Chulalongkorn University

What Name ?

Pakapim E., Pasin P., Sarit P.

template from KACTL

2025-07-20

Template (1)

template.cpp

31 lines

```
#pragma once
#include <bits/stdc++.h>
#define sz(x) (int)(x).size()
#define all(x) (x).begin(), (x).end()

using namespace std;

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

template<typename T> bool ckmin(T &a, const T &b) { return b <
     a ? a = b, 1 : 0; }
template<typename T> bool ckmax(T &a, const T &b) { return a <
     b ? a = b, 1 : 0; }

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());

const char nl = '\n';
const int INF = 0x3fffffff;
const int MOD=1000000007;
// const int MOD = 998244353;
const ll LINF = 0x1fffffffffffffff;
const db DINF = numeric_limits<db>::infinity();
const db EPS = 1e-9;
const db PI = acos(db(-1));

signed main(){
  ios_base::sync_with_stdio(0); cin.tie(NULL);
  return 0;
}
```

c.sh

2 lines

```
g++ -std=gnu++2a -Wall $1 -o a.out
./a.out
```

Mathematics (2)

2.1 Goldbatch's Conjecture

- Even number can be written in sum of two primes (Up to $1e12$)
- Range of N^{th} prime and $N+1^{th}$ prime will be less than or equal to 300 (Up to $1e12$)

2.2 Divisibility

Number of divisors of N is given by $\prod_{i=1}^{k}(a_i+1)$ where $N=\prod_{i=1}^{k}p_i^{a_i}$ and p_i are prime factors of N.

Combinatorial (3)

3.1 Permutations

3.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800

n	11	12	13	14	15	16	17
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14

n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.

Time: $\mathcal{O}(n)$

044568, 6 lines

```
int permToInt(vi &v){
  int use = 0, i = 0, r = 0;
  for (int x : v) r = r * ++i + __builtin_popcount(use & -(1 <<
       x)),
    use |= 1 << x; // (note: minus, not ~!)
  return r;
}
```

3.1.2 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n\in S}\frac{x^n}{n}\right)$$

3.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor\frac{n!}{e}\right\rceil$$

3.1.4 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G=\mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k).$$

3.2 Partitions and subsets

3.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0)=1,\ p(n) = \sum_{k\in\mathbb{Z}\setminus\{0\}} (-1)^{k+1}p(n-k(3k-1)/2)$$

$$p(n)\sim 0.145/n\cdot\exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

3.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_kp^k+...+n_1p+n_0$ and $m = m_kp^k+...+m_1p+m_0$. Then $\binom{n}{m}\equiv\prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$.

3.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!...k_n!}$.

a0a312, 6 lines

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
}
```

3.3 General purpose numbers

3.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t)=\frac{t}{e^t-1}$ (FFT-able). $B[0,\ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k\cdot(n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^\infty f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$

$$\approx \int_m^\infty f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

3.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$
$$\textstyle\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

3.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

3.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

3.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod p$$

3.3.6 Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

3.3.7 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Data Structures (4)

OrderedSet.hpp

Description: Ordered Set

"../template/Header.hpp", <bits/extc++.h> 32a919, 15 lines

```
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
// can be change to less_equal / greater

void usage() {
  ordered_set<int> st, st_2;
  st.insert(2);
  st.insert(1);
  cout << st.order_of_key(2);
  cout << *st.find_by_order(1);
  st.erase(st.find_by_order(st.order_of_key(5)));
  st.join(st_2); // merge
}
```

SparseTable.hpp

Description: Sparse Table for finding static range queries of monoid operations (in this case min).
Memory: $\mathcal{O}(N\log N)$
Time: $\mathcal{O}(1)$ query, $\mathcal{O}(N\log N)$ construction.

eedab3, 18 lines

```
struct SparseTable{
  int n;
  vector<vector<int>> t;
  SparseTable() {}
  SparseTable(const vector<int> &a) {
    n=(int)a.size();
    int lg=31-__builtin_clz(n);
    t.assign(lg+1,vector<int>(n,1e9));
    t[0]=a;
    for(int i=0;i<lg;i++)
      for(int j=0;j+(2<<i)<=n;j++)
        t[i+1][j]=min(t[i][j],t[i][j+(1<<i)]);
    }
    int query(int l,int r){
      int lg=31-__builtin_clz(r-l+1);
      return min(t[lg][l], t[lg][r-(1<<lg)+1]);
    }
};
```

FenwickTree.hpp

Description: Fenwick / Binary Indexed Tree
Memory: $\mathcal{O}(N)$
Time: query and update $\mathcal{O}(\log N)$

069d13, 28 lines

```
struct fenwick {
  int n;
  vector<int> t;
  fenwick (int n=0) {
    n=_n;
    t.assign(n+1, T{});
  }
  void update(int x, int v) {
    for (int i = x; i <= n; i+=i&-i) t[i] = t[i]+v;
  }
  void update(int l, int r, int v) {
    update(l, v); update(r+1, -v);
  }
  int query(int x) {
    int res = 0;
    for (int i = x; i >= 1; i-=i&-i) res = res+t[i];
    return res;
}
  int query(int l, int r) { return query(r) - query(l-1); }
  //find the first index that sums to >= k
  int find(const T &k) {
    int x = 0;
    int cur = 0;
    for (int i = 1<<(31-__builtin_clz(n)); i > 0; i>>=1)
      if (x+i <= n && cur+t[x+i] < k) x+=i, cur+=t[x];
    return x;
  }
};
```

2DFenwickTree.hpp

Description: 2D Fenwick Tree. Requires that elements to be updated is known in advance. Note that X is not compressed.
Memory: $\mathcal{O}(N\log N)$ (i guess)
Time: query and update $\mathcal{O}(\log^2 N)$

8f7a1f, 28 lines

```
struct TD_fenwick {
  int n;
  vector<vector<int>> vals, t;
  TD_fenwick(int n, vector<pii> &v): n(n), vals(n+1), t(n+1) {
    for (int i = 1; i <= n; ++i) vals[i].push_back(0);
    sort(all(v), [&](pii a, pii b){ return a.second < b.second;
        });
    for (auto [x, y] : v) for (; x <= n; x += x&-x) if (vals[x
        ].back() != y) vals[x].push_back(y);
    for (int i = 1; i <= n; ++i) t[i].resize(vals[i].size()+3);
  }
  inline int cp(int i, int x) {
    return upper_bound(all(vals[i]), x) - vals[i].begin();
  }
  void update(int x, int y, int val) {
    for (int i = x; i <= n; i += i&-i) {
      for (int j = cp(i, y); j < t[i].size(); j += j&-j) t[i][j
          ] += val;
    }
  }
  int query(int x, int y) {
    int res = 0;
    for (int i = x; i >= 1; i -= i&-i) {
      for (int j = cp(i, y); j >= 1; j -= j&-j) res += t[i][j];
    }
    return res;
  }
  int query(int x1, int y1, int x2, int y2) {
    return query(x2, y2) - query(x2, y1-1) - query(x1, y2) +
        query(x1, y1-1);
```

```
  }
};
```

SegmentTree.hpp

Description: Segment Tree 9b83b9, 82 lines

```
template<class Node>
struct SegTree {
  int n;
  vector<Node> t;
  SegTree(){};
  SegTree(int n, Node v=Node()) {init(n, v);}
  template<class T>
  SegTree(const vector<T> &a) {init(a);}
  void init(int n, Node v=Node()) {init(vector<Node>(n, v));}
  template<class T>
  void init(const vector<T> &a) {
    n=sz(a);
    t.assign(4<<31-__builtin_clz(n),Node());
    function<void(int, int, int)> build=[&](int l, int r, int i
        ) {
      if (l==r) return void(t[i]=a[l-1]);
      int m = (l+r)/2;
      build(l, m, 2*i);
      build(m+1, r, 2*i+1);
      pull(i);
    };
    build(1, n, 1);
  }
  void pull(int i) {t[i] = t[2*i] + t[2*i+1];}
  void modify(int l, int r, int i, int x, const Node &v) {
    if (x<l || x>r) return;
    if (l==r) return void(t[i]=v);
    int m = (l+r)/2;
    if (x<=m) modify(l, m, 2*i, x, v);
    else modify(m+1, r, 2*i+1, x, v);
    pull(i);
  }
  void modify(int x, const Node &v) {modify(1, n, 1, x, v);}
  template<class T>
  void update(int l, int r, int i, int x, const T &v) {
    if (x<l || x>r) return;
    if (l==r) return void(t[i].apply(l, r, v));
    int m = (l+r)/2;
    if (x<=m) update(l, m, 2*i, x, v);
    else update(m+1, r, 2*i+1, x, v);
    pull(i);
  }
  template<class T>
  void update(int x, const T &v) {update(1, n, 1, x, v);}
  Node query(int l, int r, int i, int x, int y) {
    if (y<l || x>r) return Node();
    if (x<=l && r<=y) return t[i];
    int m = (l+r)/2;
    return query(l, m, 2*i, x, y) + query(m+1, r, 2*i+1, x,y);
  }
  Node query(int x, int y) {return query(1, n, 1, x, y);}
  template<class F>
  int findfirst(int l, int r, int i, int x, int y, const F &f)
      {
    if (y<l||r<x||!(f(t[i]))) return -1;
    if (l==r) return l;
    int m = (l+r)/2;
    int ret = findfirst(l, m, 2*i, x, y, f);
    if (ret == -1) ret = findfirst(m+1, r, 2*i+1, x, y, f);
    return ret;
  }
  template<class F>
  int findfirst(int x, int y, const F &f) {return findfirst(1,
      n, 1, x, y, f);}
  template<class F>
  int findlast(int l, int r, int i, int x, int y, const F &f) {
    if (y<l||r<x||!(f(t[i]))) return -1;
    if (l==r) return l;
    int m = (l+r)/2;
    int ret = findlast(m+1, r, 2*i+1, x, y, f);
    if (ret == -1) ret = findlast(l, m, 2*i, x, y, f);
    return ret;
  }
  template<class F>
  int findlast(int x, int y, const F &f) {return findlast(1, n,
       1, x, y, f);}
};

struct Node {
  ll val;
  Node(ll x=LLONG_MAX):val(x){}
  void apply(int l, int r, int x) {val = x;}
  friend Node operator+(const Node &lhs, const Node &rhs) {
    return Node(min(lhs.val, rhs.val));
  }
};
```

LazySegmentTree.hpp

Description: Segment Tree with Lazy Propagation 90a572, 81 lines

```
template<class Node, class Tag>
struct LazySegTree{
  int n;
  vector<Node> t;
  vector<Tag> lz;
  LazySegTree() {}
  LazySegTree(int n, Node v=Node()) {init(n, v);}
  template<class T>
  LazySegTree(const vector<T> &a) {init(a);}
  void init(int n, Node v=Node()) {init(vector<Node>(n, v));}
  template<class T>
  void init(const vector<T> &a) {
    n=sz(a);
    t.assign((4<<(31-__builtin_clz(n)))+1, Node());
    lz.assign((4<<(31-__builtin_clz(n)))+1, Tag());
    function<void(int, int, int)> build=[&](int l, int r, int i
        ){
      if (l==r) return void(t[i]=a[l-1]);
      int m=(l+r)/2;
      build(l,m,2*i);
      build(m+1,r,2*i+1);
      pull(i);
    };
    build(1,n,1);
  }
  void pull(int i) {
    t[i]=t[2*i]+t[2*i+1];
  }
  void apply(int l,int r,int i,const Tag &v) {
    t[i].apply(l,r,v);
    lz[i].apply(l,r,v);
  }
  void push(int l, int r, int i) {
    int m=(l+r)/2;
    apply(l,m,2*i, lz[i]);
    apply(m+1,r,2*i+1, lz[i]);
    lz[i] = Tag();
  }
  void modify(int l, int r, int i, int x, const Node &v) {
    if (x<l||x>r) return;
    if (l==r) return void(t[i]=v);
    int m=(l+r)/2;
    push(l,r,i);
    modify(l,m,2*i,x,v);
    modify(m+1,r,2*i+1,x,v);
    pull(i);
  }
  void modify(int x, const Node &v) { modify(1,n,1,x,v); }
  void update(int l, int r, int x, int y, int i, const Tag &v)
      {
    if (y<l||x>r) return;
    if (x<=l&&r<=y) return apply(l, r, i, v);
    int m=(l+r)/2;
    push(l, r, i);
    update(l, m, x, y, 2*i, v);
    update(m+1, r, x, y, 2*i+1, v);
    pull(i);
  }
  void update(int x, int y, const Tag &v) { update(1, n, x, y,
      1, v); }
  void update(int x, const Tag &v) { update(1, n, x, x, 1, v);
      }
  Node query(int l, int r, int x, int y, int i) {
    if (y<l || x>r) return Node();
    if (x<=l&&r<=y) return t[i];
    int m=(l+r)/2;
    push(l, r, i);
    return query(l, m, x, y, 2*i)+query(m+1, r, x, y, 2*i+1);
  }
  Node query(int x, int y) { return query(1, n, x, y, 1); }
  Node query(int x) { return query(1, n, x, x, 1); }
};
struct Tag{
  ll val;
  Tag(ll _val=0):val(_val){}
  void apply(int l, int r, const Tag &v) { val+=v.val; }
};
struct Node{
  ll val;
  Node(ll _val=0):val(_val){}
  void apply(int l, int r, const Tag &v) { val+=v.val; }
  friend Node operator+(const Node &lhs, const Node &rhs) {
    return Node(max(lhs.val, rhs.val));
  }
};
```

DynamicSegmentTree.hpp

Description: Dynamic Segment Tree 587ad0, 66 lines

```
const int MX = 1e5 + 3;

struct Node {
  ll cnt;
  ll sum;
  Node *lc, *rc;
  Node() {
    sum = cnt = 0;
    lc = rc = nullptr;
  }
  void createChild() {
    if (lc == nullptr) lc = new Node();
    if (rc == nullptr) rc = new Node();
  }
};

Node *root = new Node();

void update(ll l, ll r, ll x, ll val, Node *cur) {
  if (l==r) {
    cur -> cnt += val;
```

```
    cur -> sum += x*val;
    return;
  }
  ll mid = (l+r)>>1;
  cur -> createChild();
  if (x <= mid) update(l, mid, x, val, cur -> lc);
  else update(mid+1, r, x, val, cur -> rc);

  cur->sum = cur->lc->sum + cur->rc->sum;
  cur->cnt = cur->lc->cnt + cur->rc->cnt;
}

// Query function for the Dynamic Segment Tree
pair<ll, ll> query(ll l, ll r, ll ql, ll qr, Node *cur) {
  if (cur == nullptr || ql > r || qr < l) {
    return {0, 0}; // {count, sum}
  }

  if (ql <= l && r <= qr) {
    return {cur->cnt, cur->sum};
  }

  ll mid = (l + r) >> 1;
  cur->createChild();

  auto left_result = query(l, mid, ql, qr, cur->lc);
  auto right_result = query(mid + 1, r, ql, qr, cur->rc);

  return {left_result.first + right_result.first,
          left_result.second + right_result.second};
}

// Wrapper function for easier usage
pair<ll, ll> query(ll ql, ll qr) {
  return query(0, MX - 1, ql, qr, root);
}

ll search(ll l, ll r, ll lsum, ll rfreq, ll k, Node *cur) {
  if (l==r) return l;
  ll mid = (l+r)>>1;
  cur -> createChild();
  __int128_t cntMid = lsum + (__int128_t)(mid+1)*rfreq + (cur->
      lc->sum) + (__int128_t)(mid+1)*(cur->rc->cnt);
  if (cntMid < (__int128_t)(mid+1)*k) return search(l, mid,
      lsum, rfreq + cur->rc->cnt, k, cur -> lc);
  else return search(mid+1, r, lsum + cur->lc->sum, rfreq, k,
      cur -> rc);
}
```

PersistentSegmentTree.hpp
Description: Persistent Segment Tree

"../template/Header.hpp" af7fe8, 60 lines

```
const int N = 1<<18;

struct node {
    int lc, rc;
    ll val;
} seg[20*N];
int root[N], cnt, a[N];

void update(int &cur, int l, int r, int x, ll val) {
    cnt++;
    seg[cnt] = seg[cur];
    cur = cnt;

    if (l==r) {
        seg[cur].val = val;
        return;
    }
    int mid = (l+r)/2;
    if (x <= mid) {
        update(seg[cur].lc, l, mid, x, val);
    }
    else {
        update(seg[cur].rc, mid+1, r, x, val);
    }
    seg[cur].val = seg[seg[cur].lc].val + seg[seg[cur].rc].val;
}

ll query(int cur, int l, int r, int ql, int qr) {
    if (r < ql || l > qr) return 0;
    if (ql <= l && r <= qr) return seg[cur].val;

    int mid = (l+r)/2;
    return query(seg[cur].lc, l, mid, ql, qr) + query(seg[cur].
        rc, mid+1, r, ql, qr);
}

void solve() {
    int n, q; cin >> n >> q;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
        update(root[1], 1, n, i, a[i]);
    }

    int sz = 1;

    while (q--) {
        int op; cin >> op;
        if (op == 1) {
            ll k, a, x; cin >> k >> a >> x;
            update(root[k], 1, n, a, x);
        }
        if (op == 2) {
            ll k, a, b; cin >> k >> a >> b;
            cout << query(root[k], 1, n, a, b) << '\n';
        }
        if (op == 3) {
            ll k; cin >> k;
            root[++sz] = root[k];
        }
    }
}
```

LiChaoTree.hpp
Description: Li-Chao Tree.

54dc16, 33 lines

```
const int N = 1<<18;

pll seg[N<<1];
ll a[N], qs[N], mqs[N];

ll f(pll p, ll x) {
    return p.st*x + p.nd;
}

void clr(int l, int r, int idx) {
    seg[idx] = {0, -1e18};
    if (l==r) return;
    int m = (l+r)/2;
    clr(l, m, 2*idx);
    clr(m+1, r, 2*idx+1);
}

void update(int l, int r, pll p, int idx) {
    int m = (l+r)/2;
    bool lef = f(p, l) > f(seg[idx], l);
    bool mid = f(p, m) > f(seg[idx], m);
    if (mid) swap(seg[idx], p);
    if (l==r) return;
    if (lef!=mid) update(l, m, p, 2*idx);
    else update(m+1, r, p, 2*idx+1);
}

ll query(int l, int r, int x, int idx) {
    if (l==r) return f(seg[idx], x);
    int m = (l+r)/2;
    if (x <= m) return max(query(l, m, x, 2*idx), f(seg[idx], x
        ));
    else return max(query(m+1, r, x, 2*idx+1), f(seg[idx], x));
}
```

BinaryTrie.hpp
Description: Binary Trie

c96a37, 68 lines

```
template<int BIT>
struct BinaryTrie {
  struct Node {
    array<int, 2> ch;
    int cnt;
    Node() : ch{-1, -1}, cnt(0) {}
  };
  vector<Node> t;
  BinaryTrie() : t{Node()} {}
  int new_node() {
    t.emplace_back(Node());
    return t.size()-1;
  }
  int size() { return t[0].cnt; }
  bool empty() { return size()==0; }
  void insert(ll val, int k=1) {
    int cur = 0;
    t[cur].cnt += k;
    for (int i = BIT-1; i >= 0; --i) {
      int b = val>>i & 1;
      if (t[cur].ch[b] == -1) t[cur].ch[b] = new_node();
      cur = t[cur].ch[b];
      t[cur].cnt += k;
    }
  }
  void erase(ll val, int k=1) {
    if (count(val) < k) return;
    int cur = 0;
    t[cur].cnt -= k;
    for (int i = BIT-1; i >= 0; --i) {
      int b = val>>i & 1;
      cur = t[cur].ch[b];
      t[cur].cnt -= k;
    }
  }
  void clear() {
    t = {Node};
  }
  ll get_max(ll val) {
    if (empty()) return LLONG_MIN;
    int cur = 0, ans = 0;
    for (int i = BIT-1; i >= 0; --i) {
      int b = val>>i & 1;
      if (t[cur].ch[!b] != -1 && t[t[cur].ch[!b]].cnt > 0) cur
          = t[cur].ch[!b], ans <<= 1, ans++;
      else cur = t[cur].ch[b], ans <<= 1;
    }
    return ans;
  }
  ll get_min(ll val) {
    if (empty()) return LLONG_MAX;
    int cur = 0, ans = 0;
```

```
    for (int i = BIT-1; i >= 0; --i) {
      int b = val>>i & 1;
      if (t[cur].ch[b] != -1 && t[t[cur].ch[b]].cnt > 0) cur =
           t[cur].ch[b], ans <<= 1, ans++;
      else cur = t[cur].ch[!b], ans <<= 1;
    }
    return ans;
  }
  int count(ll val) {
    int cur = 0;
    for (int i = BIT-1; i >= 0; --i) {
      int b = val>>i & 1;
      if (t[cur].ch[b] == -1) return false;
      cur = t[cur].ch[b];
    }
    return t[cur].cnt;
  }
};
```

Mo.hpp

Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change `step` to add/remove the edge (a, c) and remove the initial `add` call (but keep `in`).
Time: $\mathcal{O}\left(N\sqrt{Q}\right)$

a12ef4, 49 lines

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

vi mo(vector<pii> Q) {
  int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) {
    pii q = Q[qi];
    while (L > q.first) add(--L, 0);
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
  }
  return res;
}

vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0){
  int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
  add(0, 0), in[0] = 1;
  auto dfs = [&](int x, int p, int dep, auto& f) -> void {
    par[x] = p;
    L[x] = N;
    if (dep) I[x] = N++;
    for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
    if (!dep) I[x] = N++;
    R[x] = N;
  };
  dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) rep(end,0,2) {
    int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                  else { add(c, end); in[c] = 1; } a = c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
      I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
  }
  return res;
}
```

StaticTopTree.hpp

Description: Static Top Tree.

<bits/stdc++.h> 0c26f3, 89 lines

```
using namespace std;

using ll=long long;
const ll N=2e5+5,M=998244353;
vector<ll> g[N];
vector<ll> path[N];
ll par[N],a[N],sz[N],hv[N],top[N],bot[N];

struct node{
    ll m,c,k;
    node *par,*lc,*rc;
    bool s;
    node (): m(0), c(0), k(0), par(NULL), lc(NULL), rc(NULL), s
         (0) {}
    node (bool s): m(0), c(0), k(0), par(NULL), lc(NULL), rc(
         NULL), s(s) {}
    node (bool s,ll k): m(0), c(0), k(1), par(NULL), lc(NULL),
         rc(NULL), s(s) {}
    node (bool s,ll m,ll c): m(m), c(c), k(0), par(NULL), lc(
         NULL), rc(NULL), s(s) {}
};

node *pt[N],*com[N],*rake[N];

node* build_comp (int l,int r,int nt,int nb,node* np){
    if (l==r){
        int nn=path[nb][l];
        pt[nn]=new node(1,rake[nn]->k,a[nn]);
        rake[nn]->par=pt[nn];
        pt[nn]->par=np;
        pt[nn]->lc=rake[nn];
        return pt[nn];
    }
    node *re=new node(1);
    int mid=(l+r)/2;
    re->lc=build_comp(l,mid,nt,nb,re);
    re->rc=build_comp(mid+1,r,nt,nb,re);
    re->c=(re->rc->c+((re->lc->c*re->rc->m)%M))%M;
    re->m=(re->lc->m*re->rc->m)%M;
    re->par=np;
    return re;
}

node* build_rake(int l,int r,int nr,node* np){
    node *re=new node(0);
    if (l>r){
        re->k=1;
    }
    else if (l==r){
        int nn=g[nr][l];
        if (nn==hv[nr]) re->k=1;
        else{
            com[nn]=build_comp(0,path[bot[nn]].size()-1,nn,bot[
                nn],re);
            // cout << " comp " << nn << " : " << com[nn]->m <<
                ' ' << com[nn]->c << '\n';
            re->lc=com[nn];
            re->k=com[nn]->c;
        }
    }
    else{
        int mid=(l+r)/2;
        re->lc=build_rake(l,mid,nr,re);
        re->rc=build_rake(mid+1,r,nr,re);
        re->k=(re->lc->k*re->rc->k)%M;
    }
    re->par=np;
    return re;
}

void dfs (int nn){
    sz[nn]=1;
    int mx=0;
    for (auto e:g[nn]) {
        dfs(e);
        sz[nn]+=sz[e];
        if (sz[e]>mx){
            mx=sz[e];
            hv[nn]=e;
        }
    }
    if (!hv[nn]){
        path[nn].emplace_back(nn);
        top[nn]=nn;
        bot[nn]=nn;
    }
    else{
        path[bot[hv[nn]]].emplace_back(nn);
        top[bot[hv[nn]]]=nn;
        bot[nn]=bot[hv[nn]];
    }
    rake[nn]=build_rake(0,g[nn].size()-1,nn,NULL);
    // cout << " rake " << nn << " : " << rake[nn]->k << '\n';
}
```

Treap.hpp

Description: Treap

<bits/stdc++.h> 8cfd7e, 105 lines

```
using namespace std;

using ll=long long;
int n,q;
mt19937 rnd;

struct node{
    ll prio,val,sum,mn,mx,cnt=1;
    ll rev=0,lz1=0;
    bool f2=0;
    node *lc=NULL,*rc=NULL;
    node(ll val) : prio(rnd()),val(val),sum(val),mn(val),mx(val
        ) {}
    void up(node* nwv){
        if (!nwv) return;
        cnt+=nwv->cnt;
        sum+=nwv->sum;
        mx=max(mx,nwv->mx);
        mn=min(mn,nwv->mn);
        return;
    }
    void in(){
        cnt=1;
        sum=val;
        mn=val;
        mx=val;
    }
};

void push (node* nn,node* np){
```

```
    if (!nn) return;
    if (np->f2){
        nn->f2=1;
        nn->lz1=np->lz1;
    }
    else nn->lz1+=np->lz1;
    nn->rev^=np->rev;
    return;
}

void uplazy (node* nn){
    if (!nn) return;
    if (nn->rev==1){
        swap(nn->lc,nn->rc);
    }
    if (nn->f2){
        nn->val=0;
        nn->mn=0;
        nn->mx=0;
        nn->sum=0;
    }
    nn->val+=nn->lz1;
    nn->mn+=nn->lz1;
    nn->mx+=nn->lz1;
    nn->sum+=nn->cnt * nn->lz1;
    push(nn->lc,nn);
    push(nn->rc,nn);
    nn->lz1=0;
    nn->f2=0;
    nn->rev=0;
    return;
}

node* mg (node* t1,node* t2){
    uplazy(t1); uplazy(t2);
    if (!t1) return t2;
    if (!t2) return t1;
    if (t2->prio < t1->prio){
        t1->up(t2);
        t1->rc=mg(t1->rc,t2);
        return t1;
    }
    else{
        t2->up(t1);
        t2->lc=mg(t1,t2->lc);
        return t2;
    }
}

void sp (node* nn,node* &l,node* &r,int x,ll idx){
    if (!nn) return;
    uplazy(nn);
    if (idx<=x){
        l=nn;
        node* tg=nn->rc;
        uplazy(nn->lc);
        uplazy(nn->rc);
        nn->rc=NULL; nn->in(); nn->up(nn->lc);
        idx++;
        if (tg && tg->lc) idx+=tg->lc->cnt;
        sp(tg,l->rc,r,x,idx);
        l->up(l->rc);
    }
    else{
        r=nn;
        node* tg=nn->lc;
        uplazy(nn->lc);
        uplazy(nn->rc);
        nn->lc=NULL; nn->in(); nn->up(nn->rc);
        idx--;
        if(tg && tg->rc) idx-=tg->rc->cnt;
        sp(tg,l,r->lc,x,idx);
        r->up(r->lc);
    }
    return;
}
```

Number Theory (5)

ExtendedEuclid.hpp

Description: Extended Euclid algorithm for solving diophantine equation (ax + by = gcd(a, b)).
Time: $\mathcal{O}(\log \max\{a, b\})$

"../template/Header.hpp" 229e7c, 13 lines

```
pair<ll,ll> euclid(ll a,ll b){
    ll x=1,y=0,x1=0,y1=1;
    while(b!=0){
        ll q=a/b;
        x-=q*x1;
        y-=q*y1;
        a-=q*b;
        swap(x,x1);
        swap(y,y1);
        swap(a,b);
    }
    return {x,y};
}
```

euclid.h

Description: Finds two integers x and y, such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in `__gcd` instead. If a and b are coprime, then x is the inverse of $a \pmod b$. $x = x_0 + k * (b/g)$ $y = y_0 - k * (a/g)$

33ba8f, 5 lines

```
ll euclid(ll a, ll b, ll &x, ll &y) {
  if (!b) return x = 1, y = 0, a;
  ll d = euclid(b, a % b, y, x);
  return y -= a/b * x, d;
}
```

CRT.hpp

Description: Chinese Remainder Theorem.
`crt(a, m, b, n)` computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $|a| < m$ and $|b| < n$, x will obey $0 \leq x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$. If x0 and y0 is one of the solutions of ax + by = g, then the general solution is x = x0 + k * (b / g) and y = y0 - k * (a / g).
Time: $\log(n)$

"euclid.h" 04d93a, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

phiFunction.hpp

Description: *Euler's* ϕ function is defined as $\phi(n) :=$ # of positive integers $\leq n$ that are coprime with n. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1-1} ... (p_r - 1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$.

efae90, 10 lines

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
  for(int i=0; i<LIM; ++i) phi[i] = i & 1 ? i : i / 2;
  for (int i = 3; i < LIM; i += 2)
    if (phi[i] == i)
      for (int j = i; j < LIM; j += i)
        phi[j] -= phi[j] / i;
}
```

FloorSum.hpp

Description: Floor sum function. $f(a, b, c, n) = \sum_{x=0}^{n} \lfloor \frac{ax+b}{c} \rfloor$ becareful when a,b,c are negetive (use custom floor division and mod instead)
Time: $\mathcal{O}(\log a)$

d088d2, 7 lines

```
ll floor_sum(ll a,ll b,ll c,ll n){
    ll res=n*(n+1)/2*(a/c)+(n+1)*(b/c);
    a%=c,b%=c;
    if(a==0)return res;
    ll m=(a*n+b)/c;
    return res+n*m-floor_sum(c,c-b-1,a,m-1);
}
```

5.1 Prime Numbers

MillerRabin.hpp

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \mod c$.

be7e00, 25 lines

```
using ull = uint64_t;

ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}

bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n-1), d = n >> s;
  for (ull a : A) {   // ^ count trailing zeroes
    ull p = modpow(a%n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;
  }
  return 1;
}
```

LinearSieve.hpp

Description: Prime Number Generator in Linear Time
Time: $\mathcal{O}(N)$

"../template/Header.hpp" 194fb1, 15 lines

```
vi linear_sieve(int n) {
  vi prime, composite(n + 1);
  for(int i=2; i<=n; ++i) {
    if(!composite[i]) {
      prime.emplace_back(i);
    }
    for(int j=0; j<(int) prime.size() && i*prime[j]<=n; ++j) {
```

```
      composite[i * prime[j]] = true;
      if(i % prime[j] == 0) {
        break;
      }
    }
  }
  return prime;
}
```

FastEratosthenes.hpp

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s

"../template/Header.hpp" 295b58, 33 lines

```
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
  const int S = (int) round(sqrt(LIM)), R = LIM / 2;
  vi pr = {2}, sieve(S + 1);
  pr.reserve(int(LIM/log(LIM) * 1.1));
  vector<pii> cp;
  for(int i=3; i<=S; i+=2) {
    if(!sieve[i]) {
      cp.emplace_back(i, i * i / 2);
      for(int j=i*i; j<=S; j+=2*i) {
        sieve[j] = 1;
      }
    }
  }
  for(int L=1; L<=R; L+=S) {
    array<bool, S> block{};
    for(auto &[p, idx]: cp) {
      for(int i=idx; i<S+L; idx=(i+=p)) {
        block[i - L] = 1;
      }
    }
    for(int i=0; i<min(S, R-L); ++i) {
      if(!block[i]) {
        pr.emplace_back((L + i) * 2 + 1);
      }
    }
  }
  for(int i: pr) {
    isPrime[i] = 1;
  }
  return pr;
}
```

GolbatchConjecture.hpp

Description: Find two prime numbers which sum equals s
Time: $\mathcal{O}(N \log N)$

"FastEratosthenes.hpp" 88fb23, 18 lines

```
pair<int, int> goldbatchConjecture(int s, vi pr = {}){
  if (s <= 2 || s % 2 != 0) {
    return make_pair(-1, -1);
  }
  if (pr.size() == 0) {
    pr = eratosthenes();
  }
  for (auto x : pr) {
    if (x > s / 2) {
      break;
    }
    int d = s - x;
    if (binary_search(pr.begin(), pr.end(), d)) {
      return make_pair(min(x, d), max(x, d));
    }
  }
  return make_pair(-1, -1);
}
```

5.2 Modulo

ModArith.hpp

Description: Statistics on a mod'ed arithmetic sequence.
Time: $\mathcal{O}(\log m)$

"Euclid.h" 45f202, 32 lines

```
ll cdiv(ll x, ll y) { return x / y + ((x ^ y) > 0 && x % y); }

// min (ax + b) % m for 0 <= x <= n
ll minRemainder(ll a, ll b, ll m, ll n) {
  assert(a >= 0 && m > 0 && b >= 0 && n >= 0);
  a %= m, b %= m; n = min(n, m - 1);
  if (a == 0) return b;
  if (b >= a) {
    ll ad = cdiv(m - b, a);
    n -= ad; if (n < 0) return b;
    b += ad * a - m;
  }
  ll q = m / a, m2 = m % a;
  if (m2 == 0) return b;
  if (b / m2 > n / q) return b - n / q * m2;
  n -= b / m2 * q; b %= m2;
  ll y2 = (n * a + b) / m;
  ll x2 = cdiv(m2 * y2 - b, a);
  if (x2 * a - m2 * y2 + b >= m2) --x2;
  return minRemainder(a, b, m2, x2);
}

// min x >= 0 s.t. l <= (ax + b) % m <= r
ll minBetween(ll a, ll b, ll m, ll l, ll r) {
  ll x, y, g = euclid(a, m, x, y);
  if (g > 1)
    return minBetween(a/g,b/g,m/g,l/g+(l%g>b%g),r/g-(r%g<b%g));
  if (l > r) return -1; // no solution
  if ((x %= m) < 0) x += m;
  ll b2 = (l - b) * x % m;
  return minRemainder(x, b2 < 0 ? b2 + m : b2, m, r - l);
}
```

ModGen.hpp

Description: Finds a primitive root modulo p.

"Factor.h", "ModMulLL.h" ff3110, 8 lines

```
mt19937_64 rng(2137);
ll modGen(ll n) {
  map<ll, int> f; factor(n - 1, f); rep:
  ll g = rng() % (n - 1) + 1;
  for (auto [p, _] : f)
    if (modpow(g, (n - 1) / p, n) == 1) goto rep;
  return g;
}
```

ModInverse.hpp

Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime.

6f684f, 3 lines

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

ModLog.hpp

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.
Time: $\mathcal{O}(\sqrt{m})$

c040b8, 11 lines

```
ll modLog(ll a, ll b, ll m) {
  ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
  unordered_map<ll, ll> A;
  while (j <= n && (e = f = e * a % m) != b % m)
    A[e * b % m] = j++;
  if (e == b % m) return j;
  if (__gcd(m, e) == __gcd(m, b))
    rep(i,2,n+2) if (A.count(e = e * f % m))
      return n * i - A[e];
  return -1;
}
```

ModMulLL.hpp

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$).
Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

02ea06, 9 lines

```
ll modmul(ll a, ll b, ll M) {
  return (__int128)a * b % M;
}
ll modpow(ll b, ll e, ll mod) {
  ll ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
```

ModPow.hpp

b83e45, 8 lines

```
const ll mod = 1000000007; // faster if const

ll modpow(ll b, ll e) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}
```

ModSqrt.hpp

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModMulLL.h" b7cab4, 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  if (modpow(a, (p-1)/2, p) != 1) return -1; // no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

ModSum.hpp

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).

Time: $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModMulLL.h" b7cab4, 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  if (modpow(a, (p-1)/2, p) != 1) return -1; // no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

ModArithmetic.hpp

Description: Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

"euclid.h" 35bfea, 18 lines

```
const ll mod = 17; // change to something else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
};
```

Graph (6)

SCC.hpp

Description: Strongly Connected Component.

ed52c6, 20 lines

```
vi adj[MX], rev[MX];
int comp[MX], cnt;
bool vis[MX];
stack<int> order;

void dfs(int i) {
  if (vis[i]) return;
  vis[i] = 1;
  trav(j, adj[i]) dfs(j);
  order.push(i);
}
void dfs2(int i, int c) {
  if (comp[i]) return;
  comp[i] = c;
  for (auto j : rev[i]) dfs2(j, c);
}

rep(i, 1, M) dfs(i);
while (!order.empty()) dfs2(order.top(), ++cnt), order.pop();
```

ArticulationPoint.hpp

Description: Articulation Point

31c52f, 31 lines

```
// adj[u] = adjacent nodes of u
// ap = AP = articulation points
// p = parent
// disc[u] = discovery time of u
// low[u] = 'low' node of u

int dfsAP(int u, int p) {
  int children = 0;
  low[u] = disc[u] = ++Time;
  for (int& v : adj[u]) {
    if (v == p) continue; // we don't want to go back through
        the same path.
                          // if we go back is because we found
                              another way back
    if (!disc[v]) { // if V has not been discovered before
      children++;
      dfsAP(v, u); // recursive DFS call
      if (disc[u] <= low[v]) // condition #1
        ap[u] = 1;
      low[u] = min(low[u], low[v]); // low[v] might be an
          ancestor of u
    } else // if v was already discovered means that we found
        an ancestor
      low[u] = min(low[u], disc[v]); // finds the ancestor with
          the least discovery time
  }
  return children;
}

void AP() {
  ap = low = disc = vector<int>(adj.size());
  Time = 0;
  for (int u = 0; u < adj.size(); u++)
    if (!disc[u])
      ap[u] = dfsAP(u, u) > 1; // condition #2
}
```

Bridge.hpp

Description: Bridge

f7f699, 17 lines

```
vi adj[MX];
int disc[MX], low[MX], s[MX], t;
ll ans;
map<pii, pii> bridge;

void find_bridge(int u, int prt, int n) {
  disc[u] = low[u] = ++t, s[u] = 1;
  for (auto v : adj[u]) {
    if (v == prt) continue;
    if (!disc[v]) {
      find_bridge(v, u, n);
      if (disc[u] < low[v]) ckmax(ans, (ll)s[v]*(n-s[v]));
      s[u] += s[v];
    }
    ckmin(low[u], low[v]);
  }
}
```

Hierholzer.hpp

Description: Hierholzer Algorithm for finding Eulerian Circuit

92be7f, 9 lines

```
void dfs(int u) {
  while (!adj[u].empty()) {
    auto [v, idx] = adj[u].back(); adj[u].pop_back();
    if (walk[idx]) continue;
    walk[idx] = 1;
    dfs(v);
    ans[++c] = idx;
  }
}
```

6.1 Matching

HopcroftKarp.hpp

Description: Fast bipartite matching algorithm.

Time: $\mathcal{O}\left(E\sqrt{V}\right)$

"../template/Header.hpp" 0bd56f, 52 lines

```
struct HopcroftKarp{
    int n,m;
    vi l,r,lv,ptr;
    vector<vi> adj;
    HopcroftKarp(){}
    HopcroftKarp(int _n,int _m){init(_n,_m);}
    void init(int _n,int _m){
        n=_n,m=_m;
        adj.assign(n+m,vi{});
    }
    void addEdge(int u,int v){
        adj[u].emplace_back(v+n);
    }
    void bfs(){
        lv=vi(n,-1);
        queue<int> q;
        for(int i=0;i<n;i++)if(l[i]==-1){
            lv[i]=0;
            q.emplace(i);
        }
        while(!q.empty()){
            int u=q.front();
            q.pop();
            for(int v:adj[u])if(r[v]!=-1&&lv[r[v]]==-1){
                lv[r[v]]=lv[u]+1;
                q.emplace(r[v]);
            }
        }
    }
    bool dfs(int u){
        for(int &i=ptr[u];i<sz(adj[u]);i++){
            int v=adj[u][i];
            if(r[v]==-1||(lv[r[v]]==lv[u]+1&&dfs(r[v]))){
                l[u]=v,r[v]=u;
                return true;
            }
        }
        return false;
    }
    int maxMatching(){
        int match=0,cnt=0;
        l=r=vi(n+m,-1);
        do{
            ptr=vi(n);
            bfs();
            cnt=0;
            for(int i=0;i<n;i++)if(l[i]==-1&&dfs(i))cnt++;
            match+=cnt;
        }while(cnt);
        return match;
```

```
    }
};
```

Kuhn.hpp

Description: Kuhn Algorithm to find maximum bipartite matching or find augmenting path in bipartite graph.
Time: $\mathcal{O}(VE)$

"../template/Header.hpp" 4b91e8, 27 lines

```
vi adj[1010], match(1010, -1);
vector<bool> visited(1010, false);
bool try_match(int u) {
  if(visited[u]) {
    return false;
  }
  visited[u] = true;
  for(auto x: adj[u]) {
    if(match[x] == -1 || try_match(match[x])) {
      match[x] = u;
      return true;
    }
  }
  return false;
}

int max_matching() {
  for(int u=0; u<1010; ++u) {
    visited = vector<bool> (1010, false);
    try_match(u);
  }
  int cnt = 0;
  for(int u=0; u<1010; ++u) {
    cnt += (match[u] != -1);
  }
  return cnt;
}
```

WeightedMatching.hpp

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}(N^2M)$

2540b8, 34 lines

```
pair<ll, vector<int>> hungarian(const vector<vector<ll>> &a) {
  if (a.empty()) return {0, {}};
  int n = a.size() + 1, m = a[0].size() + 1;
  vector<ll> u(n), v(m);
    vector<int> p(m), ans(n - 1);
  for(int i=1;i<n;i++) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vector<ll> dist(m, LLONG_MAX);
        vector<int> pre(m, -1);
    vector<bool> done(m + 1);
    do { // dijkstra
      done[j0] = true;
      int i0 = p[j0], j1;
            ll delta = LLONG_MAX;
      for(int j=1;j<m;j++) if (!done[j]) {
        auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
      for(int j=0;j<m;j++) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  for(int j=1;j<m;j++) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

6.2 Network Flow

Dinic.hpp

Description: Dinic's Algorithm for finding the maximum flow.
Time: $\mathcal{O}(VE\log U)$ where U is the maximum flow.

2b9ab1, 88 lines

```
template<class T,bool directed=true,bool scaling=true>
struct Dinic{
    static constexpr T INF=numeric_limits<T>::max()/2;
    struct Edge{
        int to;
        T flow,cap;
        Edge(int _to,T _cap):to(_to),flow(0),cap(_cap){}
        T remain(){return cap-flow;}
    };
    int n,s,t;
    T U;
    vector<Edge> e;
    vector<vector<int>> g;
    vector<int> ptr,lv;
    bool calculated;
    T max_flow;
    Dinic(){}
    Dinic(int n,int s,int t){init(n,s,t);}
    void init(int _n,int _s,int _t){
        n=_n,s=_s,t=_t;
        U=0;
        e.clear();
        g.assign(n,{});
        calculated=false;
    }
    void add_edge(int from,int to,T cap){
        assert(0<=from&&from<n&&0<=to&&to<n);
        g[from].emplace_back(e.size());
        e.emplace_back(to,cap);
        g[to].emplace_back(e.size());
        e.emplace_back(from,directed?0:cap);
        U=max(U,cap);
    }
    bool bfs(T scale){
        lv.assign(n,-1);
        vector<int> q{s};
        lv[s]=0;
        for(int i=0;i<(int)q.size();i++){
            int u=q[i];
            for(int j:g[u]){
                int v=e[j].to;
                if(lv[v]==-1&&e[j].remain()>=scale){
                    q.emplace_back(v);
                    lv[v]=lv[u]+1;
                }
            }
        }
        return lv[t]!=-1;
    }
    T dfs(int u,int t,T f){
        if(u==t||f==0)return f;
        for(int &i=ptr[u];i<(int)g[u].size();i++){
            int j=g[u][i];
            int v=e[j].to;
            if(lv[v]==lv[u]+1){
                T res=dfs(v,t,min(f,e[j].remain()));
                if(res>0){
                    e[j].flow+=res;
                    e[j^1].flow-=res;
                    return res;
                }
            }
        }
        return 0;
    }
    T flow(){
        if(calculated)return max_flow;
        calculated=true;
        max_flow=0;
        for(T scale=scaling?1LL<<(63-__builtin_clzll(U)):1LL;
            scale>0;scale>>=1){
            while(bfs(scale)){
                ptr.assign(n,0);
                while(true){
                    T f=dfs(s,t,INF);
                    if(f==0)break;
                    max_flow+=f;
                }
            }
        }
        return max_flow;
    }
    pair<T,vector<int>> cut(){
        flow();
        vector<int> res(n);
        for(int i=0;i<n;i++)res[i]=(lv[i]==-1);
        return {max_flow,res};
    }
};
```

MinCostFlow.hpp

Description: minimum-cost flow algorithm.
Time: $\mathcal{O}(FE\log V)$ where F is max flow.

"../template/Header.hpp" 8ea1d2, 83 lines

```
template<class F,class C>
struct MinCostFlow{
    struct Edge{
        int to;
        F flow,cap;
        C cost;
        Edge(int _to,F _cap,C _cost):to(_to),flow(0),cap(_cap),
            cost(_cost){}
        F getcap(){
            return cap-flow;
        }
    };
    int n;
    vector<Edge> e;
    vector<vi> adj;
    vector<C> pot,dist;
    vi pre;
    bool neg;
    const F FINF=numeric_limits<F>::max()/2;
    const C CINF=numeric_limits<C>::max()/2;
    MinCostFlow(){}
    MinCostFlow(int _n){
        init(_n);
    }
    void init(int _n){
        n=_n;
        e.clear();
        adj.assign(n,{});
        neg=false;
```

```
    }
    void addEdge(int u,int v,F cap,C cost){
        adj[u].emplace_back(sz(e));
        e.emplace_back(v,cap,cost);
        adj[v].emplace_back(sz(e));
        e.emplace_back(u,0,-cost);
        if(cost<0)neg=true;
    }
    bool dijkstra(int s,int t){
        using P = pair<C,int>;
        dist.assign(n,CINF);
        pre.assign(n,-1);
        priority_queue<P,vector<P>,greater<P>> pq;
        dist[s]=0;
        pq.emplace(0,s);
        while(!pq.empty()){
            auto [d,u]=pq.top();
            pq.pop();
            if(dist[u]<d)continue;
            for(int i:adj[u]){
                int v=e[i].to;
                C ndist=d+pot[u]-pot[v]+e[i].cost;
                if(e[i].getcap()>0&&dist[v]>ndist){
                    pre[v]=i;
                    dist[v]=ndist;
                    pq.emplace(ndist,v);
                }
            }
        }
        return dist[t]<CINF;
    }
    pair<F,C> flow(int s,int t){
        F flow=0;
        C cost=0;
        pot.assign(n,0);
        if(neg)for(int t=0;t<n;t++)for(int i=0;i<sz(e);i++)if(e
            [i].getcap()>0){
            int u=e[i^1].to,v=e[i].to;
            pot[v]=min(pot[v],pot[u]+e[i].cost);
        } // Bellman-Ford
        while(dijkstra(s,t)){
            for(int i=0;i<n;i++)pot[i]+=dist[i];
            F aug=FINF;
            for(int u=t;u!=s;u=e[pre[u]^1].to){
                aug=min(aug,e[pre[u]].getcap());
            } // find bottleneck
            for(int u=t;u!=s;u=e[pre[u]^1].to){
                e[pre[u]].flow+=aug;
                e[pre[u]^1].flow-=aug;
            } // push flow
            flow+=aug;
            cost+=aug*pot[t];
        }
        return {flow,cost};
    }
};
```

Tree (7)

LCA.hpp

Description: LCA

523b28, 27 lines

```
vi adj[MX];
int up[MX][L+1], lvl[MX];

void dfs(int u, int prt) {
  lvl[u] = lvl[prt] + 1;
  up[u][0] = prt;
  rep(i, 1, L) up[u][i] = up[up[u][i-1]][i-1];
  for (auto v : adj[u]) {
    if (v == prt) continue;
    dfs(v, u);
  }
}

int ancestor(int u, int a) {
  rep(i, 0, L) if (a&(1<<i)) u = up[u][i];
  return u;
}

int lca(int a, int b) {
  if (lvl[a] < lvl[b]) swap(a, b);
  a = ancestor(a,lvl[a] - lvl[b]);
  if (a==b) return a;
  repd(i, 0, L)
    if (up[a][i] != up[b][i])
      a = up[a][i], b = up[b][i];
  return up[a][0];
}
```

CartesianTree.hpp

Description: Definite integral using Simpson's formula.

ad3b5d, 40 lines

```
template<class T,bool IS_MIN>
struct CartesianTree{
    int n;
    vector<T> &a;
    vector<pair<int,int>> range;
    vector<int> lch,rch,par;
    int root;

    CartesianTree(vector<T> &_a):n((int)_a.size()),a(_a){
        range.assign(n,{-1,-1});
        lch=rch=par=vector<int>(n,-1);
        if(n==1){
            range[0]={0,1};
            root=0;
            return;
        }
        auto cmp=[&](int i,int j)->bool {
            if(IS_MIN)a[i]<a[j]||(a[i]==a[j]&&i<j);
            return a[i]>a[j]||(a[i]==a[j]&&i<j);
        };
        vector<int> st;
        for(int i=0;i<n;i++){
            while(!st.empty()&&cmp(i,st.back())){
                lch[i]=st.back();
                st.pop_back();
            }
            range[i].first=(st.empty()?-1:st.back())+1;
        }
        for(int i=n-1;i>=0;i--){
            while(!st.empty()&&cmp(i,st.back())){
                rch[i]=st.back();
                st.pop_back();
            }
            range[i].second=(st.empty()?n:st.back())-1;
        }
        for(int i=0;i<n;i++)if(lch[i]!=-1)par[lch[i]]=i;
        for(int i=0;i<n;i++)if(rch[i]!=-1)par[rch[i]]=i;
        for(int i=0;i<n;i++)if(par[i]==-1)root=i;
    }
};
```

VirtualTree.hpp

Description: Virtual Tree of some problem.

<bits/stdc++.h> 575619, 114 lines

```
using namespace std;

const int N = 3e5 + 9;

vector<int> g[N];
int par[N][20], dep[N], sz[N], st[N], en[N], T;
void dfs(int u, int pre) {
  par[u][0] = pre;
  dep[u] = dep[pre] + 1;
  sz[u] = 1;
  st[u] = ++T;
  for (int i = 1; i <= 18; i++) par[u][i] = par[par[u][i - 1]][
      i - 1];
  for (auto v : g[u]) {
    if (v == pre) continue;
    dfs(v, u);
    sz[u] += sz[v];
  }
  en[u] = T;
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = 18; k >= 0; k--) if (dep[par[u][k]] >= dep[v]) u
      = par[u][k];
  if (u == v) return u;
  for (int k = 18; k >= 0; k--) if (par[u][k] != par[v][k]) u =
       par[u][k], v = par[v][k];
  return par[u][0];
}
int kth(int u, int k) {
  for (int i = 0; i <= 18; i++) if (k & (1 << i)) u = par[u][i
      ];
  return u;
}
int dist(int u, int v) {
  int lc = lca(u, v);
  return dep[u] + dep[v] - 2 * dep[lc];
}
int isanc(int u, int v) {
  return (st[u] <= st[v]) && (en[v] <= en[u]);
}
vector<int> t[N];
// given specific nodes, construct a compressed directed tree
    with these vertices(if needed some other nodes included)
// returns the nodes of the tree
// nodes.front() is the root
// t[] is the specific tree
vector<int> buildtree(vector<int> v) {
  // sort by entry time
  sort(v.begin(), v.end(), [](int x, int y) {
    return st[x] < st[y];
  });
  // finding all the ancestors, there are few of them
  int s = v.size();
  for (int i = 0; i < s - 1; i++) {
    int lc = lca(v[i], v[i + 1]);
    v.push_back(lc);
  }
  // removing duplicated nodes
  sort(v.begin(), v.end());
  v.erase(unique(v.begin(), v.end()), v.end());
  // again sort by entry time
  sort(v.begin(), v.end(), [](int x, int y) {
    return st[x] < st[y];
  });
  stack<int> st;
```

```
  st.push(v[0]);
  for (int i = 1; i < v.size(); i++) {
    while (!isanc(st.top(), v[i])) st.pop();
    t[st.top()].push_back(v[i]);
    st.push(v[i]);
  }
  return v;
}
int ans;
int imp[N];
int yo(int u) {
  vector<int> nw;
  for (auto v : t[u]) nw.push_back(yo(v));
  if (imp[u]) {
    for (auto x : nw) if (x) ans++;
    return 1;
  } else {
    int cnt = 0;
    for (auto x : nw) cnt += x > 0;
    if (cnt > 1) {
      ans++;
      return 0;
    }
    return cnt;
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int i, j, k, n, m, q, u, v;
  cin >> n;
  for (i = 1; i < n; i++) cin >> u >> v, g[u].push_back(v), g[v
      ].push_back(u);
  dfs(1, 0);
  cin >> q;
  while (q--) {
    cin >> k;
    vector<int> v;
    for (i = 0; i < k; i++) cin >> m, v.push_back(m), imp[m] =
        1;
    int fl = 1;
    for (auto x : v) if (imp[par[x][0]]) fl = 0;
    ans = 0;
    vector<int> nodes;
    if (fl) nodes = buildtree(v);
    if (fl) yo(nodes.front());
    if (!fl) ans = -1;
    cout << ans << '\n';
    // clear the tree
    for (auto x : nodes) t[x].clear();
    for (auto x : v) imp[x] = 0;
  }
  return 0;
}
// https://codeforces.com/contest/613/problem/D
```

HLD.hpp

Description: HLD

"../template/Header.hpp" cf6882, 45 lines

```
vector<vi> adj;
vector<int> sz, lvl, hv, hd, p, disc;
int t;

void dfs(int u, int parent) {
  sz[u] = 1;
  lvl[u] = lvl[parent] + 1;
  p[u] = parent;
  int c_hv=0, c_max=0;
  for(auto v: adj[u]) {
    if(v == parent) continue;
    dfs(v, u);
    sz[u] += sz[v];
    if(c_max < sz[v]) {
      c_hv = v;
      c_max = sz[v];
    }
  }
  hv[u] = c_hv;
}

void hld(int u, int parent) {
  if(hd[u] == 0) {
    hd[u] = u;
  }
  disc[u] = ++t;
  if(hv[u] != 0) {
    hd[hv[u]] = hd[u];
    hld(hv[u], u);
  }
  for(auto v: adj[u]) {
    if(v == parent || v == hv[u]) {
      continue;
    }
    hld(v, u);
  }
}

int lca(int u, int v) {
  while(hd[u] != hd[v]) {
    if(lvl[hd[u]] > lvl[hd[v]]) swap(u, v);
    v=p[hd[v]];
  }
  return lvl[u] < lvl[v] ? u: v;
}
```

CentroidDecom.hpp

Description: Centroid

"../template/Header.hpp" e46d44, 32 lines

```
vector<vi> adj;
vi sz;
vector<bool> used;

int find_size(int u, int p) {
  sz[u] = 1;
  for(auto v: adj[u]) {
    if(v == p || used[v]) continue;
    sz[u] += find_size(v, u);
  }
  return sz[u];
}

int find_cen(int u, int p, int t) {
  for(auto v: adj[u]) {
    if(v == p || used[v]) continue;
    if(sz[v] * 2 > t) find_cen(v, u, t);
  }
  return u;
}

void decom(int u) {
  u = find_cen(u, 0, find_size(u, 0));
  used[u] = true;
  for(auto v: adj[u]) {
    // dfs do something
  }
  for(auto v: adj[u]) {
    if(used[v]) continue;
    decom(v);
  }
}
```

RootedTreeIsomorphism.hpp

Description: Rooted Tree Isomorphism Check

7fe119, 29 lines

```
const int MX = 1e5 + 3;

vi adj[2][MX];
map<vector<int>, int> mp;
int cnt;

int dfs(int i, int u, int prt) {
  vector<int> m;
  for (auto v : adj[i][u]) {
    if (v == prt) continue;
    m.pb(dfs(i, v, u));
  }
  sort(all(m));
  if (!mp.count(m)) mp[m] = ++cnt;
  return mp[m];
}

void solve() {
  int n; cin >> n;
  rep(j, 2) rep(i, 2, n) {
    int u, v; cin >> u >> v;
    adj[j][u].pb(v); adj[j][v].pb(u);
  }
  cout << (dfs(0, 1, 0) == dfs(1, 1, 0) ? "YES" : "NO") << nl;
  //clear previous testcase
  rep(j, 2) rep(i, 1, n) adj[j][i].clear();
  mp.clear();
  cnt = 0;
}
```

UnrootedTreeIsomorphism.hpp

Description: Unrooted Tree Isomorphism Check

642f54, 45 lines

```
const int MX = 1e5 + 3;

vi adj[MX][2];
int sz[MX][2], cnt;
vector<int> centroid[2];
map<vector<int>, int> mp;

void find_centroid(int u, int t, int n, int prt=0) {
  bool is_centroid = 1;
  sz[u][t] = 1;
  for (auto v : adj[u][t]) {
    if (v == prt) continue;
    find_centroid(v, t, n, u);
    if (sz[v][t] > n/2) is_centroid = 0;
    sz[u][t] += sz[v][t];
  }
  if (n - sz[u][t] > n/2) is_centroid = 0;
  if (is_centroid) centroid[t].pb(u);
}

int dfs(int u, int t, int prt=0) {
  vector<int> m;
  for (auto v : adj[u][t]) {
    if (v == prt) continue;
    m.pb(dfs(v, t, u));
  }
  sort(all(m));
  if (!mp.count(m)) mp[m] = ++cnt;
  return mp[m];
}
```

```
void solve() {
  int n; cin >> n;
  rep(j, 2) rep(i, 2, n) {
    int u, v; cin >> u >> v;
    adj[u][j].pb(v), adj[v][j].pb(u);
  }
  rep(j, 2) find_centroid(1, j, n);
  bool iso = 0;
  for (auto c0 : centroid[0]) for (auto c1 : centroid[1]) {
    iso |= dfs(c0, 0) == dfs(c1, 1);
  }
  cout << (iso ? "YES" : "NO") << nl;
  rep(j, 2) rep(i, 1, n) adj[i][j].clear();
  rep(j, 2) centroid[j].clear();
```

Strings (8)

KMP.hpp

Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.
Time: $\mathcal{O}(N)$

d4375c, 16 lines

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}

vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
  return res;
}
```

ZAlgo.hpp

Description: z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

"../template/Header.hpp" 58dc79, 11 lines

```
vector<int> z_algorithm(const string &s){
    int n=(int)s.size();
    vector<int> z(n);
    z[0]=n;
    for(int i=1,l=0,r=1;i<n;i++){
        if(i<r)z[i]=min(r-i,z[i-l]);
        while(i+z[i]<n&&s[z[i]]==s[i+z[i]])z[i]++;
        if(i+z[i]>r)l=i,r=i+z[i];
    }
    return z;
}
```

AhoCorasick.hpp

Description: Aho Corasick Automaton

13041a, 75 lines

```
struct AC {
  struct Node {
    ll val = 0, mark = 0;
    vector<int> ch;
    Node() { ch.assign(27, 0); }
  };
  vector<Node> t;
  vector<int> out, link;
  AC () { push_back(); }
  int push_back() {
    t.push_back(Node());
    out.push_back(0);
    link.push_back(0);
    return t.size()-1;
  }
  int insert(const string s) {
    int idx = 0;
    for (auto c : s) {
      int v = (c==' '?26:c-'a');
      if (!t[idx].ch[v]) t[idx].ch[v] = push_back();
      idx = t[idx].ch[v];
    }
    t[idx].mark = 1;
    return idx;
  }
  void compute() {
    queue<int> q; q.push(0);
    while (!q.empty()) {
      int u = q.front(); q.pop();
      for (int c = 0; c < 27; ++c) {
        int v = t[u].ch[c];
        if (!v) t[u].ch[c] = t[link[u]].ch[c];
        else {
          link[v] = u ? t[link[u]].ch[c] : 0;
          out[v] = (t[link[v]].mark ? link[v] : out[link[v]]);
          q.push(v);
        }
      }
    }
  }
  int advance(int idx, char c) {
    int v = (c==' '?26:c-'a');
    while (idx && !t[idx].ch[v]) idx = link[idx];
    return t[idx].ch[v];
  }
};

string p[MX];
ll idx[MX], po[MX];

signed main() {
  ios_base::sync_with_stdio(0); cin.tie(NULL);
  int n, m, k; cin >> n >> m >> k;
  AC aho;
  string s; getline(cin, s);
  for (int i = 1; i <= n; ++i) getline(cin, p[i]), idx[i] = aho
      .insert(p[i]);
  aho.compute();
  po[0] = 1;
  for (int i = 1; i <= m; ++i) po[i] = (po[i-1] * 173) % MOD;
  for (int i = 1; i <= m; ++i) {
    int u, v; cin >> u >> v;
    aho.t[idx[u]].val = (aho.t[idx[u]].val + po[i]) % MOD;
    aho.t[idx[v]].val = (aho.t[idx[v]].val - po[i]) % MOD;
  }
  getline(cin, s);
  for (int i = 1; i <= k; ++i) {
    string s; getline(cin, s);
    ll val = 0, u = 0;
    for (auto c : s) {
      u = aho.advance(u, c);
      for (int v = u; v; v = aho.out[v]) val = (val + aho.t[v].
          val) % MOD;
    }
    cout << (val ? "no" : "yes") << nl;
  }
}
```

Ukkonen.hpp

Description: Ukkonen's Algorithm for suffix tree construction.

b43022, 59 lines

```
const int N=1000000,    // maximum possible number of nodes in
      suffix tree
    INF=1000000000; // infinity constant
string a;       // input string for which the suffix tree is
      being built
int t[N][26],   // array of transitions (state, letter)
    l[N],   // left...
    r[N],   // ...and right boundaries of the substring of a
          which correspond to incoming edge
    p[N],   // parent of the node
    s[N],   // suffix link
    tv,     // the node of the current suffix (if we're mid-
          edge, the lower node of the edge)
    tp,     // position in the string which corresponds to the
          position on the edge (between l[tv] and r[tv],
          inclusive)
    ts,     // the number of nodes
    la;     // the current character in the string

void ukkadd(int c) { // add character s to the tree
    suff:;      // we'll return here after each transition to
          the suffix (and will add character again)
    if (r[tv]<tp) { // check whether we're still within the
          boundaries of the current edge
        // if we're not, find the next edge. If it doesn't
            exist, create a leaf and add it to the tree
        if (t[tv][c]==-1) {t[tv][c]=ts;l[ts]=la;p[ts++]=tv;tv=s
            [tv];tp=r[tv]+1;goto suff;}
        tv=t[tv][c];tp=l[tv];
    } // otherwise just proceed to the next edge
    if (tp==-1 || c==a[tp]-'a')
        tp++; // if the letter on the edge equal c, go down
            that edge
    else {
        // otherwise split the edge in two with middle in node
            ts
        l[ts]=l[tv];r[ts]=tp-1;p[ts]=p[tv];t[ts][a[tp]-'a']=tv;
        // add leaf ts+1. It corresponds to transition through
            c.
        t[ts][c]=ts+1;l[ts+1]=la;p[ts+1]=ts;
        // update info for the current node - remember to mark
            ts as parent of tv
        l[tv]=tp;p[tv]=ts;t[p[ts]][a[l[ts]]-'a']=ts;ts+=2;
        // prepare for descent
        // tp will mark where are we in the current suffix
        tv=s[p[ts-2]];tp=l[ts-2];
        // while the current suffix is not over, descend
        while (tp<=r[ts-2]) {tv=t[tv][a[tp]-'a'];tp+=r[tv]-l[tv
            ]+1;}
        // if we're in a node, add a suffix link to it,
            otherwise add the link to ts
        // (we'll create ts on next iteration).
        if (tp==r[ts-2]+1) s[ts-2]=tv; else s[ts-2]=ts;
        // add tp to the new edge and return to add letter to
            suffix
        tp=r[tv]-(tp-r[ts-2])+2;goto suff;
    }
}

void build() {
    ts=2;
    tv=0;
    tp=0;
    fill(r,r+N,(int)a.size()-1);
    // initialize data for the root of the tree
    s[0]=1;
```

```
    l[0]=-1;
    r[0]=-1;
    l[1]=-1;
    r[1]=-1;
    memset (t, -1, sizeof t);
    fill(t[1],t[1]+26,0);
    // add the text to the tree, letter by letter
    for (la=0; la<(int)a.size(); ++la)
        ukkadd (a[la]-'a');
}
```

SuffixArray.hpp

Description: Suffix Array.

b62f8f, 31 lines

```
struct SuffixArray{
    int n;
    vector<int> sa,isa,lcp;
    SuffixArray(){}
    SuffixArray(const string &s){init(s);}
    void init(const string &s){
        n=(int)s.size();
        sa=isa=lcp=vector<int>(n+1);
        sa[0]=n;
        iota(sa.begin()+1,sa.end(),0);
        sort(sa.begin()+1,sa.end(),[&](int i,int j){return s[i
            ]<s[j];});
        for(int i=1;i<=n;i++){
            int x=sa[i-1],y=sa[i];
            isa[y]=i>1&&s[x]==s[y]?isa[x]:i;
        }
        for(int len=1;len<=n;len<<=1){
            vector<int> ps(sa),pi(isa),pos(n+1);
            iota(pos.begin(),pos.end(),0);
            for(auto i:ps)if((i-=len)>=0)sa[pos[isa[i]]++]=i;
            for(int i=1;i<=n;i++){
                int x=sa[i-1],y=sa[i];
                isa[y]=pi[x]==pi[y]&&pi[x+len]==pi[y+len]?isa[x
                    ]:i;
            }
        }
        for(int i=0,k=0;i<n;i++){
            for(int j=sa[isa[i]-1];j+k<n&&s[j+k]==s[i+k];k++);
            lcp[isa[i]]=k;
            if(k)k--;
        }
    }
};
```

PrefixFunction.hpp

Description: Prefix function. pi[i] := the length of the longest proper prefix of s[0:i] which is also a suffix of s[0:i].

3d65fe, 11 lines

```
template<class STR>
vector<int> prefix_function(const STR &s){
    int n=(int)s.size();
    vector<int> pi(n);
    for(int i=1,j=0;i<n;i++){
        while(j>0&&s[i]!=s[j])j=pi[j-1];
        if(s[i]==s[j])j++;
        pi[i]=j;
    }
    return pi;
}
```

SuffixAutomaton.hpp

Description: Suffix Automaton.
Find whether a string t is a substring of a string s by traversing the automaton.
Find whether a string t is a suffix of a string s by checking whether the last node is a terminal node.
Find the number of distinct substrings of a string s by calculating the number of distinc path using DP.
Count the number of occurrences of string t in string s. Let p be the node we end up at after traversing t in the automaton. The answer is the number of paths from p to terminal nodes.
Find first occurrence of string t in string s by calculating the longest path in the automaton after reaching node p.

a50940, 49 lines

```
template<class STR>
struct SuffixAutomaton{
    using T = typename STR::value_type;
    struct Node{
        map<T,int> nxt;
        int link,len;
        Node(int link,int len):link(link),len(len){}
    };
    vector<Node> nodes;
    int last;
    SuffixAutomaton():nodes{Node(-1,0)},last(0){}
    SuffixAutomaton(const STR &s):SuffixAutomaton(){
        for(auto c:s)extend(c);
    }
    int new_node(int link,int len){
        nodes.emplace_back(Node(link,len));
        return (int)nodes.size()-1;
    }
    void extend(T c){
        int cur=new_node(0,nodes[last].len+1);
        int p=last;
        while(p!=-1&&!nodes[p].nxt.count(c)){
            nodes[p].nxt[c]=cur;
            p=nodes[p].link;
        }
        if(p!=-1){
            int q=nodes[p].nxt[c];
            if(nodes[p].len+1==nodes[q].len){
                nodes[cur].link=q;
            }else{
                int r=new_node(nodes[q].link,nodes[p].len+1);
                nodes[r].nxt=nodes[q].nxt;
                while(p!=-1&&nodes[p].nxt[c]==q){
                    nodes[p].nxt[c]=r;
                    p=nodes[p].link;
                }
                nodes[q].link=nodes[cur].link=r;
            }
        }
        last=cur;
    }
    ll distinct_substrings(){
        ll res=0;
        for(int i=1;i<(int)nodes.size();i++){
            res+=nodes[i].len-nodes[nodes[i].link].len;
        }
        return res;
    }
};
```

Geometry (9)

9.1 Geometric primitives

Point.h

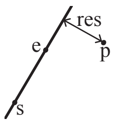
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

lineDistance.h

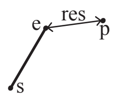
Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.



"Point.h" f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h



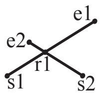
Description:
Returns the shortest distance between point p and the line segment from point s to e.
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 5c88f4, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description: If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



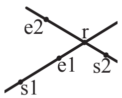
Usage: `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

lineIntersection.h

Description: If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.



Usage: `auto res = lineInter(s1,e1,s2,e2);`
`if (res.first == 1)`
`cout << "intersection point at " << res.second << endl;`

"Point.h" a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 ⇔ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage: `bool left = sideOf(p1,p2,q)==1;`

"Point.h" 3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

OnSegment.h

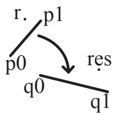
Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTransformation.h

Description:



Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

"Point.h" 03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

LineProjectionReflection.h

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h" b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
  P v = b - a;
  return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage: `vector<Angle> v = {w[0], w[0].t360() ...}; // sorted`
`int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }`
`// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i`

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

9.2 Circles

CircleIntersection.h

Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

CircleTangents.h

Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h" b0153d, 13 lines

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

CircleLine.h

Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

"Point.h" e0cfba, 9 lines

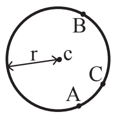
```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

CirclePolygonIntersection.h
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

circumcircle.h
Description:



The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.

"Point.h" 1caa3a, 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h
Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

"circumcircle.h" 09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

9.3 Polygons

InsidePolygon.h
Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.
Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
bool in = inPolygon(v, P{3, 3}, false);
Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

PolygonArea.h
Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" f12300, 6 lines

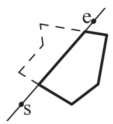
```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

PolygonCenter.h
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

PolygonCut.h
Description:

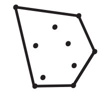


Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

ConvexHull.h
Description:



Returns a vector of the points of the convex hull in counterclockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Time: $\mathcal{O}(n \log n)$

"Point.h" 310954, 13 lines

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

HullDiameter.h
Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h" c571b8, 12 lines

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

PointInsideHull.h
Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

```
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon. `extrVertex` returns the point of a hull with the max projection onto a line.
Time: $\mathcal{O}(\log n)$

"Point.h" 7cf45b, 39 lines

```
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

9.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h" ac41a6, 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

ManhattanMST.h

Description: Given N points, returns up to 4*N edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights w(p, q) = |p.x - q.x| + |p.y - q.y|. Edges are in the form (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.
Time: $\mathcal{O}(N \log N)$

"Point.h" df6f59, 23 lines

```
typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
  vi id(sz(ps));
  iota(all(id), 0);
  vector<array<int, 3>> edges;
  rep(k,0,4) {
    sort(all(id), [&](int i, int j) {
         return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
    map<int, int> sweep;
    for (int i : id) {
      for (auto it = sweep.lower_bound(-ps[i].y);
                it != sweep.end(); sweep.erase(it++)) {
        int j = it->second;
        P d = ps[i] - ps[j];
        if (d.y > d.x) break;
        edges.push_back({d.y + d.x, i, j});
      }
      sweep[-ps[i].y] = i;
    }
    for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
  }
  return edges;
}
```

kdTree.h

Description: KD-tree (2d, can be extended to 3d)

"Point.h" bac5b0, 63 lines

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  // find nearest point to a point, and its squared distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

DelaunayTriangulation.h

Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are collinear or any four are on the same circle, behavior is undefined.
Time: $\mathcal{O}(n^2)$

"Point.h", "3dHull.h" c0e7bc, 10 lines

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3[t.a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
}
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], . . . }, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$

"Point.h" eefdf5, 88 lines

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
```

```
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

9.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

3058c3, 6 lines

```
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilist) {
  double v = 0;
  for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
  return v / 6;
}
```

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

8058ae, 32 lines

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.

Time: $\mathcal{O}\left(n^2\right)$

"Point3D.h" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis ($0 =$ north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```

ComplexGeometry.hpp
Description: geometry using std complex

<complex> 9a240d, 33 lines

```
using namespace std;

using point = complex<double>;

double dot(const point &a,const point &b){ return real(conj(a)*
     b); }
double cross(const point &a,const point &b){ return imag(conj(a
     )*b); }

point rotate_by(const point &p,const point about, double
     radians){
    return (p-about)*exp(point(0,radians))+about;
}

point project(const point &p,const point &about1,const point &
     about2){
    point z=p-about1;
    point w=about2-about1;
    return w*dot(z,w)/norm(w)+about1;
}

point reflex(const point &p,const point &about1,const point &
     about2){
    point z=p-about1;
    point w=about2-about1;
    return conj(z/w)*w+about1;
}

point intersect(const point &a,const point &b,const point &p,
     const point &q){
    double d1=cross(p-a,b-a);
    double d2=cross(q-a,b-a);
    return (d1*q-d2*p)/(d1-d2); // undefined if they are
         parallel
}

// find angle abc
point angle(const point &a,const point &b,const point &c){
    return abs(remainder(arg(a-b)-arg(c-b),2.0*M_PI));
}
```

DefiniteIntegral.hpp
Description: Definite integral using Simpson's formula.

edbaf5, 15 lines

```
template<class T,class F>
T difinite_integral(T a,T b,const F &f,int n){
    T res=0;
    T dx=(b-a)/n;
    T fl=0,fr=f(a);
    for(int i=0;i<n;i++){
        T l=a+dx*i;
        T r=l+dx;
        fl=fr;
        fr=f(r);
        T fm=f((l+r)/2);
        res+=fl+4*fm+fr;
    }
    return res*dx/6;
}
```

Dynamic Programming (10)

CHT.hpp
Description: Convex Hull Container

5012e6, 23 lines

```
struct CHT {
  struct Line {
    ll m, c;
    Line (ll _m, ll _c) { m = _m, c = _c; }
    ll eval(ll x) { return m*x + c; }
  };
  deque<Line> dq;

  bool check(Line base, Line cur, Line add) {
    return (cur.c - base.c)*(base.m - add.m) < (add.c - base.c)
         *(base.m - cur.m);
  }

  void insert(ll m, ll c) {
    Line l(m, c);
    while (dq.size() > 1 && !check(dq.end()[-2], dq.back(), l))
         dq.pop_back();
    dq.push_back(l);
  }

  ll query(ll x) {
    while (dq.size() > 1 && dq[0].eval(x) < dq[1].eval(x)) dq.
         pop_front();
    return dq[0].eval(x);
  }
};
```

Knuth.hpp
Description: Knuth

04ace3, 30 lines

```
int solve() {
    int N;
    ... // read N and input
    int dp[N][N], opt[N][N];

    auto C = [&](int i, int j) {
        ... // Implement cost function C.
    };

    for (int i = 0; i < N; i++) {
        opt[i][i] = i;
        ... // Initialize dp[i][i] according to the problem
    }

    for (int i = N-2; i >= 0; i--) {
        for (int j = i+1; j < N; j++) {
            int mn = INT_MAX;
            int cost = C(i, j);
            for (int k = opt[i][j-1]; k <= min(j-1, opt[i+1][j
                 ]); k++) {
                if (mn >= dp[i][k] + dp[k+1][j] + cost) {
                    opt[i][j] = k;
                    mn = dp[i][k] + dp[k+1][j] + cost;
                }
            }
            dp[i][j] = mn;
        }
    }

    return dp[0][N-1];
}
```

DVC.hpp
Description: Optimize $O(N^2K)$ to $O(NK\log N)$

"../template/Header.hpp" aa5ddf, 19 lines

```
vector<vl> cst, dp;
ll cost(int l, int r) {
  return cst[l][r];
}

void divide(int l, int r, int opt_l, int opt_r, int c) {
  if(l > r) return ;
  int mid = (l + r) / 2;
  pair<ll, int> best = make_pair(INF, -1);
  for(int k=opt_l; k<=min(mid, opt_r); ++k) {
    best = min(best, make_pair(dp[c - 1][k] + cost(k + 1, mid),
          k));
  }
  dp[c][mid] = best.first;
  divide(l, mid - 1, opt_l, best.second, c);
  divide(mid + 1, r, best.second, opt_r, c);
}

// for(int c=1; c<=K; ++c) divide(1, N, 1, N, c);
```

SlopeTrick.hpp
Description: Absolute Smth

"../template/Header.hpp" f62f9a, 36 lines

```
ll extending_value;

struct slope_trick {
  multiset<ll> ms_l, ms_r;
  ll min_y = 0ll, lz_l = 0ll, lz_r = 0ll;
  bool extending = false;
  void add_line(ll v) {
    if(extending) {
      lz_l -= extending_value;
      lz_r -= extending_value;
    }
    extending = true;
    if(ms_l.empty() && ms_r.empty()) {
      ms_l.emplace(v);
      ms_r.emplace(v);
    }
    else if(v <= *ms_l.rbegin() + lz_l) {
      min_y += (*ms_l.rbegin() + lz_l) - v;
      ms_r.emplace(*ms_l.rbegin() + lz_l - lz_r);
      ms_l.erase(--ms_l.end());
      ms_l.emplace(v - lz_l);
      ms_l.emplace(v - lz_l);
    }
    else if(v >= *ms_r.begin() + lz_r) {
      min_y += v - (*ms_r.begin() + lz_r);
      ms_l.emplace(*ms_r.begin() + lz_r - lz_l);
      ms_r.erase(ms_r.begin());
      ms_r.emplace(v - lz_r);
      ms_r.emplace(v - lz_r);
    }
    else {
      ms_l.emplace(v - lz_l);
      ms_r.emplace(v - lz_r);
    }
  }
};
```

AlienTrick.hpp
Description: Alien Trick

<bits/stdc++.h> 8435cb, 53 lines

```
using namespace std;

using ll=long long;
using tl=__int128_t;
using db=double;
```

```cpp
const ll N=500005;
ll c[N],sum[N];
deque<tuple<ll,ll,int>> dq;

ll minPenguinValue(int n, int m, vector<int> a){
    ll l=0,r=2e18,re;
    for (int i=1;i<=n;i++){
        sum[i]=sum[i-1]+ll(a[i-1]);
        c[i]=c[i-1]+sum[i]*ll(a[i-1]);
    }
    while (l<=r){
        ll mid=l+(r-l)/2;
        while (!dq.empty()) dq.pop_back();
        dq.emplace_back(0,0,0);
        for (int i=1;i<=n;i++){
            while (dq.size()>1){
                auto [m1,c1,u1]=dq.front();
                dq.pop_front();
                auto [m2,c2,u2]=dq.front();
                if (c2-c1>=sum[i]*(m1-m2)){
                    dq.emplace_front(m1,c1,u1);
                    break;
                }
            }
            auto [nm,nc,nu]=dq.front();
            ll nv=nm*sum[i]+nc+c[i]+mid;
            ll m3=-sum[i],c3=-c[i]+sum[i]*sum[i]+nv;
            if (i==n){
                if (nu+1>m) l=mid+1;
                else{
                    re=nv-m*mid;
                    r=mid-1;
                }
            }
            while (dq.size()>1){
                auto [m2,c2,u2]=dq.back();
                dq.pop_back();
                auto [m1,c1,u1]=dq.back();
                if (db(c3-c1)/db(m1-m3)<=db(c3-c2)/db(m2-m3)){
                    dq.emplace_back(m2,c2,u2);
                    break;
                }
            }
            dq.emplace_back(m3,c3,nu+1);
        }
    }
    return re;
}
```

Polynomials (11)

11.1 Newton's Method

if $F(Q)=0$, then $Q_{2n}\equiv Q_n-\frac{F(Q_n)}{F'(Q_n)}\ (\mathrm{mod} x^{2n})$

$$Q=P^{-1}:Q_{2n}\equiv Q_n\cdot(2-P\cdot Q_n^2)\ (\mathrm{mod}\ x^{2n})$$

$$Q=\ln P=\int\frac{P'}{P}\mathrm{d}x$$

$$Q=e^p:Q_{2n}\equiv Q_n(1+P-\ln Q_n)\ (\mathrm{mod}\ x^{2n})$$

$$Q=\sqrt{P}:Q_{2n}\equiv\frac{1}{2}(Q_n+P\cdot Q_n^{-1})\ (\mathrm{mod}\ x^{2n})$$

$$Q=P^k=\alpha^k x^{kt}e^{k\ln T};P=\alpha\cdot x^t\cdot T,T(0)=1$$

Interpolation.hpp

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x)=a[0]*x^0+\ldots+a[n-1]*x^{n-1}$. For numerical precision, pick $x[k]=c*\cos(k/(n-1)*\pi),k=0\ldots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

08bf48, 13 lines

```cpp
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

FormalPowerSeries.hpp

Description: basic operations of formal power series

3d4f4b, 136 lines

```cpp
template<class mint>
struct FormalPowerSeries:vector<mint>{
    using vector<mint>::vector;
    using FPS = FormalPowerSeries;

    FPS &operator+=(const FPS &rhs){
        if(rhs.size()>this->size())this->resize(rhs.size());
        for(int i=0;i<rhs.size();i++)(*this)[i]+=rhs[i];
        return *this;
    }
    FPS &operator+=(const mint &rhs){
        if(this->empty())this->resize(1);
        (*this)[0]+=rhs;
        return *this;
    }
    FPS &operator-=(const FPS &rhs){
        if(rhs.size()>this->size())this->resize(rhs.size());
        for(int i=0;i<rhs.size();i++)(*this)[i]-=rhs[i];
        return *this;
    }
    FPS &operator-=(const mint &rhs){
        if(this->empty())this->resize(1);
        (*this)[0]-=rhs;
        return *this;
    }
    // FPS &operator*=(const FPS &rhs){
    //     auto res=NTT<mint>()(*this,rhs);
    //     return *this=FPS(res.begin(),res.end());
    // }
    FPS &operator*=(const mint &rhs){
        for(auto &a:*this)a*=rhs;
        return *this;
    }
    friend FPS operator+(FPS lhs,const FPS &rhs){return lhs+=
        rhs;}
    friend FPS operator+(FPS lhs,const mint &rhs){return lhs+=
        rhs;}
    friend FPS operator+(const mint &lhs,FPS &rhs){return rhs+=
        lhs;}
    friend FPS operator-(FPS lhs,const FPS &rhs){return lhs-=
        rhs;}
    friend FPS operator-(FPS lhs,const mint &rhs){return lhs-=
        rhs;}
    friend FPS operator-(const mint &lhs,FPS rhs){return -(rhs-
        lhs);}
    friend FPS operator*(FPS lhs,const FPS &rhs){return lhs*=
        rhs;}
    friend FPS operator*(FPS lhs,const mint &rhs){return lhs*=
        rhs;}
    friend FPS operator*(const mint &lhs,FPS rhs){return rhs*=
        lhs;}

    FPS operator-(){return (*this)*-1;}

    FPS rev(){
        FPS res(*this);
        reverse(res.beign(),res.end());
        return res;
    }
    FPS pre(int sz){
        FPS res(this->begin(),this->begin()+min((int)this->size
            (),sz));
        if(res.size()<sz)res.resize(sz);
        return res;
    }
    FPS shrink(){
        FPS res(*this);
        while(!res.empty()&&res.back()==mint{})res.pop_back();
        return res;
    }
    FPS operator>>(int sz){
        if(this->size()<=sz)return {};
        FPS res(*this);
        res.erase(res.begin(),res.begin()+sz);
        return res;
    }
    FPS operator<<(int sz){
        FPS res(*this);
        res.insert(res.begin(),sz,mint{});
        return res;
    }
    FPS diff(){
        const int n=this->size();
        FPS res(max(0,n-1));
        for(int i=1;i<n;i++)res[i-1]=(*this)[i]*mint(i);
        return res;
    }
    FPS integral(){
        const int n=this->size();
        FPS res(n+1);
        res[0]=0;
        if(n>0)res[1]=1;
        ll mod=mint::get_mod();
        for(int i=2;i<=n;i++)res[i]=(-res[mod%i])*(mod/i);
        for(int i=0;i<n;i++)res[i+1]*=(*this)[i];
        return res;
    }
    mint eval(const mint &x){
        mint res=0,w=1;
        for(auto &a:*this)res+=a*w,w*=x;
        return res;
    }

    FPS inv(int deg=-1){
        assert(!this->empty()&&(*this)[0]!=mint(0));
        if(deg==-1)deg=this->size();
        FPS res{mint(1)/(*this)[0]};
        for(int i=2;i>>1<deg;i<<=1){
            res=(res*(mint(2)-res*pre(i))).pre(i);
        }
        return res.pre(deg);
    }
    FPS log(int deg=-1){
        assert(!this->empty()&&(*this)[0]==mint(1));
```

```
        if(deg==-1)deg=this->size();
        return (pre(deg).diff()*inv(deg)).pre(deg-1).integral()
            ;
    }
    FPS exp(int deg=-1){
        assert(this->empty()||(*this)[0]==mint(0));
        if(deg==-1)deg=this->size();
        FPS res{mint(1)};
        for(int i=2;i>>1<deg;i<<=1){
            res=(res*(pre(i)-res.log(i)+mint(1))).pre(i);
        }
        return res.pre(deg);
    }
    FPS pow(ll k,int deg=-1){
        const int n=this->size();
        if(deg==-1)deg=n;
        if(k==0){
            FPS res(deg);
            if(deg)res[0]=mint(1);
            return res;
        }
        for(int i=0;i<n;i++){
            if(__int128_t(i)*k>=deg)return FPS(deg,mint(0));
            if((*this)[i]==mint(0))continue;
            mint rev=mint(1)/(*this)[i];
            FPS res=(((*this*rev)>>i).log(deg)*k).exp(deg);
            res=((res*binpow((*this)[i],k))<<(i*k)).pre(deg);
            return res;
        }
        return FPS(deg,mint(0));
    }
};
using FPS=FormalPowerSeries<mint>;
```

FFT.hpp

Description: Fast Fourier transform
Time: $\mathcal{O}(N \log N)$

"../template/Header.hpp" 30fff5, 83 lines

```
using ll = long long;
using poly = vector<ll>;
const ll M=998244353;
ll pr=3,inv_pr,m;

ll bp(ll a,ll b){
    ll re=1ll;
    while (b){
        if (b&1) re=(re*a)%M;
        a=(a*a)%M;
        b>>=1;
    }
    return re;
}

void fft (poly &a, bool inv){
    ll n=a.size();
    if (n==1) return;
    poly o(n>>1ll),e(n>>1ll);
    for (ll i=0;i<n;i+=2){
        e[i>>1ll]=a[i];
        o[i>>1ll]=a[i+1];
    }
    fft(e,inv);
    fft(o,inv);
    ll w = bp(pr,((M-1ll)/n));
    if (inv) w = bp(inv_pr,((M-1ll)/n));
    for (ll i=0;i<(n>>1ll);i++){
        a[i]=(e[i]+((bp(w,i)*o[i])%M))%M;
        a[i+(n>>1ll)]=((e[i]-((bp(w,i)*o[i])%M))%M+M)%M;
    }
}

poly mul (poly a, poly b){
    poly re;
    ll n=1;
    while (n<a.size()+b.size()) n<<=1ll;
    a.resize(n);
    b.resize(n);
    fft(a,0);
    fft(b,0);
    re.resize(n);
    for (ll i=0;i<n;i++){
        re[i]=(a[i]*b[i])%M;
    }
    fft(re,1);
    for (auto &e:re) e=(e*bp(n,M-2))%M;
    return re;
}

poly inv(poly &a){
    poly re;
    re.emplace_back(bp(a[0],M-2));
    poly b;
    for (int n=1;n<a.size();n<<=1){
        while (b.size()<min(int(a.size()),n*2)) b.emplace_back(
            a[b.size()]);
        poly tmp = mul(re,re);
        tmp = mul(tmp,b);
        for (int i=0;i<n;i++){
            re[i]=(((re[i]*2)%M-tmp[i])%M+M)%M;
        }
        for (int i=0;i<n;i++){
            re.emplace_back((M-tmp[i+n])%M);
        }
    }
    return re;
}

poly sqrtp(poly &a){
    poly re;
    re.emplace_back(1); // need re[0]^2 == a[0], but now a[0]
        == 1
    poly b;
    for (int n=1;n<a.size();n<<=1){
        while (b.size()<min(int(a.size()),n*2)) b.emplace_back(
            a[b.size()]);
        for (int i=0;i<n;i++) re.emplace_back(0);
        poly tmp = mul(b,inv(re));
        for (int i=0;i<n*2;i++){
            re[i]=(re[i]+tmp[i])%M;
            re[i]=(re[i]*bp(2,M-2))%M;
        }
    }
    return re;
}
```

NTT.hpp

Description: Number theoretic transform
Time: $\mathcal{O}(N \log N)$

"../template/Header.hpp", "../modular-arithmetic/BinPow.hpp",
"../modular-arithmetic/MontgomeryModInt.hpp" 2b2392, 39 lines

```
template<class mint=mint>
struct NTT{
  using vm = vector<mint>;

  static constexpr mint root=mint::get_root();
    static_assert(root!=0);

  static void ntt(vm &a){
    int n=a.size(),L=31-__builtin_clz(n);
    vm rt(n);
    rt[1]=1;
    for(int k=2,s=2;k<n;k*=2,s++){
      mint z[]={1,binpow(root,MOD>>s)};
      for(int i=k;i<2*k;i++)rt[i]=rt[i/2]*z[i&1];
    }
    vi rev(n);
    for(int i=1;i<n;i++)rev[i]=(rev[i/2]|(i&1)<<L)/2;
    for(int i=1;i<n;i++)if(i<rev[i])swap(a[i],a[rev[i]]);
    for(int k=1;k<n;k*=2)for(int i=0;i<n;i+=2*k)for(int j=0;j<k
        ;j++){
      mint z=rt[j+k]*a[i+j+k];
      a[i+j+k]=a[i+j]-z;
      a[i+j]+=z;
    }
  }
  static vm conv(const vm &a,const vm &b){
    if(a.empty()||b.empty())return {};
    int s=a.size()+b.size()-1,n=1<<(32-__builtin_clz(s));
    mint inv=mint(n).inv();
    vm in1(a),in2(b),out(n);
    in1.resize(n),in2.resize(n);
    ntt(in1),ntt(in2);
    for(int i=0;i<n;i++)out[-i&(n-1)]=in1[i]*in2[i]*inv;
    ntt(out);
    return vm(out.begin(),out.begin()+s);
  }
  vm operator()(const vm &a,const vm &b){
    return conv(a,b);
  }
};
```

Convolutions (12)

AndConvolution.hpp

Description: Bitwise AND Convolution. Superset Zeta Transform: $A'[S] = \sum_{T \supseteq S} A[T]$. Superset Mobius Transform: $A[T] = \sum_{S \supseteq T} (-1)^{|S-T|} A'[S]$.
Time: $\mathcal{O}(N \log N)$.

"../template/Header.hpp" 7916f8, 34 lines

```
template<class T>
void superset_zeta(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=1;i<n;i<<=1){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j^i]+=a[j];
            }
        }
    }
}

template<class T>
void superset_mobius(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=n;i>>=1;){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j^i]-=a[j];
            }
        }
    }
}

template<class T>
```

```
vector<T> and_convolution(vector<T> a,vector<T> b){
    superset_zeta(a);
    superset_zeta(b);
    for(int i=0;i<(int)a.size();i++)a[i]*=b[i];
    superset_mobius(a);
    return a;
}
```

GCDConvolution.hpp

Description: GCD Convolution. Multiple Zeta Transform: $A'[n] = \sum_{n|m} A[m]$. Multiple Mobius Transform: $A[n] = \sum_{n|m} \mu(m/n)A'[m]$.
Time: $\mathcal{O}(N \log\log N)$.

"../template/Header.hpp" 7f6c2d, 34 lines

```
template<class T>
void multiple_zeta(vector<T> &a){
    int n=(int)a.size();
    vector<bool> is_prime(n,true);
    for(int p=2;p<n;p++){
        if(!is_prime[p])continue;
        for(int i=(n-1)/p;i>=1;i--){
            is_prime[i*p]=false;
            a[i]+=a[i*p];
        }
    }
}

template<class T>
void multiple_mobius(vector<T> &a){
    int n=(int)a.size();
    vector<bool> is_prime(n,true);
    for(int p=2;p<n;p++){
        if(!is_prime[p])continue;
        for(int i=1;i*p<n;i++){
            is_prime[i*p]=false;
            a[i]-=a[i*p];
        }
    }
}

template<class T>
vector<T> gcd_convolution(vector<T> a,vector<T> b){
    multiple_zeta(a);
    multiple_zeta(b);
    for(int i=0;i<(int)a.size();i++)a[i]*=b[i];
    multiple_mobius(a);
    return a;
}
```

LCMConvolution.hpp

Description: LCM Convolution. Divisor Zeta Transform: $A'[n] = \sum_{d|n} A[d]$. Divisor Mobius Transform: $A[n] = \sum_{d|n} \mu(n/d)A'[d]$.
Time: $\mathcal{O}(N \log\log N)$.

"../template/Header.hpp" 41fe9d, 34 lines

```
template<class T>
void divisor_zeta(vector<T> &a){
    int n=(int)a.size();
    vector<bool> is_prime(n,true);
    for(int p=2;p<n;p++){
        if(!is_prime[p])continue;
        for(int i=1;i*p<n;i++){
            is_prime[i*p]=false;
            a[i*p]+=a[i];
        }
    }
}

template<class T>
void divisor_mobius(vector<T> &a){
    int n=(int)a.size();
    vector<bool> is_prime(n,true);
    for(int p=2;p<n;p++){
        if(!is_prime[p])continue;
        for(int i=(n-1)/p;i>=1;i--){
            is_prime[i*p]=false;
            a[i*p]-=a[i];
        }
    }
}

template<class T>
vector<T> lcm_convolution(vector<T> a,vector<T> b){
    divisor_zeta(a);
    divisor_zeta(b);
    for(int i=0;i<(int)a.size();i++)a[i]*=b[i];
    divisor_mobius(a);
    return a;
}
```

ORConvolution.hpp

Description: Bitwise OR Convolution. Subset Zeta Transform: $A'[S] = \sum_{T\subseteq S} A[T]$. Subset Mobius Transform: $A[T] = \sum_{S\subseteq T} (-1)^{|T-S|}A'[S]$.
Time: $\mathcal{O}(N \log N)$.

"../template/Header.hpp" c58b77, 34 lines

```
template<class T>
void subset_zeta(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=1;i<n;i<<=1){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j]+=a[j^i];
            }
        }
    }
}

template<class T>
void subset_mobius(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=n;i>>=1;){
        for(int j=0;j<n;j++){
            if(j&i){
                a[j]-=a[j^i];
            }
        }
    }
}

template<class T>
vector<T> or_convolution(vector<T> a,vector<T> b){
    subset_zeta(a);
    subset_zeta(b);
    for(int i=0;i<(int)a.size();i++)a[i]*=b[i];
    subset_mobius(a);
    return a;
}
```

XORConvolution.hpp

Description: Bitwise XOR Convolution. Fast Walsh-Hadamard Transform: $A'[S] = \sum_T (-1)^{|S\&T|}A[T]$.
Time: $\mathcal{O}(N \log N)$.

"../template/Header.hpp" 05848d, 29 lines

```
template<class T>
void fwht(vector<T> &a){
    int n=(int)a.size();
    assert(n==(n&-n));
    for(int i=1;i<n;i<<=1){
        for(int j=0;j<n;j++){
            if(j&i){
                T &u=a[j^i],&v=a[j];
                tie(u,v)=make_pair(u+v,u-v);
            }
        }
    }
}

template<class T>
vector<T> xor_convolution(vector<T> a,vector<T> b){
    int n=(int)a.size();
    fwht(a);
    fwht(b);
    for(int i=0;i<n;i++)a[i]*=b[i];
    fwht(a);
    T div=T(1)/T(n);
    if(div==T(0)){
        for(auto &x:a)x/=n;
    }else{
        for(auto &x:a)x*=div;
    }
    return a;
}
```

MaxPlusConvolution.hpp

Description: Max Plus Convolution. Find $C[k] = max_{i+j=k}\{A[i] + B[j]\}$.
Time: $\mathcal{O}(N)$.

7176a2, 94 lines

```
// SMAWCK algorithm for finding row-wise maxima.
// f(i,j,k) checks if M[i][j] <= M[i][k].
// f(i,j,k) checks if M[i][k] is at least as good as M[i][j].
// higher is better.
template<class F>
vector<int> smawck(const F &f,const vector<int> &rows,const
     vector<int> &cols){
    int n=(int)rows.size(),m=(int)cols.size();
    if(max(n,m)<=2){
        vector<int> ans(n,-1);
        for(int i=0;i<n;i++){
            for(int j:cols){
                if(ans[i]==-1||f(rows[i],ans[i],j)){
                    ans[i]=j;
                }
            }
        }
        return ans;
    }
    if(n<m){
        // reduce
        vector<int> st;
        for(int j:cols){
            while(true){
                if(st.empty()){
                    st.emplace_back(j);
                    break;
                }else if(f(rows[(int)st.size()-1],st.back(),j))
                    {
                    st.pop_back();
                }else if(st.size()<n){
                    st.emplace_back(j);
                    break;
                }else{
                    break;
                }
            }
```

```
        }
        return smawck(f,rows,st);
    }
    vector<int> ans(n,-1);
    vector<int> new_rows;
    for(int i=1;i<n;i+=2){
        new_rows.emplace_back(rows[i]);
    }
    auto res=smawck(f,new_rows,cols);
    for(int i=0;i<new_rows.size();i++){
        ans[2*i+1]=res[i];
    }
    for(int i=0,l=0,r=0;i<n;i+=2){
        if(i+1==n)r=m;
        while(r<m&&cols[r]<=ans[i+1])r++;
        ans[i]=cols[l++];
        for(;l<r;l++){
            if(f(rows[i],ans[i],cols[l])){
                ans[i]=cols[l];
            }
        }
        l--;
    }
    return ans;
}

template<class F>
vector<int> smawck(const F &f,int n,int m){
    vector<int> rows(n),cols(m);
    iota(rows.begin(),rows.end(),0);
    iota(cols.begin(),cols.end(),0);
    return smawck(f,rows,cols);
}

// Max Plus Convolution.
// b must be convex, i.e. b[i]-b[i-1]>=b[i+1]-b[i].
template<class T>
vector<T> max_plus_convolution_arbitary_convex(vector<T> a,
    const vector<T> &b){
    if(a.empty()||b.empty())return {};
    if((int)b.size()==1){
        for(auto &x:a)x+=b[0];
        return a;
    }
    int n=(int)a.size(),m=(int)b.size();
    auto f=[&](int i,int j){
        return a[j]+b[i-j];
    };
    auto cmp=[&](int i,int j,int k){
        if(i<k)return false;
        if(i-j>=m)return true;
        return f(i,j)<=f(i,k);
    };
    auto best=smawck(cmp,n+m-1,n);
    vector<T> ans(n+m-1);
    for(int i=0;i<n+m-1;i++){
        ans[i]=f(i,best[i]);
    }
    return ans;
}
```

Various (13)

GaussianElimination.hpp

Description: Gaussian Elimination

d847fe, 45 lines

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity
    or a big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

XORBasis.hpp

Description: XOR Basis

4f2ef8, 33 lines

```
template<int BIT>
struct XOR_basis {
  vector<ll> basis;
  int mSize = 0;
  XOR_basis() { basis.assign(BIT, 0); }
  //Insert v into basis, if v is already in the span of the
      basis, do nothing. O(BIT)
  void insert(ll v) {
    for (int i = BIT-1; i >= 0; --i) {
      if (!(v>>i & 1)) continue;
      if (basis[i]) v ^= basis[i];
      else {basis[i] = v, mSize++; return; }
    }
  }
  int size() { return mSize; }
  //Perform row reduction to make basis in reduced row echelon
      form in O(BIT^2)
  void reduce() {
    for (int i = BIT-1; i >= 0; --i) {
      for (int j = i-1; j >= 0; --j) {
        if (!basis[j]) continue;
        if (basis[i]>>j & 1) basis[i] ^= basis[j];
      }
    }
  }
  //Check whether v is in span of the current basis in O(BIT)
  bool in_span(ll v) {
    for (int i = BIT-1; i >= 0; --i) {
      if (!(v>>i & 1)) continue;
      v ^= basis[i];
    }
    return (v == 0);
  }
  ll &operator[] (int i) { return basis[i]; }
};
```

RangeXor.hpp

Description: find all range of x such that l <= x xor p < r.

"../template/Header.hpp" cc7fb9, 18 lines

```
template<class F>
void range_xor(ll p,ll l,ll r,const F &query){
    for(int i=0;i<60;i++){
        if(l==r)break;
        ll b=1LL<<i;
        if(l&b){
            query(l^p,(l^p)+b);
            l+=b;
        }
        if(r&b){
            r-=b;
            query(r^p,(r^p)+b);
        }
        if(p&b){
            p^=b;
        }
    }
}
```

13.1 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

13.1.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `rep(b,0,K) rep(i,0,(1 << K))`
  `if (i & 1 << b) D[i] += D[i^(1 << b)];`
  computes all sums of subsets.

13.1.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.

- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

Competitive Programming Topics (14)

topics.txt

159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```

troubleshooting.txt

52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
```

```
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```