 6a1da2
/*
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>
2c 3d vector<int> kmp(T s){
23 27     vector<int> pi(s.size());
a8 88     for(int i = 1; i < s.size(); i++){
48 8d         pi[i] = pi[i-1];
aa e3         while(pi[i] != 0 && s[pi[i]] != s[i]){
67 77             pi[i] = pi[pi[i]-1];
ea cb         }
bd 18         pi[i]+=s[i]==s[pi[i]];
0b cb     }
b1 81     return pi;
6a cb }
```

## 5.4 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 };
```

## 5.5 Trie.hpp

```
Hash: 7db66d
/*
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     }

ed bc     void insert(string& s, int ct = 1){
fc 04         int id = 0;
60 be         int pos = 0;
ac 51         while(pos < s.size()){
2a 72             char cur = s[pos]-INI;
78 42             if(trie[id].ptr[cur] == -1)
cb a3                 trie[id].ptr[cur] = trie.size(),
    trie.push_back({});
a8 c2             trie[id].sub+=ct;
de 8a             id = trie[id].ptr[cur];
f8 65             pos++;
35 cb         }
a5 c2         trie[id].sub += ct;
62 9a         trie[id].term += ct;
fd cb     }

e9 a7     int find(string& s){
6e 43         int id = 0, pos = 0;
85 51         while(pos < s.size()){
b6 72             char cur = s[pos]-INI;
a6 42             if(trie[id].ptr[cur] == -1)
23 da                 return -1;
3d 8a             id = trie[id].ptr[cur];
d6 65             pos++;
f8 cb         }
```

```
8a 64         return id;
d4 cb     }
7d 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: ea22eb
/*
from https://github.com/defnotmee/definitely-not-a-lib

Given an array of ancestors (next), 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(next): constructs the structure. next 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

e4 71 struct BinLift{

92 8b     int n, lg;

2b 6b     struct lift{
59 70         int to = 0;
a7 21     };

          // what happens when you go through a, and then go through b?
04 4e     static lift merge(lift a, lift b){
be 72         return {b.to};
fb cb     }

e9 50     matrix<lift> jmp;

91 72     BinLift(vector<int> next) : n(next.size()), lg(1){

8e f1         for(int tmp = 1; tmp < n; tmp*=2, lg++);

ab e4         jmp = matrix<lift>(lg,vector<lift>(next.size()));

              // initialize jmp[0][i]
73 e8         for(int i = 0; i < next.size(); i++)
8a 76             jmp[0][i] = {next[i]};


0e 82         for(int i = 1; i < lg; i++){
cd 1a             for(int j = 0; j < next.size(); j++){
7d 52                 jmp[i][j] = merge(jmp[i-1][j],
    jmp[i-1][jmp[i-1][j].to]);
7f cb             }
60 cb         }
b5 cb     }

66 fe     lift k_up(int id, int k){
9c 51         lift ret{id}; // needs to be an identity element through
    merge
cc 95         while(k){
57 3e             ret = merge(ret, jmp[__builtin_ctz(k)][ret.to]);
8f ab            k-=k&-k;
a0 cb        }
d4 ed        return ret;
21 cb    }

13 b2    lift lca(int a, int b, vector<int>& h){
8d be        if(h[a] < h[b])
bf 25            swap(a,b);

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

1c 97        if(la.to == lb.to)
f4 c9            return la;

bf 35        for(int i = lg-1; i >= 0; i--){
70 7e            if(jmp[i][la.to].to != jmp[i][lb.to].to)
6c 4c                la = merge(la,jmp[i][la.to]), lb = merge(lb,jmp[i][lb.to]);
c0 cb        }

93 d4        la = merge(la, jmp[0][la.to]);
c0 04        lb = merge(lb, jmp[0][lb.to]);


c2 91        return merge(la,lb);
5f cb    }

ea 21 };
```

## 6.3 Dinic.hpp

```
Hash: f5d86f
/*
from https://github.com/defnotmee/definitely-not-a-lib

Uses Dinic's algorithm to calculate the maximum flow between
s and t in a graph.

O(V^2E) in general, O(Esqrt(V)) on unit networks (edges that are
not connected to s or t have unit capacity, like in matching).

Usage: Declare FlowGraph(n,s,t) and add edges to it. When done, call
max_flow(). It returns the maximum flow between s and t. By default,
s = 0 and t = n-1.

After calling max_flow, the edges with EVEN indices on FlowGraph::edges
will have the "flow" variable corresponding to the ammount of flow passing
through them in the answer dinic provides.
*/

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

39 6b struct FlowEdge {
85 fb     ll u, v, cap, flow = 0;

55 de     ll to(ll id){
c7 b1         return id == u ? v : u;
19 cb     }
bf 21 };

3a 2b struct FlowGraph{
1a 1a     int n;
73 d1     int s, t;
55 bd     vector<FlowEdge> edges;

8b 13     vector<bstring<int>> g;

07 f8     FlowGraph(int n = 0, int _s = 0, int _t = -1) : n(n), s(_s), t(_t), g(n){
db 6a         if(t == -1)
2e 51             t = n-1;
7c cb     }

0b 5b     void add_edge(ll u, ll v, ll cap){
78 44         g[u].push_back(edges.size());
ad c4         edges.push_back({u,v,cap});
5e 81         g[v].push_back(edges.size());
04 4b         edges.push_back({v,u,0});
d5 cb     }

68 2a     ll max_flow(){
62 b7         ll ret = 0;
31 66         while(true){
24 a2             ll cur = block_flow();
55 b3             if(!cur)
ce c2                 break;
ff 2e             ret+=cur;
37 cb         }
86 ed         return ret;
36 cb     }

52 bf     private:
14 c7     vector<int> ptr, dist;
f5 30     ll block_flow(){
58 b7         ll ret = 0;
92 8b         dist = bfs();
5c d9         ptr = vector<int>(n);
d4 32         return dfs(s,INFL); // INFL needs to be >= than the max flow of the graph
44 cb     }

14 02     vector<int> bfs(){
50 8a         vector<int> dist(n,n);

e3 26         queue<int> q;
8d a9         dist[s] = 0;
ed 08         q.push(s);

78 14         while(!q.empty()){
cf 69             int cur = q.front();
b8 83             q.pop();
05 19             for(int eid : g[cur]){
1a 4c                 FlowEdge cedge = edges[eid];
2d a3                 int to = cedge.to(cur);

64 89                 if(cedge.cap == cedge.flow)
b0 5e                     continue;

0d 03                 if(dist[to] > dist[cur]+1){
82 d2                     dist[to] = dist[cur]+1;
51 91                     q.push(to);
3f cb                 }
e5 cb             }
4c cb         }

fd 8d         return dist;
2f cb     }

a0 1a     ll dfs(int id, ll pushed){
1f 15         if(pushed == 0)
a8 bb             return 0;
e5 bf         if(id == t)
12 44             return pushed;

16 38         ll rem = pushed;
60 3e         while(rem && ptr[id] < g[id].size()){
9e 4f             int eid = g[id][ptr[id]];
31 61             int to = edges[eid].to(id);
18 69             ptr[id]++;

0a 5b             if(dist[id] >= dist[to])
01 5e                 continue;

2d 5a             ll usable = min(rem, edges[eid].cap-edges[eid].flow);
2d 7b             ll used = dfs(to,usable);

43 db             edges[eid].flow+=used;
9c 28             edges[eid^1].flow-=used;

11 94             rem-=used;
cd cb         }
1e fc         return pushed-rem;
93 cb     }
f5 21 };
```

## 6.4 DsuRollback.hpp

```
Hash: 2bee4e
/*
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){ // Only call when id is the root
    of a group. Use size(find(id)) otherwise.
6e e0         return -v[id];
6b cb     }

b4 96     constexpr int pai(int id){ // Returns parent of id
26 0c         return v[id] < 0 ? id : v[id];
d7 cb     }


a5 13     int find(int id){ // removing path compression
b7 a4         return v[id] < 0 ? id : find(v[id]);
45 cb     }

0d c8     bool onion(int a, int b){
0a bc         a = find(a);
1d b8         b = find(b);

59 ae         if(a == b)
4d bb             return 0;

b6 ad         if(size(a) > size(b)) // union by size
2e 25             swap(a,b);

17 4c         comp_ct--;
2a 69         history.push_back({a,b,v[a],v[b]});
67 72         v[b] += v[a];
cc 4c         v[a] = b;
f8 6a         return 1;
fd cb     }

bb 5c     void rollback(){
12 d5         auto [a,b,va,vb] = history.back();
a4 8b         v[a] = va;
5c 99         v[b] = vb;
8d 2c         comp_ct++;
90 7d         history.pop_back();
29 cb     }

aa 3d     bool same(int a, int b){
5f c0         return find(a) == find(b);
e7 cb     }

11 cd     constexpr int snapshot(){
06 53         return history.size();
ed cb     }
```

```
2b 21 };
```

## 6.5 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.
*/

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 };
```

## 6.6 FunctionalGraph.hpp

```
Hash: 9b6a1b
/*
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;
76 c7     vector<bstring<int>> rev, clist;

17 f6     FuncGraph(vector<int> v) : n(v.size()), pai(v), height(n),
1c 5a     rev(n), cycleid(n,-1), cyclepos(n), clist(n), is_cycle(n), tin(n), tout(n){
31 83         for(int i = 0; i < n; i++)
ec 3a             rev[pai[i]].push_back(i);

76 60         for(int i = 0; i < n; i++){
38 f4             if(cycleid[i] == -1)
73 bc                 get_cycle(i);
67 cb         }
60 cb     }

14 d0     void get_cycle(int id){
17 5b         int a = id, b = id;

b0 01         do{
c1 5a             a = pai[a];
26 57             b = pai[pai[b]];
a7 54         } while(a != b);

d1 5f         process_cycle(a);
dc cb     }

6d 97     void process_cycle(int id){
80 e9         int cid = cycleid[id] = id;


4a 02         int v = id;
b8 01         do{
7a b5             cyclepos[v] = clist[cid].size();
26 89             clist[cid].push_back(v);
6d 15             is_cycle[v] = 1;
bf 90             v = pai[v];
4a 5a             cycleid[v] = cid;
5d 81         } while(v != id);

d8 01         do{
ba 6b             dfs(v);
c2 90             v = pai[v];
```

```
af 81         } while(v != id);

23 cb     }

70 26     void dfs(int id){
65 36         tout[id] = tin[id];
69 c6         for(int i : rev[id]){
9f 75             if(cycleid[i] == -1){
b4 24                 cycleid[i] = cycleid[id];
44 68                 cyclepos[i] = cyclepos[id];
12 db                 height[i] = height[id]+1;
46 7b                 tin[i] = ++tout[id];
6f 1e                 dfs(i);
1e e6                 tout[id] = tout[i];
48 cb             }
71 cb         }
7b cb     }

          // returns directed distance from a to b, or INF if its not possible to go from a to b
d0 b5     int dist(int a, int b){
57 f4         if(cycleid[a] != cycleid[b])
b0 cd             return INF;
f2 5f         if(is_cycle[a] && !is_cycle[b])
4b cd             return INF;
84 e7         if(!is_cycle[a] && !is_cycle[b]){
4f e4             if(height[a] < height[b] || cyclepos[a] != cyclepos[b])
ab cd                 return INF;
ef 17             if(tin[b] <= tin[a] && tin[a] <= tout[b]){
d5 91                 return height[a]-height[b];
aa cb             }
a0 cd             return INF;
12 cb         }

31 53         return height[a]+dist_in_cycle(cyclepos[a], cyclepos[b], clist[cycleid[a]].size());
ab cb     }

4e bf     private:

db 9b     int dist_in_cycle(int a, int b, int csize){
cb 7e         if(b >= a)
34 49             return b-a;
f2 03         return csize+b-a;
7c cb     }
```

```
9b 21 };
```

## 6.7 Hld.hpp

```
Hash: deda28
/*
from https://github.com/defnotmee/definitely-not-a-lib
*/

d7 d7 #ifndef O_O
cb c8 #include"../../utility/template.cpp"
4f f4 #include"rooted_tree.hpp"
1f 8c #include"../../data structures/segtree_lazy.hpp"
7b f2 #endif

8f 69 struct HLD : Tree {
ec bf     private:
59 e0     SegTree st;
d7 31     vector<int> head;

0f 67     public:

0e 6d     HLD(int n, int root = 0) : Tree(n, root), st(n), head(n) {}

bb 11     void calc_tree(){
eb 9b         assert(m == n-1);
f8 00         prec(root);
ae 7d         hld(root,root);
61 cb     }

21 62     void calc_tree(vector<ll>& v){
d4 6e         calc_tree();
91 ae         vector<ll> v2(n);
b5 83         for(int i = 0; i < n; i++)
4c 16             v2[tin[i]] = v[i];
39 c9         st = SegTree(v2);
d6 cb     }

11 7b     int lca(int a, int b){
d2 2d         while(head[a] != head[b]){
ad 06             if(tin[a] < tin[b])
8f 25                 swap(a,b);
22 1f             a = pai[head[a]];
be cb         }
62 9b         return min(a,b,[&](int a, int b){
1a db             return tin[a] < tin[b];
57 c0         });
5b cb     }

ba b5     int dist(int a, int b){
d6 c5         return height[a] + height[b] - 2*height[lca(a,b)];
2b cb     }

a0 82     using lazy = SegTree::lazy;
c4 e9     using seg = SegTree::seg;

44 f5     void update_point(int id, SegTree::lazy upd){
e7 9c         st.update(tin[id], tin[id], upd);
4e cb     }

          // if no_root = 1, the root won't be included in the update;
fd d4     void update_subtree(int id, SegTree::lazy upd, int no_root =
    0){
fe 58         st.update(tin[id]+no_root, tout[id], upd);
d8 cb     }

          // if no_root = 1, the root won't be included in the update;
7a 6c     void update_path(int a, int b, SegTree::lazy upd, int
    no_root = 0){
40 2d         while(head[a] != head[b]){
3f 06             if(tin[a] < tin[b])
7f 25                 swap(a,b);
ce eb             st.update(tin[head[a]], tin[a], upd);
90 1f             a = pai[head[a]];
ab cb         }
f8 a0         if(tin[a] > tin[b])
99 25             swap(a,b);
d2 b2         st.update(tin[a]+no_root, tin[b], upd);
54 cb     }

e9 e6     seg query_point(int id){
7a 6f         return st.query(tin[id],tin[id]);
a4 cb     }

          // if no_root = 1, the root won't be included in the query;
4b 30     seg query_subtree(int id, int no_root = 0){
ab 82         return st.query(tin[id]+no_root,tout[id]);
c3 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
92 33     seg query_path(int a, int b, int no_root = 0){
28 86         seg retl = seg(), retr = seg();
41 2d         while(head[a] != head[b]){
a0 4c             seg& ret = tin[a] > tin[b] ? retl : retr;
1b 33             int& v = tin[a] > tin[b] ? a : b;
3b 6b             ret = st.merge(ret,st.query(tin[head[v]], tin[v]));
36 58             v = pai[head[v]];
e3 cb         }

2b a0         if(tin[a] > tin[b])
b2 25             swap(a,b);

f5 37         return
    st.merge(st.merge(retl,st.query(tin[a]+no_root,tin[b])), retr);

51 cb     }

45 bf     private:

cc c7     void prec(int id){
              // tout[id] = tin[id];
33 5a         if(g[id].size() && g[id][0] == pai[id]) // not on
    rooted_tree.hpp
19 a8             swap(g[id][0], g[id].back());// not on
    rooted_tree.hpp
55 20         for(int& v : g[id]){ // & not in rooted_tree.hpp
87 85             if(v == pai[id])
36 5e                 continue;
af 21             pai[v] = id;
5a 09             height[v] = height[id]+1;
                  // tin[v] = tout[id]+1;
d1 f9             prec(v);
                  // tout[id] = tout[v];
2b b0             sub[id]+=sub[v];
ed df             if(sub[v] > sub[g[id][0]]) // not on rooted_tree.hpp
89 00                 swap(v,g[id][0]); // not on rooted_tree.hpp
e4 cb         }
32 cb     }

25 a2     void hld(int id, int hd){
7f 36         tout[id] = tin[id];
83 a6         head[id] = hd;
60 5c         if(g[id].size() && g[id][0] != pai[id]){
19 38             tin[g[id][0]] = tout[id]+1;
08 e2             hld(g[id][0],hd);
c5 8a             tout[id] = tout[g[id][0]];
f9 cb         }
f5 8f         for(int i = 1; i < g[id].size(); i++){
6b 85             int v = g[id][i];
36 85             if(v == pai[id])
73 5e                 continue;
35 bd             tin[v] = tout[id]+1;
ce 97             hld(v, v);
df b1             tout[id] = tout[v];
db cb         }
40 cb     }
de 21 };
```

## 6.8 Isomorphism.hpp

```
Hash: af0415
/*
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"
4f f4 #include"rooted_tree.hpp"
6d f2 #endif

d8 50 struct Rooted_Isomorphism : Tree{

e3 99     ull seed;
5a 81     vector<ll> hashsub;
38 79     ull hasher(ull x){
              // http://xorshift.di.unimi.it/splitmix64.c
ca 6e         x+=0x9e3779b97f4a7c15;
e8 3e         x = (x^(x>>30)) * 0xbf58476d1ce4e5b9;
07 31         x = (x^(x>>27)) * 0x94d049bb133111eb;
47 10         return x^(x>>31)^seed;
b3 cb     }

ff 73     Rooted_Isomorphism(int n = 0, int root = 0, ull seed =
    RANDOM) : Tree(n,root), seed(seed), hashsub(n) {}

          // use this if you want the same graph for a different root,
              otherwise important info wont be reset
74 1e     Rooted_Isomorphism(Rooted_Isomorphism& r, int root) :
    Rooted_Isomorphism(r.n, root){
a1 c9         m = r.m;
1c 69         g = r.g;
dc cb     }

          // returns hash of the whole tree
8b d9     ull calc_tree(){
bc 9b         assert(m == n-1);
00 00         prec(root);
4e 0d         return hashsub[root];
78 cb     }

d0 bf     private:

0d c7     void prec(int id){
7e 36         tout[id] = tin[id];
32 81         for(int v : g[id]){
8c 85             if(v == pai[id])
08 5e                 continue;
14 21             pai[v] = id;
ec 09             height[v] = height[id]+1;
58 bd             tin[v] = tout[id]+1;
74 f9             prec(v);
a0 b1             tout[id] = tout[v];
d3 b0             sub[id]+=sub[v];
01 ff             hashsub[id]+=hashsub[v]; // not on rooted_tree.hpp
44 cb         }
dd 06         hashsub[id] = hasher(hashsub[id]); // not on
    rooted_tree.hpp
f4 cb     }

81 21 };
```

```
8e 50 struct Unrooted_Isomorphism{
df 40     Rooted_Isomorphism tree;

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

a3 01     void add_edge(int a, int b){
da 3b         tree.add_edge(a,b);
e5 cb     }

41 d9     ull calc_tree(){
0f e2         tree.calc_tree();
1a 17         auto [c1, c2] = tree.find_centroids();

7b 99         tree = Rooted_Isomorphism(tree,c1);
85 0f         ull tmp = tree.calc_tree();

36 f9         tree = Rooted_Isomorphism(tree,c2);

6c b6         return min(tmp, tree.calc_tree());
24 cb     }
af 21 };
```

## 6.9 Lca.hpp

```
Hash: f5e683
/*
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 };
```

## 6.10 RootedTree.hpp

```
Hash: b9cea6
/*
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;
fb 13     vector<bstring<int>> g;
75 cb     int m = 0;

e0 3d     Tree(int n = 0, int root = 0) : n(n), root(root),
25 1d     tin(n), tout(n), sub(n,1), pai(n,root), height(n), g(n){}

          // Takes a tree, changes the root and preprocesses it
8d af     Tree(Tree& t, int root) : Tree(t.n, root){
c4 9a         g = t.g;
c3 6e         calc_tree();
f2 cb     }
db 01     void add_edge(int a, int b){
7e 02         g[a].push_back(b);
f7 3e         g[b].push_back(a);
f9 7b         m++;
7c cb     }

fc 11     void calc_tree(){
69 9b         assert(m == n-1);
d7 00         prec(root);
9a cb     }

          // call only after calc_tree
2b 37     pii find_centroids(){
75 8e         int id = root;

d8 66         while(true){
57 81             for(int v : g[id]){
b7 e2                 if(pai[id] != v && sub[v]*2 >= n){
b3 c4                     id = v;
5e 20                     goto NEXT;
b0 cb                 }
88 cb             }
2a c2             break;
77 8f             NEXT:;
9d cb         }
5a f3         if(sub[id]*2 == n)
97 b4             return {pai[id], id};
e9 70         return {id,id};
6d cb     }

18 d9     protected:
a9 c7     void prec(int id){
ba 36         tout[id] = tin[id];
a7 81         for(int v : g[id]){
aa 85             if(v == pai[id])
9f 5e                 continue;
c1 21             pai[v] = id;
15 09             height[v] = height[id]+1;
5f bd             tin[v] = tout[id]+1;
50 f9             prec(v);
0b b1             tout[id] = tout[v];
41 b0             sub[id]+=sub[v];
0b cb         }
9e cb     }
b9 21 };
```

## 6.11 Scc.hpp

```
Hash: 470390
/*
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;
71 99     vector<bstring<int>> g, r;

7d 1b     vector<int> scc;
ba 0b     int scc_count = 0;

aa 20     SCC(int n = 0) : n(n), g(n), r(n), scc(n,-1){}
d8 01     void add_edge(int a, int b){
95 02         g[a].push_back(b);
05 7c         r[b].push_back(a);
64 cb     }

c5 db     void kosaraju(){
b3 f9         vector<int> check(n);

f6 51         vector<int> euler;
77 06         euler.reserve(n);

a3 83         for(int i = 0; i < n; i++)
d1 9f             if(!check[i]) dfs(i,check,euler);

f3 6b         reverse(all(euler));

f7 f1         for(int i : euler)
ee 0e             if(check[i] == 1) rdfs(i,check), scc_count++;
d0 cb     }

d2 36     struct Condensation{
d5 1a         int n; // number of nodes
ef 43         int sn; // number of sccs
ad 13         vector<bstring<int>> g; // Edges going out of the scc
cc 3f         vector<bstring<int>> in_scc; // List of vertices in scc[i]

f4 e4         Condensation(int n, int sn) : n(n), sn(sn), g(sn), in_scc(sn){};
d9 21     };

bf c5     Condensation get_condensation(){
36 1a         if(scc.back() == -1)
ef 75             kosaraju();

31 10         Condensation ret(n,scc_count);

6e 60         for(int i = 0; i < n; i++){
3d a1             ret.in_scc[scc[i]].push_back(i);
d9 48             for(int j : g[i]){
d9 95                 if(scc[j] != scc[i])
23 f0                     ret.g[scc[i]].push_back(scc[j]);
e9 cb             }
b2 cb         }

              // comment if you dont care about repeated edges
01 a6         for(int i = 0; i < scc_count; i++){
ae 31             sort(all(ret.g[i]));
d3 26             ret.g[i].erase(unique(all(ret.g[i])),ret.g[i].end());
7d cb         }
52 ed         return ret;
de cb     }

e4 bf     private:

c7 4f     void dfs(int id, vector<int>& check, vector<int>& euler){
e2 e8         check[id] = 1;
1b 54         for(int i : g[id])
f5 34             if(!check[i])
78 c3                 dfs(i,check,euler);
dc 09         euler.push_back(id);
9a cb     }

c6 ed     void rdfs(int id, vector<int>& check){
09 d1         scc[id] = scc_count;
f3 a1         check[id] = 2;
89 d0         for(int i : r[id])
6e 9a             if(check[i] == 1)
9b 17                 rdfs(i,check);
c3 cb     }

47 21 };
```

## 6.12 UnionFind.hpp

```
Hash: cc4e7b
/*
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


b2 0c struct UnionFind{
81 bf     private:
```

```
96 99     vector<int> v; // Either parent (if v[i] >= 0) or size (if v[i] < 0 and i is a root) of the component

bc 67     public:
14 92     UnionFind(int n = 0) : v(n,-1){}

01 a6     constexpr int size(int id){ // Only call when id is the root of a group. Use size(find(id)) otherwise.
fb e0         return -v[id];
04 cb     }

e6 96     constexpr int pai(int id){ // Returns parent of id
de 0c         return v[id] < 0 ? id : v[id];
9e cb     }

be 13     int find(int id){
52 e7         if(v[id] < 0)
f9 64             return id;
d2 48         return v[id] = find(v[id]);
38 cb     }

          // Returns 1 if a and b were in different groups.
          // Useful for Kruskal.
cf c8     bool onion(int a, int b){
7d bc         a = find(a);
23 b8         b = find(b);

96 ae         if(a == b)
51 bb             return 0;

e2 ad         if(size(a) > size(b)) // union by size
43 25             swap(a,b);

8c 72         v[b] += v[a];
2d 4c         v[a] = b;
63 6a         return 1;
a3 cb     }

e4 3d     bool same(int a, int b){
8b c0         return find(a) == find(b);
98 cb     }
cc 21 };
```