Algorithm Competition Template Library

newESCOM version 2022-04-01

"Latin America regional finals" Edition

Contents

Contest (1)

template.cpp 13 lines

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define endl '\n'
#define __ ios_base::sync_with_stdio(false),cin.tie(NULL);

signed main() { __
  int n;
  cin>>n;
  vector<int> nums(n);
  for(auto &c:nums)cin>>c;
  return 0;
}
```

.bashrc 2 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
  -fsanitize=undefined,address'
```

hash.sh 3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

troubleshoot.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?
```

Mathematics (2)

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \dots + c_k a_{n-k}$, and $r_1, \dots, r_k$ are distinct roots of $x^k + c_1 x^{k-1} + \dots + c_k$, there are $d_1, \dots, d_k$ s.t.

$$a_n = d_1 r_1^n + \dots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v + w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v+w)=\frac{\tan v+\tan w}{1-\tan v\tan w}$$
$$\sin v+\sin w=2\sin\frac{v+w}{2}\cos\frac{v-w}{2}$$
$$\cos v+\cos w=2\cos\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$(V+W)\tan(v-w)/2=(V-W)\tan(v+w)/2$$

where V,W are lengths of sides opposite angles v,w.

$$a\cos x+b\sin x=r\cos(x-\phi)$$
$$a\sin x+b\cos x=r\sin(x+\phi)$$

where $r=\sqrt{a^2+b^2},\phi=\operatorname{atan2}(b,a)$.

2.4 Geometry

2.4.1 Triangles

Side lengths: a,b,c

Semiperimeter: $p=\dfrac{a+b+c}{2}$

Area: $A=\sqrt{p(p-a)(p-b)(p-c)}$

Circumradius: $R=\dfrac{abc}{4A}$

Inradius: $r=\dfrac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a=\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$

Length of bisector (divides angles in two):

$$s_a=\sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$$

Law of sines: $\dfrac{\sin\alpha}{a}=\dfrac{\sin\beta}{b}=\dfrac{\sin\gamma}{c}=\dfrac{1}{2R}$

Law of cosines: $a^2=b^2+c^2-2bc\cos\alpha$

Law of tangents: $\dfrac{a+b}{a-b}=\dfrac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

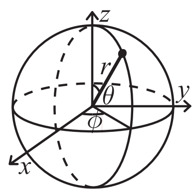
2.4.2 Quadrilaterals

With side lengths a,b,c,d, diagonals e,f, diagonals angle θ, area A and magic flux $F=b^2+d^2-a^2-c^2$:

$$4A=2ef\cdot\sin\theta=F\tan\theta=\sqrt{4e^2f^2-F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef=ac+bd$, and $A=\sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.4.3 Spherical coordinates



$$\begin{aligned} x&=r\sin\theta\cos\phi & r&=\sqrt{x^2+y^2+z^2}\\ y&=r\sin\theta\sin\phi & \theta&=\operatorname{acos}(z/\sqrt{x^2+y^2+z^2})\\ z&=r\cos\theta & \phi&=\operatorname{atan2}(y,x)\end{aligned}$$

2.5 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x=\frac{1}{\sqrt{1-x^2}}\qquad \frac{d}{dx}\arccos x=-\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x=1+\tan^2 x\qquad \frac{d}{dx}\arctan x=\frac{1}{1+x^2}$$
$$\int\tan ax=-\frac{\ln|\cos ax|}{a}\qquad \int x\sin ax=\frac{\sin ax-ax\cos ax}{a^2}$$
$$\int e^{-x^2}=\frac{\sqrt{\pi}}{2}\operatorname{erf}(x)\qquad \int xe^{ax}dx=\frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx=[F(x)g(x)]_a^b-\int_a^b F(x)g'(x)dx$$

2.6 Sums

$$c^a+c^{a+1}+\cdots+c^b=\frac{c^{b+1}-c^a}{c-1},c\neq1$$

$$1+2+3+\cdots+n=\frac{n(n+1)}{2}$$
$$1^2+2^2+3^2+\cdots+n^2=\frac{n(2n+1)(n+1)}{6}$$
$$1^3+2^3+3^3+\cdots+n^3=\frac{n^2(n+1)^2}{4}$$
$$1^4+2^4+3^4+\cdots+n^4=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.7 Series

$$e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots,(-\infty<x<\infty)$$
$$\ln(1+x)=x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots,(-1<x\leq1)$$
$$\sqrt{1+x}=1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\ldots,(-1\leq x\leq1)$$
$$\sin x=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots,(-\infty<x<\infty)$$
$$\cos x=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots,(-\infty<x<\infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu=\mathbb{E}(X)=\sum_x xp_X(x)$ and variance $\sigma^2=V(X)=\mathbb{E}(X^2)-(\mathbb{E}(X))^2=\sum_x(x-\mathbb{E}(X))^2p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX+bY)=a\mathbb{E}(X)+b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX+bY)=a^2V(X)+b^2V(Y).$$

2.8.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\mathrm{Bin}(n,p)$, $n=1,2,\ldots$, $0\leq p\leq1$.

$$p(k)=\binom{n}{k}p^k(1-p)^{n-k}$$

$$\mu=np,\ \sigma^2=np(1-p)$$

$\mathrm{Bin}(n,p)$ is approximately $\mathrm{Po}(np)$ for small p.

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability p is $\mathrm{Fs}(p)$, $0\leq p\leq1$.

$$p(k)=p(1-p)^{k-1},\ k=1,2,\ldots$$

$$\mu=\frac{1}{p},\ \sigma^2=\frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\mathrm{Po}(\lambda)$, $\lambda=t\kappa$.

$$p(k)=e^{-\lambda}\frac{\lambda^k}{k!},k=0,1,2,\ldots$$

$$\mu=\lambda,\ \sigma^2=\lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\mathrm{U}(a,b)$, $a<b$.

$$f(x)=\begin{cases}\frac{1}{b-a} & a<x<b\\ 0 & \text{otherwise}\end{cases}$$

$$\mu=\frac{a+b}{2},\ \sigma^2=\frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\mathrm{Exp}(\lambda)$, $\lambda>0$.

$$f(x)=\begin{cases}\lambda e^{-\lambda x} & x\ge 0\\ 0 & x<0\end{cases}$$

$$\mu=\frac{1}{\lambda},\ \sigma^2=\frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu,\sigma^2)$, $\sigma>0$.

$$f(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1\sim\mathcal{N}(\mu_1,\sigma_1^2)$ and $X_2\sim\mathcal{N}(\mu_2,\sigma_2^2)$ then

$$aX_1+bX_2+c\sim\mathcal{N}(\mu_1+\mu_2+c,a^2\sigma_1^2+b^2\sigma_2^2)$$

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1,X_2,\ldots$ be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P}=(p_{ij})$, with $p_{ij}=\Pr(X_n=i|X_{n-1}=j)$, and $\mathbf{p}^{(n)}=\mathbf{P}^n\mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)}=\Pr(X_n=i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.

π is a stationary distribution if $\pi=\pi\mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i=\frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. π_j/π_i is the expected number of visits in state j between two visits in state i.

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty}\mathbf{P}^k=\mathbf{1}\pi$.

A Markov chain is an A-chain if the states can be partitioned into two sets $\mathbf{A}$ and $\mathbf{G}$, such that all states in $\mathbf{A}$ are absorbing ($p_{ii}=1$), and all states in $\mathbf{G}$ leads to an absorbing state in $\mathbf{A}$. The probability for absorption in state $i\in\mathbf{A}$, when the initial state is j, is $a_{ij}=p_{ij}+\sum_{k\in\mathbf{G}}a_{ik}p_{kj}$. The expected time until absorption, when the initial state is i, is $t_i=1+\sum_{k\in\mathbf{G}}p_{ki}t_k$.

Data structures (3)

OrderStatisticTree.cpp

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change `null_type`.
Time: $\mathcal{O}(\log N)$

6ac8ab, 19 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
  tree_order_statistics_node_update>;
void example() {
 Tree<int> t, t2; t.insert(8);
 auto it = t.insert(10).first;
 assert(it == t.lower_bound(9));
 assert(t.order_of_key(10) == 1);
 assert(t.order_of_key(11) == 2);
 assert(*t.find_by_order(0) == 8);
 t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
// Multiset
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less_equal<int>,rb_tree_tag,tree_order_statistics_node_update> ordered_set;
```

HashMap.cpp

Description: Hash map with mostly the same API as unordered_map, but $\sim$ 3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

3f859a, 10 lines

```
#include <bits/stdc++.h>
using namespace std;
typedef ll long long
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
 const uint64_t C = ll(4e18 * acos(0)) | 71;
 ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h( { } , { } , { } , { } , {1<<16 } );
```

SegmentTree.cpp

Description: A Segment Tree is a data structure that allows answering range queries over an array effectively, This includes finding an asossiative function of consecutive array elements
Usage: `for(int i = 0;i<n;i++)update(i,i,vector[i]);`
`STmin ST(N); fill like the recursive one`

Time: build: $\mathcal{O}(n\log N)$ query: $\mathcal{O}(\log N)$.

33759f, 56 lines

```
/*+ ---- Recursive segment tree with lazy propagation ---- */
vector<int> st;
vector<int> lazy;
void propagate(int v,int l ,int r){
  if(!lazy[v])return ;
 // For asigments replace += to =
  st[v] += ((r-l)+1)*lazy[v];
  if(l!= r){
    lazy[v<<1] += lazy[v];
    lazy[v<<1|1]+= lazy[v];
  }
  lazy[v] = 0;
}
int n; /*+ n is global for use default values and send less parameters */
void update(int l,int r,int val,int v = 1,int sl = 0,int sr = n-1){
  propagate(v,sl,sr);
  if(r<sl || l>sr || sl>sr)return ;
  if(sl>= l && sr<=r){
    lazy[v] += val;
    propagate(v,sl,sr);
    return;
  }
  int m = (sl+sr)>>1;
  update(l,r,val,v<<1,sl,m);
  update(l,r,val,v<<1|1,m+1,sr);
  st[v] = st[v<<1]+st[v<<1|1];
}
int query(int l,int r,int v = 1,int sl = 0,int sr = n-1){
  propagate(v,sl,sr);
  if(r<sl || l>sr || sl>sr)return 0;
  if(sl>= l && sr<=r)return st[v];
  int m = (sl+sr)>>1;
  return query(l,r,v<<1,sl,m)+query(l,r,v<<1|1,m+1,sr);
}
/*+ ---- Iterative segment tree much faster, setted to return min in a range ---- */
struct STmin{
  int n;
  vector<int> st;
  STmin(int n):n(n){
    st.resize(2*n,inf);
  }
 inline void update(int x, int val) {
```

```
  x += n;
  st[x] = val;
  for (; x >>= 1 ; st[x] = min(st[x<<1], st[x<<1|1]));
 }
 inline int query(int l, int r) {
  int ans = inf;
    if(r<l)return 0;
  for (l += n, r += n; l <= r; l = (l + 1) / 2, r = (r - 1) / 2) {
   if (l & 1) ans = min(ans, st[l]);
   if (~r & 1) ans = min(ans, st[r]);
  }
  return ans;
 }
};
```

SegmentTreeDynamic.cpp

Description: A Segment Tree that stores data only if is needed or asked , that alows to manage bigger "arrays" more than 10^7

Usage: `Node st(0,maximum_size);`

Time: $\mathcal{O}(\log N)$ per query.

372f41, 49 lines

```
struct Node{
  int sum,greater,l,r,lazy;
  bool prop;
  vector<Node> sons;
  Node(int _l,int _r):l(_l),r(_r),lazy(0),greater(0),sum(0),prop(
      false){}
  void propagate(){
    if(sons.empty() && l!=r){
      int m = (l+r)>>1;
      sons.push_back(Node(l,m));
      sons.push_back(Node(m+1,r));
    }
    if(prop){
      sum = greater = lazy*((r-l)+1);
      if(l!=r){
        sons[0].prop = true;
        sons[1].prop = true;
        sons[1].lazy = lazy;
        sons[0].lazy = lazy;
      }
      prop = false;
    }
  }
  // Update in a range [a,b]
  void update(int a,int b ,int v){
    propagate();
    if(a>r || b<l)return ;
    if(l>=a && r<=b){
      lazy = v;
      prop = true;
      propagate();
        return;
    }
    int m = (l+r)>>1;
    sons[0].update(a,b,v);
    sons[1].update(a,b,v);
    sum = sons[0].sum+sons[1].sum;
    greater=max(sons[0].greater,sons[0].sum+sons[1].greater);
  }
  int query(int k){
    propagate();
    if(l == r){ return greater>k?l-1:l; }
    sons[0].propagate();
    // sons[1].propagate();
    if(sons[0].greater>k)
      return sons[0].query(k);
    else
      return sons[1].query(k-sons[0].sum);
  }
};
```

SegmentTreePersistent.cpp

Description: A persistent data structure is a data structure that remembers it previous state for each modification. This allows to access any version of this data structure that interest us and execute a query on it.

Time: $\mathcal{O}(\log N)$ per query.

5ad594, 52 lines

```
const int maxn = 800007;
int L[maxn],R[maxn],st[maxn],lazy[maxn],N;
int n; /** Must be global for default values in functions */
int newLeaf(int val){
  int p = ++N;
  L[p] = R[p] = 0;
  st[p] = val;
  return p;
}
int newParent(int l,int r){
  int p = ++N;
  L[p] = l;
  R[p] = r;
  st[p] = st[l]+st[r];
  return p;
}
int newLazy(int v,int val,int l,int r){
  int p = ++N;
  L[p] = L[v];
  R[p] = R[v];
  lazy[p] += val;
  st[p] = st[v]+((r-l)+1)*val;
  return p;
}
int build(vector<int> &A,int l = 0,int r = n-1){
  if(l== r)return newLeaf(A[l]);
  int mid = (l+r)>>1;
  return newParent(build(A,l,mid),build(A,mid+1,r));
}
void propagate(int p,int l,int r){
  if(lazy[p]==0)return;
  if(l!= r){
    int mid = (l+r)>>1;
    L[p] = newLazy(L[p],lazy[p],l,mid);
    R[p] = newLazy(R[p],lazy[p],mid+1,r);
  }
  lazy[p] = 0;
}
int update(int l,int r,int val,int p,int sl = 0 ,int sr = n-1){
  if(sr<l || sl>r)return p;
  if(sl>=l && sr<=r)return newLazy(p,val,sl,sr);
  propagate(p,sl,sr);
  int mid = (sl+sr)>>1;
  return newParent(update(l,r,val,L[p],sl,mid),update(l,r,val,R[p],
      mid+1,sr));
}
int query(int l,int r,int p,int sl = 0,int sr = n-1){
  if(sr<l || sl> r)return 0;
  if(sl>=l && sr<=r)return st[p];
  int mid = (sl+sr)>>1;
  propagate(p,sl,sr);
  return query(l,r,L[p],sl,mid)+query(l,r,R[p],mid+1,r);
}
```

SegmentTreeDynamic2D.cpp

Description: A 2D segment tree with updates in a point

Usage: `ST t(0, n - 1);`
`t.upd(x, y, v); // update in a point(x,y) a value v (asigment)`
`t.query(x1, x2, y1, y2); // Returns an assosiative function in a submatrix`

Time: $\mathcal{O}(\log N)$ per query.

ab213f, 143 lines

```
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
const int N = 3e5 + 9;
struct node {
 node *l, *r;
 int pos, key, mn, mx;
 long long val, g;
 node(int position, long long value) {
  l = r = nullptr;
  mn = mx = pos = position;
  key = rnd();
  val = g = value;
 }
 void pull() {
  g = val;
  if(l) g = __gcd(g, l->g);
  if(r) g = __gcd(g, r->g);
  mn = (l ? l->mn : pos);
  mx = (r ? r->mx : pos);
```

```cpp
  }
};
//memory O(n)
struct treap {
  node *root;
  treap() {
    root = nullptr;
  }
  void split(node *t, int pos, node *&l, node *&r) {
    if (t == nullptr) {
      l = r = nullptr;
      return;
    }
    if (t->pos < pos) {
      split(t->r, pos, l, r);
      t->r = l;
      l = t;
    } else {
      split(t->l, pos, l, r);
      t->l = r;
      r = t;
    }
    t->pull();
  }
  node* merge(node *l, node *r) {
    if (!l || !r) return l ? l : r;
    if (l->key < r->key) {
      l->r = merge(l->r, r);
      l->pull();
      return l;
    }
    r->l = merge(l, r->l);
    r->pull();
    return r;
  }
  bool find(int pos) {
    node *t = root;
    while (t) {
      if (t->pos == pos) return true;
      if (t->pos > pos) t = t->l;
      else t = t->r;
    }
    return false;
  }
  void upd(node *t, int pos, long long val) {
    if (t->pos == pos) {
      t->val = val;
      t->pull();
      return;
    }
    if (t->pos > pos) upd(t->l, pos, val);
    else upd(t->r, pos, val);
    t->pull();
  }
  void insert(int pos, long long val) { //set a_pos = val
    if (find(pos)) upd(root, pos, val);
    else {
      node *l, *r;
      split(root, pos, l, r);
      root = merge(merge(l, new node(pos, val)), r);
    }
  }
  long long query(node *t, int st, int en) {
    if (t->mx < st || en < t->mn) return 0;
    if (st <= t->mn && t->mx <= en) return t->g;
    long long ans = (st <= t->pos && t->pos <= en ? t->val : 0);
    if (t->l) ans = __gcd(ans, query(t->l, st, en));
    if (t->r) ans = __gcd(ans, query(t->r, st, en));
    return ans;
  }
  long long query(int l, int r) { //gcd of a_i such that l <= i <= r
    if (!root) return 0;
    return query(root, l, r);
  }
  void print(node *t) {
    if (!t) return;
    print(t->l);
    cout << t->val << " ";
    print(t->r);
  }
};
//total memory along with treap = nlogn
struct ST {
  ST *l, *r;
  treap t;
  int b, e;
  ST() {
    l = r = nullptr;
  }
  ST(int st, int en) {
    l = r = nullptr;
    b = st, e = en;
  }
  void fix(int pos) {
    long long val = 0;
    if (l) val = __gcd(val, l->t.query(pos, pos));
    if (r) val = __gcd(val, r->t.query(pos, pos));
    t.insert(pos, val);
  }
  void upd(int x, int y, long long val) { //set a[x][y] = val
    if (e < x || x < b) return;
    if (b == e) {
      t.insert(y, val);
      return;
    }
    if (b != e) {
      if (x <= (b + e) / 2) {
        if (!l) l = new ST(b, (b + e) / 2);
        l->upd(x, y, val);
      } else {
        if (!r) r = new ST((b + e) / 2 + 1, e);
        r->upd(x, y, val);
      }
    }
    fix(y);
  }
  long long query(int i, int j, int st, int en) { //gcd of a[x][y]
      ↪ such that i <= x <= j && st <= y <= en
    if (e < i || j < b) return 0;
    if (i <= b && e <= j) return t.query(st, en);
    long long ans = 0;
    if (l) ans = __gcd(ans, l->query(i, j, st, en));
    if (r) ans = __gcd(ans, r->query(i, j, st, en));
    return ans;
  }
};
```

SegmentTreeBeats.cpp

Description: A segment tree with special queries, allows to update For all i in [l,r), change Ai to max/min(Ai, x) Query for the sum of Ai in [l, r]

Usage: `build(); //check that array is global`
`update_max/min(--l,r,x); // update is in a range [l,r) and 0 indexed`

Time: $\mathcal{O}(\log N)$ per query.

5a9847, 126 lines

```cpp
const int INF = 1e15;
struct info {
  int mx1, mx2, mx_cnt, mn1, mn2, mn_cnt;
  info(int a = -INF, int b = -INF, int c = 0, int d = INF, int e =
      ↪ INF, int f = 0) {
    mx1 = a, mx2 = b, mx_cnt = c, mn1 = d, mn2 = e, mn_cnt = f;
  }
  friend inline info merge(info u, info v) {
    if (u.mx1 < v.mx1) {
      u.mx_cnt = 0;
      u.mx2 = u.mx1;
      u.mx1 = v.mx1;
    }
    if (u.mx1 == v.mx1) {
      u.mx_cnt += v.mx_cnt;
      u.mx2 = max(u.mx2, v.mx2);
    }
    else
      u.mx2 = max(u.mx2, v.mx1);
    if (u.mn1 > v.mn1) {
      u.mn_cnt = 0;
```

```
      u.mn2 = u.mn1;
      u.mn1 = v.mn1;
    }
    if (u.mn1 == v.mn1) {
      u.mn_cnt += v.mn_cnt;
      u.mn2 = min(u.mn2, v.mn2);
    }
    else
      u.mn2 = min(u.mn2, v.mn1);
    return u;
  }
};
struct node {
  info p;
  long long sum;
  int mx_lazy, mn_lazy;
  node(info a = info(), long long b = 0) {
    p = a, sum = b, reset();
  }
  inline void reset() {
    mx_lazy = -INF, mn_lazy = INF;
  }
  inline void set_max(int x) {
    if (x <= p.mn1)
      return;
    sum += 1LL * (x - p.mn1) * p.mn_cnt;
    if (p.mx1 == p.mn1)
      p.mx1 = x;
    if (p.mx2 == p.mn1)
      p.mx2 = x;
    p.mn1 = mx_lazy = x;
  }
  inline void set_min(int x) {
    if (x >= p.mx1)
      return;
    sum += 1LL * (x - p.mx1) * p.mx_cnt;
    if (p.mn1 == p.mx1)
      p.mn1 = x;
    if (p.mn2 == p.mx1)
      p.mn2 = x;
    p.mx1 = mn_lazy = x;
  }
  friend inline node merge(node u, node v) {
    u.p = merge(u.p, v.p);
    u.sum += v.sum;
    u.reset();
    return u;
  }
};
const int N = 2e5;
node seg[N << 2];
int n, q, a[N]; /** Global variables are important for default values in functions */
inline void find(int id) {
  seg[id] = merge(seg[id << 1], seg[id << 1 | 1]);
}
inline void shift(int id) {
  for (auto p: {id << 1, id << 1 | 1}) {
    seg[p].set_max(seg[id].mx_lazy);
    seg[p].set_min(seg[id].mn_lazy);
  }
  seg[id].reset();
}
void build(int id = 1, int st = 0, int en = n) {
  if (en - st == 1) {
    info x;
    x.mx1 = a[st], x.mx_cnt = 1;
    x.mn1 = a[st], x.mn_cnt = 1;
    seg[id] = {x, a[st]};
    return;
  }
  int mid = st + en >> 1;
  build(id << 1, st, mid);
  build(id << 1 | 1, mid, en);
  find(id);
}
void update_max(int l, int r, int x, int id = 1, int st = 0, int en = n) {
  if (r <= st || en <= l || seg[id].p.mn1 >= x)
    return;
  if (l <= st && en <= r && seg[id].p.mn2 > x)
    return seg[id].set_max(x);
  shift(id);
  int mid = st + en >> 1;
  update_max(l, r, x, id << 1, st, mid);
  update_max(l, r, x, id << 1 | 1, mid, en);
  find(id);
}
void update_min(int l, int r, int x, int id = 1, int st = 0, int en = n) {
  if (r <= st || en <= l || seg[id].p.mx1 <= x)
    return;
  if (l <= st && en <= r && seg[id].p.mx2 < x)
    return seg[id].set_min(x);
  shift(id);
  int mid = st + en >> 1;
  update_min(l, r, x, id << 1, st, mid);
  update_min(l, r, x, id << 1 | 1, mid, en);
  find(id);
}
long long get(int l, int r, int id = 1, int st = 0, int en = n) {
  if (r <= st || en <= l)
    return 0;
  if (l <= st && en <= r)
    return seg[id].sum;
  shift(id);
  int mid = st + en >> 1;
  return get(l, r, id << 1, st, mid) + get(l, r, id << 1 | 1, mid, en);
}
```

FenwickTree.cpp

Description: Fenwick tree is an structures that allows compute an assosiative but not invertible functon (Group) in a range [l,r] efficienly
Usage: `bit.resize(n);`
`for(auto &c:nums){cin>>c;add(i++,c);}`
Time: $\mathcal{O}(\log N)$ per query or $\mathcal{O}(\log N^2)$ for bit2D.

5b7e48, 50 lines

```
//Usefull define to print vectors
#define print(A)for(auto c:A)cout<<c<<" ";cout<<endl;
#define printM(A)for(auto c:A){print(c);}
vector<int> bit;
vector<vector<int>> bit2D;
int n,m;
int sum(int idx) {
    int ret = 0;
    for (++idx; idx > 0; idx -= idx & -idx)ret += bit[idx];
    return ret;
}
int sum(int l, int r) {
    return sum(r) - sum(l - 1);
}
void add(int idx, int delta) {
    for (++idx; idx < n; idx += idx & -idx) bit[idx] += delta;
}
/** This only can accept querys in a point */
void range_add(int l, int r, int val) {
    add(l, val);
    add(r + 1, -val);
}
// Search for first position such \sum_{0}^{pos} a[i] \geq s;
int bit_search(int s){
    int sum = 0;
    int pos = 0;
    for(int i = ceil(log2(n));i>=0;i--){
        if((pos+(1<<i))<n && (sum+bit[pos+(1<<i)])<s){
            sum+=bit[pos+(1<<i)];
            pos+=(1<<i);
        }
    }
    return pos;
}
// Return sum over submatrix with corners (0,0), (x,y)
int sum2D(int x, int y) {
    int ret = 0;
```

```
  for (int i = x; i >= 0; i = (i & (i + 1)) - 1)
    for (int j = y; j >= 0; j = (j & (j + 1)) - 1)
      ret += bit2D[i][j];
  return ret;
}
int sum2D(int x0,int y0,int x,int y){
  return sum2D(x,y)-sum2D(x,y0-1)-sum2D(x0-1,y)+sum2D(x0-1,y0-1);
}
void add2D(int x, int y, int delta) {
  for (int i = x; i < n; i = i | (i + 1))
    for (int j = y; j < m; j = j | (j + 1))
      bit2D[i][j] += delta;
}
```

waveletTree.cpp

Description: Binary tree based in values instead of ranges like segment tree, thah alows compute queries in a range like , kth smallest element in a range [l,r], other queries in the code.
Considerations:

- compression if the elements are to big.
- Array passed is modified

Usage: `wavelet wt(Array,Max_element+1);`
Time: $\mathcal{O}(\log N)$.

c283cd, 112 lines

```
typedef vector<int>::iterator it;
struct wavelet{
  vector<vector<int>> mapLeft;
  int mx;
  wavelet(vector<int> &A,int mx):mapLeft(mx*2),mx(mx){
    build(A.begin(),A.end(),0,mx-1,1);
  }
  void build(it s,it e,int l,int r,int v){
    if(l== r)return;
    int m = (l+r)>>1;
    mapLeft[v].reserve(e-s+1);
    mapLeft[v].push_back(0);
    auto f = [m](int x){
      return x<=m;
    };
    for(it iter = s; iter!= e;iter++)
      mapLeft[v].push_back(mapLeft[v].back() + (*iter<=m));
    it p = stable_partition(s,e,f);
    build(s,p,l,m,v<<1);
    build(p,e,m+1,r,v<<1|1);
  }
  //counts the number of elements equal to c in range [1,i]
  //IF you want in the range [i,j] only calls rank(j)- rank(i-1)
  int rank(int c,int i){
    i++;
    int l = 0,r = mx-1,u = 1,m,left;
    while(l!= r){
      m = (l+r)>>1;
      left = mapLeft[u][i];
      u<<=1;
      if(c<=m)
        i = left,r = m;
      else
        i-=left,l = m+1,u|=1;
    }
    return i;
  }
  // return the kth smallest element in a range [i,j]
  // k=1 is the smallest
  // 0 indexed this is indexes are in [0,n-1]
  int kth(int i,int j,int k){
    j++;
    int l = 0,r = mx-1,u = 1,li,lj;
    while(l!=r){
      int m = (l+r)>>1;
      li = mapLeft[u][i],lj = mapLeft[u][j];
      u<<=1;
      if(k<= lj-li)
        i = li,j = lj, r = m;
      else
        i-=li,j-=lj,l = m+1,u|=1,k-=(lj-li);
    }
    return r;
  }
  int l,r;
  // count the ocurrences of numbers in the range [a,b]
  // and only in the secuende [i,j]
  // can be seen as how many points are in a specified rectangle with corns i,a and j,b
  int range(int i ,int j ,int a,int b){
    if( b<a || j<i)return 0;
    l = a,r = b;
    return range(i,j+1,0,mx-1,1);
  }
  int range(int i, int j,int a,int b,int v){
    if(b<l || a>r)return 0;
    if(a>=l && b<=r)return j-i;
    int m = (a+b)>>1;
    int li = mapLeft[v][i],lj = mapLeft[v][j];
    return range(li,lj,a,m,v<<1)+range(i-li,j-lj,m+1,b,v<<1|1);
  }
  /*
    Return the minimun number that their frecuence in the range [i, j] is at least k
    complexity depends of k, if k is small and j-i is large maybe go up to o(n)
    the problem tested has a k up to (j-i)/5 and the complexity has O(5 log n)
  */
  int minimun_of_ocurrences(int i,int j,int k){
    return minimun_of_ocurrences(i,j+1,k,1,0,mx-1);
  }
  int minimun_of_ocurrences(int i,int j,int k,int v ,int l,int r ){
    if(l == r)return j-i>= k?l:mx+2;
    if(j-i<k)return mx+2;
    int m = (l+r)>>1;
    int li = mapLeft[v][i],lj = mapLeft[v][j];
    int c = lj-li;
    int ans= mx+2;
    if(c >=k)
      ans = min(ans,minimun_of_ocurrences(li,lj,k,v<<1,l,m));
    if((j-i)-c>=k)
      ans = min(ans,minimun_of_ocurrences(i-li,j-lj,k,v<<1|1,m+1,r));
    if(c <k && (j-i)-c<k)return mx+2;
    return ans;
  }
  /* swap element arr[i] and arr[i+1] */
  /*- No tested */
  void swapadjacent(int i){
    swapadjacent(i,0,mx-1,1);
  }
  void swapadjacent(int i,int l,int r,int v){
    if(l == r)
      return ;
    mapLeft[v][i]= mapLeft[v][i-1] + mapLeft[v][i+1] - mapLeft[v][i];
    // c[i] = c[i-1] + c[i+1] - c[i];
    if(mapLeft[v][i+1]-mapLeft[v][i] == mapLeft[v][i] - mapLeft[v][i-1]){
      if(mapLeft[v][i]-mapLeft[v][i-1])
        return swapadjacent(mapleft[v][i],l,mid,v<<1);
      else
        return swapadjacent(i-mapLeft[v][i],mid+1,r,v<<1|1);
    }
    else
      return ;
  }
};
```

ImplicitTreap.cpp

Description: A powerfull dynamic array that allows operations like: Insert/erase in every position, Range sum/minimum/max, Reverse/Rotate a sub array
Usage: `Root is global and not need modifications`
`Only erase need root -> erase(root,pos)`
`All operations are 0 indexed`
Time: $\mathcal{O}(\log N)$

df5d5a, 162 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<> dis(numeric_limits<int>::min(), numeric_limits<int>::max()) ;
struct Treap{
```

```
  Treap *l = NULL,*r = NULL;
  int p,sz = 1,val,sum = 0,mn = 1e9;
  int rev = 0, lazySum = 0,lazyReplace = 0;
  int sumPend = 0,ReplacePend = 0;
  Treap(int v ,int prior = dis(rng)):val(v),p(prior) { }
};
Treap *root = NULL;
void update(Treap *T) {
  if (!T) return;
  T->sz = 1;
  T->sum = T->val;
  T->mn = T->val;
  if (T->l)  {
   T->sz += T->l->sz;
   T->sum += T->l->sum;
   T->mn = min(T->mn,T->l->mn);
  }
  if (T->r)  {
   T->sz += T->r->sz;
   T->sum += T->r->sum;
   T->mn = min(T->mn,T->r->mn);
  }
}
void applyRev(Treap *T) {
  if(!T)return;
  T->rev^=1;
  swap(T->l,T->r);
}
void applySum(Treap *T,int x) {
  if(!T)return;
  T->val+=x;
  T->mn+=x;
  T->sumPend+=x;
  T->lazySum = 1;
  T->sum+=x*T->sz;
}
void applyReplace(Treap *T,int x) {
  if(!T)return;
  T->val=x;
  T->mn=x;
  T->ReplacePend=x;
  T->lazyReplace = 1;
  T->sum=x*T->sz;
}
void lazy(Treap *T) {
  if(!T)return;
  if(T->rev) {
     applyRev(T->l);
     applyRev(T->r);
     T->rev = 0;
  }
  if(T->lazySum) {
     applySum(T->l,T->sumPend);
     applySum(T->r,T->sumPend);
     T->lazySum = 0;
     T->sumPend = 0;
  }
  if(T->lazyReplace) {
     applyReplace(T->l,T->ReplacePend);
     applyReplace(T->r,T->ReplacePend);
     T->lazyReplace = 0;
     T->ReplacePend = 0;
  }
}
pair<Treap*,Treap*> split(Treap *T,int idx,int cont = 0) {
  if(!T)return {NULL,NULL};
  lazy(T);
  Treap *L, *R;
  int idxt = cont + (T->l?T->l->sz:0);
  if(idx<idxt)
     tie(L,T->l) = split(T->l,idx,cont),R = T;
  else
     tie(T->r,R) = split(T->r,idx,idxt+1),L = T;
  update(T);
  return {L,R};
}
void insert(Treap *&T,Treap *v,int x, int cnt)  {
  lazy(T);
  int idxt = T ? cnt + (T->l ? T->l->sz : 0) : 0;
  if (!T) T = v;
  else if (v->p > T->p)
     tie(v->l, v->r) = split(T, x, cnt), T = v;
  else if (x < idxt) insert(T->l, v, x, cnt);
  else insert(T->r, v, x, idxt + 1);
  update(T);
}
void insert(int e, int i)  {
  insert(root, new Treap(e), i-1, 0);
}
Treap *merge(Treap *a,Treap *b) {
  lazy(a),lazy(b);
  Treap *T;
  if(!a || !b)T = a?a:b;
  else if(a->p > b->p)
     a->r = merge(a->r,b),T = a;
  else b->l = merge(a,b->l),T = b;
  update(T);
  return T;
}
void erase(Treap *&T,int x ,int cnt = 0) {
  if(!T)return;
  lazy(T);
  int left = cnt+(T->l? T->l->sz:0);
  if(left == x)T = merge(T->l,T->r);
  else if(x<left)erase(T->l,x,cnt);
  else erase(T->r,x,left+1);
  update(T);
}
void print(Treap *t)  {
  if (!t) return;
  lazy(t);
  print(t->l);
  print(t->r);
}
void push_back(int e)  {
  root = merge(root, new Treap(e));
}
void op(int l,int r, function<void(Treap *T)> f) {
  Treap *a,*b,*c;
  tie(a,b) = split(root,l-1);
  tie(b,c) = split(b,r-l);
  f(b);
  root = merge(a, merge(b,c));
}
void reverse(int l,int r) {
  op(l,r,[&](Treap *T) { applyRev(T); } );
}
void rotate(int l,int r,int k) {
  op(l,r,[&](Treap *T) {
     Treap *l,*r;
     k%=T->sz;
     tie(l,r) = split(T,T->sz-k-1);
     T = merge(r,l);
  } );
}
void add(int l,int r,int x) {
  op(l,r,[&](Treap *T) {
     applySum(T,x);
  } );
}
void replace(int l,int r,int x) {
  op(l,r,[&](Treap *T) {
     applyReplace(T,x);
  } );
}
int get_sum(int l,int r) {
  int ans;
  op(l,r,[&](Treap *T) {
     ans = T->sum;
  } );
  return ans;
}
int get_min(int l,int r) {
```

```
  int mn;
  op(l,r,[&](Treap *T){
    mn = T->mn;
  });
  return mn;
}
```

UnionFind.cpp

Description: Disjoint-set data structure.
Usage: `iota(padre.begin(),padre.end(),0);`
Time: $\mathcal{O}(\alpha(N))$

daebb2, 16 lines

```
const int maxn = 100007;
vector<int> padre(maxn);
vector<int> sz(maxn);
int raiz(int v){
  return v== padre[v]?v:padre[v] = raiz(padre[v]);
}
void union_bySize(int u,int v){
  u = raiz(u);
  v = raiz(v);
  if (u != v) {
    if (sz[u] < sz[v])
      swap(u, v);
    padre[v] = u;
    sz[u] += sz[v];
  }
}
```

UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: `int t = uf.time(); ...; uf.rollback(t);`
Time: $\mathcal{O}(\log(N))$

150d07, 21 lines

```
struct RollbackUF {
  vector<int> e; vector<pair<int,int>> st;
  RollbackUF(int n) : e(n, -1) { }
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back( { a, e[a] } );
    st.push_back( { b, e[b] } );
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

SubMatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
Time: $\mathcal{O}(N^2+Q)$

c59ada, 13 lines

```
template<class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u][l];
  }
};
```

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
Time: $\mathcal{O}(\log N)$

af1807, 29 lines

```
struct Line {
  mutable int k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(int x) const { return p < x; }
};
struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const int inf = intONG_MAX;
  int div(int a, int b) { // floored division
    return a / b - ((a ^b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(int k, int m) {
    auto z = insert( { k, m, 0 } ), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  int query(int x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

SparseTable.cpp

Description: Sparse table is similar to segment tree but don´t allows updates
Time: $\mathcal{O}(1)$ per query, $\mathcal{O}(N\log N)$ build

966c2f, 17 lines

```
const int maxn = 100007;
int ST[maxn][25];
int lg[maxn];
void build(vector<int> &A){
  lg[1] = 0;
    for(int i = 2;i<maxn;i++)
    lg[i] = lg[i/2]+1;
  for(int i = 0;i<n;i++)
      ST[i][0] = A[i];
    for(int j = 1;j<25;j++)
    for(int i=0; i+(1<<j)<=n; i++)
      ST[i][j] = __gcd(ST[i][j-1],ST[i+(1<<(j-1))][j-1]);
}
int query(int l, int r){
  int j = lg[r-l+1];
  return __gcd(ST[l][j],ST[r-(1<<j)+1][j]);
}
```

MoQueries.h

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: $\mathcal{O}(N\sqrt{Q})$

a12ef4, 47 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/√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;
}
```

QuadTree.cpp

Description: A divide and conquer structure that divides a plane in fou sections to answer efficiently queries like how many points are inside of a rectangle

Usage: `node* head = new node(range(0,MAXN,0,MAXN));`
`count(range(S[i].x,MAXN-S[i].x,S[i].y,MAXN-S[i].y),head); Range is an structure for a rectangle`

Time: $\mathcal{O}(n \log N)$.

458ccc, 68 lines

```
struct point{
  int x,y;
  point(int _x,int _y):x(_x),y(_y){}
};
int capacity = 4;
struct range{
  int x,y,w,h; // w is width and h is height,x and y are the left
    ↪ upper corner
  range(int _x,int _w,int _y,int _h):x(_x),y(_y),w(_w),h(_h){}
  bool contains(point p){
    if( p.x >= x &&
      p.x <= x + w &&
      p.y >= y &&
      p.y <= y + h)
      return true;
    return false;
  }
  bool intersects(range R){
    return !(R.x > x+w ||
         R.x+R.w < x ||
         R.y > y+h ||
         R.y+R.h < y
    );
  }
};
struct node{
  range boundary;
  node(range bound):boundary(bound){}
  bool divided = false;
  vector<point> P;
  node *nw = NULL,*ne = NULL,*sw = NULL,*se = NULL;
  void divide(){
    divided = true;
    nw = new node(range(0,boundary.w/2,boundary.h/2,boundary.h/2));
    ne = new node(range(boundary.w/2,boundary.w/2,boundary.h/2,
      ↪ boundary.h/2));
    sw = new node(range(0,boundary.w/2,0,boundary.h/2));
    se = new node(range(boundary.w,boundary.w/2,0,boundary.h/2));
  }
};
int MAXN = 65536/4;
bool insert(point p,node *N){
  if(!N->boundary.contains(p))return false;
  if(!N->P.size()<capacity){
    N->P.push_back(p);
    return true;
  }
  else{
    if(!N->divided)N->divide();
    if(insert(p,N->nw))return true;
    if(insert(p,N->ne))return true;
    if(insert(p,N->sw))return true;
    if(insert(p,N->se))return true;
  }
  return true;
}
int count (range R,node *N){
  int ans = 0;
  if(!N->boundary.intersects(R))return 0;
  for(auto p:N->P){
    if(R.contains(p))ans++;
  }
  if(N->divided){
    ans+=count(R,N->nw);
    ans+=count(R,N->ne);
    ans+=count(R,N->sw);
    ans+=count(R,N->se);
  }
  return ans;
}
```

IntervalTree.cpp

Description: Interval tree is a structure that stores segments in a efficient way, allows to get all intervals that intersects with another interval

Usage: `root = build_interval_tree(vector<recta>);`
`query(root,R); R is an instance of recta`

Time: $\mathcal{O}(ans)$ where ans is the number of intervals that intersects.

5df955, 105 lines

```
typedef long long int lli;
typedef long double ld;
struct recta {
  ld x1,x2;
  int id;
  friend ostream& operator << (ostream &out, const recta&p ){
    out<<"("<<p.x1<<","<<p.x2<<", "<<p.id<<")";
    return out;
  }
};
struct central{
  ld x;
  vector<recta> x1order;
  vector<recta> x2order;
  friend ostream& operator <<(ostream &out, const central&p){
    out<<"[ ";
    for(int i = 0;i<p.x1order.size();i++){
      out<<p.x1order[i]<<" ";
    }
    out<<"]";
    return out;
  }
};
struct node {
 node *l, *r;
  central C;
 node(node *l, node *r, central C) :
  l(l), r(r),C(C) {}
};
inline bool leaf(node *x){
  return !x->l && !x->r;
}
node* build_interval_tree(vector<recta> &R){
  if(R.size() == 0)return NULL;
  int n = R.size();
  int mid = (n-1)>>1;
 vector<recta> r1,r2;
  central c;
  ld x = (R[mid].x1+R[mid].x2)/2.0;
  c.x = x;
  for(int i = 0;i<n;i++){
    if(islessequal(R[i].x1,x) && islessequal(x,R[i].x2)){
```

```
        c.x1order.push_back(R[i]);
        c.x2order.push_back(R[i]);
      }
      else if(R[i].x2<x)
        r1.push_back(R[i]);
      else
        r2.push_back(R[i]);
    }
    sort(c.x1order.begin(),c.x1order.end(),[&](recta a,recta b){
      return islessequal(a.x1,b.x1);
    });
    sort(c.x2order.begin(),c.x2order.end(),[&](recta a,recta b){
      return islessequal(a.x2,b.x2);
    });
  node *left = build_interval_tree(r1);
  node *right = build_interval_tree(r2);
  return new node(left,right,c);
}
set<lli> ids;
void findI(central C,recta R,bool dir){
  if(dir){
    int l = -1, r = C.x2order.size();
    while(l+1<r){
      int m = (l+r)>>1;
      if(isgreaterequal(C.x2order[m].x2,R.x1))
        r = m;
      else
        l = m;
    }
    int n = C.x2order.size();
    for(int i = r;i<n;i++)
      ids.insert(C.x2order[i].id);
  }
  else{
    int l = -1, r = C.x2order.size();
    while(l+1<r){
      int m = (l+r)>>1;
      if(islessequal(C.x1order[m].x1,R.x2))
        l = m;
      else
        r = m;
    }
    for(int i = l;i>=0;i--)
      ids.insert(C.x1order[i].id);
  }
}
void query(node *t, const recta& R){
  if(!t)return;
  if(isgreaterequal(t->C.x,R.x1) && islessequal(t->C.x,R.x2)){
    for(int i = 0;i<t->C.x1order.size();i++)
      ids.insert(t->C.x1order[i].id);
    query(t->l,R);
    query(t->r,R);
  }
  else if(isless(R.x2,t->C.x)){
    findI(t->C,R,0);
    query(t->l,R);
  }
  else{
    findI(t->C,R,1);
    query(t->r,R);
  }
}
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`
Time: $\mathcal{O}(n^2 \log(1/\epsilon))$

"Polynomial.h" b00bfe, 23 lines

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^(p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

PolyInterpolate.h

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

08bf48, 13 lines

```
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;
}
```

BerlekampMassey.h

Description: Recovers any n-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.
Usage: `berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}`
Time: $\mathcal{O}(N^2)$

cbc8b5, 20 lines

```
#include "ModPow()"
vector<int> berlekampMassey(vector<int> s) {
  int n = sz(s), L = 0, m = 0;
  vector<int> C(n), B(n), T;
  C[0] = B[0] = 1;
  int b = 1;
  for(int i = 0;i<n;i++){
    ++m;
    int d = s[i] % mod;
    for(int j = 1;j<L+1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C; int coef = d * modpow(b, mod-2) % mod;
    for(int j = m;j<n;j++) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
```

```
  L = i + 1 - L; B = T; b = d; m = 0;
 }
 C.resize(L + 1); C.erase(C.begin());
 for (int& x : C) x = (mod - x) % mod;
 return C;
}
```

LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0\ldots \geq n-1]$ and $tr[0\ldots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`
Time: $\mathcal{O}\left(n^2 \log k\right)$

f4e444, 22 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
 int n = sz(tr);
 auto combine = [&](Poly a, Poly b) {
  Poly res(n * 2 + 1);
  rep(i,0,n+1) rep(j,0,n+1)
   res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
  for (int i = 2 * n; i > n; --i) rep(j,0,n)
   res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
  res.resize(n + 1);
  return res;
 };
 Poly pol(n + 1), e(pol);
 pol[0] = e[1] = 1;
 for (++k; k; k /= 2) {
  if (k % 2) pol = combine(pol, e);
  e = combine(e, e);
 }
 ll res = 0;
 rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
 return res;
}
```

ExtendedPolynomial.cpp

Description: A complete structure for polynomial and his operations

3aa30e, 229 lines

```
const int N = 3e5 + 9, mod = 998244353;
struct base {
 double x, y;
 base() { x = y = 0; }
 base(double x, double y): x(x), y(y) { }
};
inline base operator + (base a, base b) { return base(a.x + b.x, a.y + b.y); }
inline base operator - (base a, base b) { return base(a.x - b.x, a.y - b.y); }
inline base operator * (base a, base b) { return base(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
inline base conj(base a) { return base(a.x, -a.y); }
int lim = 1;
vector<base> roots = { {0, 0}, {1, 0} };
vector<int> rev = {0, 1};
const double PI = acosl(- 1.0);
void ensure_base(int p) {
 if(p <= lim) return;
 rev.resize(1 << p);
 for(int i = 0; i < (1 << p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (p - 1));
 roots.resize(1 << p);
 while(lim < p) {
  double angle = 2 * PI / (1 << (lim + 1));
  for(int i = 1 << (lim - 1); i < (1 << lim); i++) {
   roots[i << 1] = roots[i];
   double angle_i = angle * (2 * i + 1 - (1 << lim));
   roots[(i << 1) + 1] = base(cos(angle_i), sin(angle_i));
  }
  lim++;
 }
}
void fft(vector<base> &a, int n = -1) {
 if(n == -1) n = a.size();
 assert((n & (n - 1)) == 0);
 int zeros = __builtin_ctz(n);
 ensure_base(zeros);
 int shift = lim - zeros;
 for(int i = 0; i < n; i++) if(i < (rev[i] >> shift)) swap(a[i], a[rev[i] >> shift]);
 for(int k = 1; k < n; k <<= 1) {
  for(int i = 0; i < n; i += 2 * k) {
   for(int j = 0; j < k; j++) {
    base z = a[i + j + k] * roots[j + k];
    a[i + j + k] = a[i + j] - z;
    a[i + j] = a[i + j] + z;
   }
  }
 }
}
//eq = 0: 4 FFTs in total
//eq = 1: 3 FFTs in total
vector<int> multiply(vector<int> &a, vector<int> &b, int eq = 0) {
 int need = a.size() + b.size() - 1;
 int p = 0;
 while((1 << p) < need) p++;
 ensure_base(p);
 int sz = 1 << p;
 vector<base> A, B;
 if(sz > (int)A.size()) A.resize(sz);
 for(int i = 0; i < (int)a.size(); i++) {
  int x = (a[i] % mod + mod) % mod;
  A[i] = base(x & ((1 << 15) - 1), x >> 15);
 }
 fill(A.begin() + a.size(), A.begin() + sz, base{0, 0});
 fft(A, sz);
 if(sz > (int)B.size()) B.resize(sz);
 if(eq) copy(A.begin(), A.begin() + sz, B.begin());
 else {
  for(int i = 0; i < (int)b.size(); i++) {
   int x = (b[i] % mod + mod) % mod;
   B[i] = base(x & ((1 << 15) - 1), x >> 15);
  }
  fill(B.begin() + b.size(), B.begin() + sz, base{0, 0});
  fft(B, sz);
 }
 double ratio = 0.25 / sz;
 base r2(0, - 1), r3(ratio, 0), r4(0, - ratio), r5(0, 1);
 for(int i = 0; i <= (sz >> 1); i++) {
  int j = (sz - i) & (sz - 1);
  base a1 = (A[i] + conj(A[j])), a2 = (A[i] - conj(A[j])) * r2;
  base b1 = (B[i] + conj(B[j])) * r3, b2 = (B[i] - conj(B[j])) * r4;
  if(i != j) {
   base c1 = (A[j] + conj(A[i])), c2 = (A[j] - conj(A[i])) * r2;
   base d1 = (B[j] + conj(B[i])) * r3, d2 = (B[j] - conj(B[i])) * r4;
   A[i] = c1 * d1 + c2 * d2 * r5;
   B[i] = c1 * d2 + c2 * d1;
  }
  A[j] = a1 * b1 + a2 * b2 * r5;
  B[j] = a1 * b2 + a2 * b1;
 }
 fft(A, sz); fft(B, sz);
 vector<int> res(need);
 for(int i = 0; i < need; i++) {
  long long aa = A[i].x + 0.5;
  long long bb = B[i].x + 0.5;
  long long cc = A[i].y + 0.5;
  res[i] = (aa + ((bb % mod) << 15) + ((cc % mod) << 30))%mod;
 }
 return res;
}
template <int32_t MOD>
struct modint {
 int32_t value;
 modint() = default;
 modint(int32_t value_) : value(value_) { }
 inline modint<MOD> operator + (modint<MOD> other) const { int32_t c = this->value + other.value; return modint<MOD>(c >= MOD ? c - MOD : c); }
 inline modint<MOD> operator - (modint<MOD> other) const { int32_t c = this->value - other.value; return modint<MOD>(c < 0 ? c + MOD : c); }
```

```cpp
 inline modint<MOD> operator * (modint<MOD> other) const { int32_t c = (int64_t)this->value * other.value % MOD; return modint<MOD>(c < 0 ? c + MOD : c); }
 inline modint<MOD> & operator += (modint<MOD> other) { this->value += other.value; if (this->value >= MOD) this->value -= MOD; return *this; }
 inline modint<MOD> & operator -= (modint<MOD> other) { this->value -= other.value; if (this->value < 0) this->value += MOD; return *this; }
 inline modint<MOD> & operator *= (modint<MOD> other) { this->value = (int64_t)this->value * other.value % MOD; if (this->value < 0) this->value += MOD; return *this; }
 inline modint<MOD> operator - () const { return modint<MOD>(this->value ? MOD - this->value : 0); }
 modint<MOD> pow(uint64_t k) const {
  modint<MOD> x = *this, y = 1;
  for (; k; k >>= 1) {
   if (k & 1) y *= x;
   x *= x;
  }
  return y;
 }
 modint<MOD> inv() const { return pow(MOD - 2); } // MOD must be a prime
 inline modint<MOD> operator / (modint<MOD> other) const { return *this * other.inv(); }
 inline modint<MOD> operator /= (modint<MOD> other) { return *this *= other.inv(); }
 inline bool operator == (modint<MOD> other) const { return value == other.value; }
 inline bool operator != (modint<MOD> other) const { return value != other.value; }
 inline bool operator < (modint<MOD> other) const { return value < other.value; }
 inline bool operator > (modint<MOD> other) const { return value > other.value; }
};
template <int32_t MOD> modint<MOD> operator * (int64_t value, modint<MOD> n) { return modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD> operator * (int32_t value, modint<MOD> n) { return modint<MOD>(value % MOD) * n; }
template <int32_t MOD> ostream & operator << (ostream & out, modint<MOD> n) { return out << n.value; }
using mint = modint<mod>;
struct poly {
 vector<mint> a;
 inline void normalize() {
  while((int)a.size() && a.back() == 0) a.pop_back();
 }
 template<class... Args> poly(Args... args): a(args...) { }
 poly(const initializer_list<mint> &x): a(x.begin(), x.end()) { }
 int size() const { return (int)a.size(); }
 inline mint coef(const int i) const { return (i < a.size() && i >= 0) ? a[i]: mint(0); }
 mint operator[](const int i) const { return (i < a.size() && i >= 0) ? a[i]: mint(0); } //Beware!! p[i] = k won't change the value of p.a[i]
 bool is_zero() const {
  for (int i = 0; i < size(); i++) if (a[i] != 0) return 0;
  return 1;
 }
 poly operator + (const poly &x) const {
  int n = max(size(), x.size());
  vector<mint> ans(n);
  for(int i = 0; i < n; i++) ans[i] = coef(i) + x.coef(i);
  while ((int)ans.size() && ans.back() == 0) ans.pop_back();
  return ans;
 }
 poly operator - (const poly &x) const {
  int n = max(size(), x.size());
  vector<mint> ans(n);
  for(int i = 0; i < n; i++) ans[i] = coef(i) - x.coef(i);
  while ((int)ans.size() && ans.back() == 0) ans.pop_back();
  return ans;
 }
 poly operator * (const poly& b) const {
  if(is_zero() || b.is_zero()) return {};
  vector<int> A, B;
  for(auto x: a) A.push_back(x.value);
  for(auto x: b.a) B.push_back(x.value);
  auto res = multiply(A, B, (A == B));
  vector<mint> ans;
  for(auto x: res) ans.push_back(mint(x));
  while ((int)ans.size() && ans.back() == 0) ans.pop_back();
  return ans;
 }
 poly operator * (const mint& x) const {
  int n = size();
  vector<mint> ans(n);
  for(int i = 0; i < n; i++) ans[i] = a[i] * x;
  return ans;
 }
 poly operator / (const mint &x) const{ return (*this) * x.inv(); }
 poly& operator += (const poly &x) { return *this = (*this) + x; }
 poly& operator -= (const poly &x) { return *this = (*this) - x; }
 poly& operator *= (const poly &x) { return *this = (*this) * x; }
 poly& operator *= (const mint &x) { return *this = (*this) * x; }
 poly& operator /= (const mint &x) { return *this = (*this) / x; }
 poly mod_xk(int k) const { return {a.begin(), a.begin() + min(k, size())}; } //modulo by x^k
 poly mul_xk(int k) const { // multiply by x^k
  poly ans(*this);
  ans.a.insert(ans.a.begin(), k, 0);
  return ans;
 }
 poly div_xk(int k) const { // divide by x^k
  return vector<mint>(a.begin() + min(k, (int)a.size()), a.end());
 }
 poly substr(int l, int r) const { // return mod_xk(r).div_xk(l)
  l = min(l, size());
  r = min(r, size());
  return vector<mint>(a.begin() + l, a.begin() + r);
 }
 poly differentiate() const {
  int n = size(); vector<mint> ans(n);
  for(int i = 1; i < size(); i++) ans[i - 1] = coef(i) * i;
  return ans;
 }
 poly integrate() const {
  int n = size(); vector<mint> ans(n + 1);
  for(int i = 0; i < size(); i++) ans[i + 1] = coef(i) / (i + 1);
  return ans;
 }
 poly inverse(int n) const { // 1 / p(x) % x^n, O(nlogn)
  assert(!is_zero()); assert(a[0] != 0);
  poly ans{mint(1) / a[0]};
  for(int i = 1; i < n; i *= 2) {
   ans = (ans * mint(2) - ans * ans * mod_xk(2 * i)).mod_xk(2 * i);
  }
  return ans.mod_xk(n);
 }
 poly log(int n) const { //ln p(x) mod x^n
  assert(a[0] == 1);
  return (differentiate().mod_xk(n) * inverse(n)).integrate().mod_xk(n);
 }
 poly exp(int n) const { //e ^p(x) mod x^n
  if(is_zero()) return {1};
  assert(a[0] == 0);
  poly ans({1});
  int i = 1;
  while(i < n) {
   poly C = ans.log(2 * i).div_xk(i) - substr(i, 2 * i);
   ans -= (ans * C).mod_xk(i).mul_xk(i);
   i *= 2;
  }
  return ans.mod_xk(n);
 }
};
```

SubsetSum.cpp

Description: Some aplications of functions of polynomial

"ExtendedPolynomial" a3bd8d, 45 lines

```
// number of subsets of an array of n elements having sum equal to k
    ↪ for each k from 1 to m
int main() {
  int n, m; cin >> n >> m;
  vector<int> a(m + 1, 0);
  for (int i = 0; i < n; i++) {
    int k; cin >> k; // k >= 1, handle [k = 0] separately
    if (k <= m) a[k]++;
  }
  poly p(m + 1, 0);
  for (int i = 1; i <= m; i++) {
    for (int j = 1; i * j <= m; j++) {
      if (j & 1) p.a[i * j] += mint(a[i]) / j;
      else p.a[i * j] -= mint(a[i]) / j;
    }
  }
  p = p.exp(m + 1);
  for (int i = 1; i <= m; i++) cout << p[i] << ' '; cout << '\n'; //
      ↪ check for m = 0
  return 0;
}
// Calc bell numbers
vector<mint> bell(int n) { // e^(e^x - 1)
  poly p(n + 1);
  mint f = 1;
  for (int i = 0; i <= n; i++) {
    p.a[i] = mint(1) / f;
    f *= i + 1;
  }
  p.a[0] -= 1;
  p = p.exp(n + 1);
  vector<mint> ans(n + 1);
  f = 1;
  for (int i = 0; i <= n; i++) {
    ans[i] = p[i] * f;
    f *= i + 1;
  }
  return ans;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; cin >> n;
  auto ans = bell(n);
  cout << ans[n] << '\n';
  return 0;
}
```

4.2 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`

Time: $\mathcal{O}(\log((b-a)/\epsilon))$

31d45b, 14 lines

```
double gss(double a, double b, double (*f)(double)) {
  double r = (sqrt(5)-1)/2, eps = 1e-7;
  double x1 = b - r*(b-a), x2 = a + r*(b-a);
  double f1 = f(x1), f2 = f(x2);
  while (b-a > eps)
    if (f1 < f2) { //change to > to find maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - r*(b-a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
}
```

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.

Usage: `vvd A = {{1,-1}, {-1,1}, {-1,-2}};`
`vd b = {1,1,-4}, c = {-1,-1}, x;`
`T val = LPSolver(A, b, c).solve(x);`

Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.

aa8530, 62 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;
  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }
  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }
  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }
  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

4.3 Matrices

Template.cpp

Description: A template for Matrix structure with all basic operations like exponensation ,sum , rest, determinant ,reduction by gauss etc

Usage: `Matrix<type> M`

fea995, 379 lines

```cpp
#define mod 1e9+7
#define INF INT_MAX
const double EPS = 1e-9;
typedef long long int lli;
template <typename T>
struct Matrix {
   vector < vector <T> > A;
   int r,c;
   Matrix(){
      this->r = 0;
      this->c = 0;
   }
   Matrix(int r,int c){
      this->r = r;
      this->c = c;
      A.assign(r , vector <T> (c));
   }
   Matrix(int r,int c,const T &val){
      this->r = r;
      this->c = c;
      A.assign(r , vector <T> (c , val));
   }
   Matrix(int n){
      this->r = this->c = n;
      A.assign(n , vector <T> (n));
      for(int i=0;i<n;i++)
         A[i][i] = (T)1;
   }
   Matrix operator * (const Matrix<T> &B){
//   Matrix <T> C(r,B.c,0);
//   for(int i=0 ; i<r ; i++)
//      for(int j=0 ; j<B.c ; j++)
//         for(int k=0 ; k<c ; k++)
//            C[i][j] = (C[i][j] + ( (long long )A[i][k] * (long long
    ↪ )B[k][j] ));
//   return C;
      Matrix<T> C(r,B.c,0);
      for(int i = 0;i<r;i++){
         for(int j = 0;j<B.c;j++){
            for(int k = 0;k<c;k++){
               C[i][j] = (C[i][j] + ((lli)A[i][k] * (lli)B[k][j]));
               if(C[i][j]>= 8ll*mod*mod)
                  C[i][j]%=mod;
            }
         }
      }
      for(int i = 0;i<r;i++)for(int j = 0;j<c;j++)C[i][j]%=mod;
      return C;
   }
   Matrix operator + (const Matrix<T> &B){
      assert(r == B.r);
      assert(c == B.c);
      Matrix <T> C(r,c,0);
      int i,j;
      for(i=0;i<r;i++)
         for(j=0;j<c;j++)
            C[i][j] = ((A[i][j] + B[i][j]));
      return C;
   }
   Matrix operator*(int & c) {
      Matrix<T> C(r, c);
      for(int i = 0; i < r; i++)
         for(int j = 0; j < c; j++)
            C[i][j] = A[i][j] * c;
      return C;
   }
   Matrix operator - (){
      Matrix <T> C(r,c,0);
      int i,j;
      for(i=0;i<r;i++)
         for(j=0;j<c;j++)
            C[i][j] = -A[i][j];
      return C;
   }
   Matrix operator - (const Matrix<T> &B){
      assert(r == B.r);
      assert(c == B.c);
      Matrix <T> C(r,c,0);
      int i,j;
      for(i=0;i<r;i++)
         for(j=0;j<c;j++)
            C[i][j] = A[i][j] - B[i][j];
      return C;
   }
   Matrix operator ^(long long n){
      assert(r == c);
      int i,j;
      Matrix <T> C(r);
      Matrix <T> X(r,c,0);
      for(i=0;i<r;i++)
         for(j=0;j<c;j++)
            X[i][j] = A[i][j];
      while(n){
         if(n&1)
            C *= X;
         X *= X;
         n /= 2;
      }
      return C;
   }
   vector<T>& operator [] (int i){
      assert(i < r);
      assert(i >= 0);
      return A[i];
   }
   const vector<T>& operator [] (int i) const{
      assert(i < r);
      assert(i >= 0);
      return A[i];
   }
   friend ostream& operator << (ostream &out,const Matrix<T> &M){
      for (int i = 0; i < M.r; ++i) {
         for (int j = 0; j < M.c; ++j) {
            out << M[i][j] << " ";
         }
         out << '\n';
      }
      return out;
   }
   void operator *= (const Matrix<T> &B){
      (*this) = (*this)*B;
   }
   void operator += (const Matrix<T> &B){
      (*this) = (*this)+B;
   }
   void operator -= (const Matrix<T> &B){
      (*this) = (*this)-B;
   }
   void operator ^= (long long n){
      (*this) =(*this)^n;
   }
   //Inverse
   bool Inverse(Matrix<double> &inverse){
      if(this->detGauss() == 0)return false;
      int n = A[0].size();
      Matrix<double> temp(n,2*n);
      for(int i = 0;i<n;i++)
         for(int j = 0;j<n;j++)temp[i][j] = A[i][j];
      Matrix<double> ident(n);
      for(int i = 0;i<n;i++)
         for(int j = n;j<2*n;j++)temp[i][j] = ident[i][j-n];
      int m = n*2;
      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 (temp[i][col]) > abs (temp[sel][col]))
               sel = i;
         if (abs (temp[sel][col]) < EPS)
            continue;
         for (int i=col; i<m; ++i)
            swap (temp[sel][i], temp[row][i]);
         where[col] = row;
```

```
            double div = temp[row][col];
            for(int i = 0;i<m;i++)
                if(fabs(temp[row][i])>EPS)temp[row][i] /=div;
            for (int i=0; i<n; ++i)
                if (i != row) {
                    double c = temp[i][col] / temp[row][col];
                    for (int j=col; j<m; ++j)
                        temp[i][j] -= temp[row][j] * c;
                }
            ++row;
        }
        for(int i = 0;i<n;i++)
            for(int j = 0;j<n;j++)
                inverse[i][j] = temp[i][j+n];
        return true;
    }
    //Adjoint
    Matrix<T> minor(int x, int y){
        Matrix<T> M(r-1, c-1);
        for(int i = 0; i < c-1; ++i)
            for(int j = 0; j < r-1; ++j)
                M[i][j] = A[i < x ? i : i+1][j < y ? j : j+1];
        return M;
    }
    T cofactor(int x, int y){
        T ans = minor(x, y).detGauss();
        if((x + y) % 2 == 1) ans *= -1;
        return ans;
    }
    Matrix<T> cofactorMatrix(){
        Matrix<T> C(r, c);
        for(int i = 0; i < c; i++)
            for(int j = 0; j < r; j++)
                C[i][j] = cofactor(i, j);
        return C;
    }
    Matrix<T> Adjunta(){
        int n = A[0].size();
        Matrix<int> adjoint(n);
        Matrix<double> inverse(n);
        this->Inverse(inverse);
        int determinante = this->detGauss();
        if(determinante){
            for(int i = 0;i<n;i++)
                for(int j = 0;j<n;j++)
                    adjoint[i][j] = (T)round((inverse[i][j]*determinante));
        }
        else {
            adjoint = this->cofactorMatrix().transpose();
        }
        return adjoint;
    }
    //Transpuesta
    Matrix transpose(){
        Matrix <T> C(c,r);
        int i,j;
        for(i=0;i<r;i++)
            for(j=0;j<c;j++)
                C[j][i] = A[i][j];
        return C;
    }
    //Traza
    T trace(){
        T sum = 0;
        for(int i = 0; i < min(r, c); i++)
            sum += A[i][i];
        return sum;
    }
    //Determinante
    int determinant() {
        int n = r;
        Matrix<T> temp(n);
        temp.A = A;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                temp[i][j] %= mod;
        lli res = 1;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                for (; temp[j][i]; res = -res) {
                    long long t = temp[i][i] / temp[j][i];
                    for (int k = i; k < n; k++) {
                        temp[i][k] = (temp[i][k] - temp[j][k] * t) % mod;
                        std::swap(temp[j][k], temp[i][k]);
                    }
                }
            }
            if (temp[i][i] == 0)
                return 0;
            res = res * temp[i][i] % mod;
        }
        if (res < 0)
            res += mod;
        return static_cast<int>(res);
    }
    int detGauss(){
        assert(r == c);
        double det = 1;
        Matrix<double> temp(r);
        temp.r = r;
        temp.c = c;
        int n = r;
        for(int i = 0;i<n;i++)
            for(int j = 0;j<n;j++)
                temp[i][j] = (double)A[i][j];
        for (int i=0; i<n; ++i) {
            int k = i;
            for (int j=i+1; j<n; ++j)
                if (fabs (temp[j][i]) > fabs (temp[k][i]))
                    k = j;
            if (abs (temp[k][i]) < EPS) {
                det = 0;
                break;
            }
            swap (temp[i], temp[k]);
            if (i != k)
                det = -det;
            det *= temp[i][i];
            for (int j=i+1; j<n; ++j)
                temp[i][j] /= temp[i][i];
            for (int j=0; j<n; ++j)
                if (j != i && abs (temp[j][i]) > EPS)
                    for (int k=i+1; k<n; ++k)
                        temp[j][k] -= temp[i][k] * temp[j][i];
        }
        return (int)det;
    }
    int gauss (vector<double> & ans) {
        Matrix<double> Temp(this->r,this->c);
        int n = (int) Temp.A.size();
        int m = (int) Temp[0].size() - 1;
        for(int i = 0;i<n;i++)
            for(int j = 0;j<n;j++)
                Temp[i][j] = (double)A[i][j];
        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 (fabs (Temp[i][col]) > fabs (Temp[sel][col]))
                    sel = i;
            if (fabs (Temp[sel][col]) < EPS)
                continue;
            for (int i=col; i<m; ++i)
                swap (Temp[sel][i], Temp[row][i]);
            where[col] = row;
            for (int i=0; i<n; ++i)
                if (i != row) {
                    double c = Temp[i][col] / Temp[row][col];
                    for (int j=col; j<m; ++j)
                        Temp[i][j] -= Temp[row][j] * c;
                }
            ++row;
```

```
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
      if (where[i] != -1)
        ans[i] = Temp[where[i]][m] / Temp[where[i]][i];
    for (int i=0; i<n; ++i) {
      double sum = 0;
      for (int j=0; j<m; ++j)
        sum += ans[j] * Temp[i][j];
      if (fabs (sum - Temp[i][m]) > EPS)
        return 0;
    }
    for (int i=0; i<m; ++i)
      if (where[i] == -1)
        return INF;
    return 1;
  }
  /*+ Kirchhoff Matrix Tree Theorem Describe in Graphs -> Math */
  int Kirchof(){
    cin>>n>>m>>k;
    Matrix<lli> Kirchof(n);
    for(int i = 0;i<m;i++){
      cin>>a>>b;
      a--;
      b--;
      Kirchof[a][b] = Kirchof[b][a] = 1;
      Kirchof[a][a]++;
      Kirchof[b][b]++;
    }
    for(int i =0;i<n;i++)
      Kirchof[i][i] = (1ll*n*k%mod-Kirchof[i][i]+mod)%mod;
    lli ans = 1;
    ans = ans*(mod_pow(1ll*k*n%mod*k%mod*n%mod,mod-2));
    lli determinante =Kirchof.det();
    ans = ans*(mod_pow(determinante,k))%mod;
    cout<<ans<<endl;
  }
};
/*+   [f(n)]      [1 1 1 1 1 1] [f(5)]
      [f(n-1)]    [1 0 0 0 0 0] [f(4)]
      [f(n-2)]    [0 1 0 0 0 0] [f(3)]
      [f(n-3)]    [0 0 1 0 0 0] [f(2)]
      [f(n-4)]    [0 0 0 1 0 0] [f(1)]
      [(e)]       [0 0 0 0 1 0] [ e ]
*/
lli Linear_recurrence(vector<lli> C, vector<lli> init,lli n,bool
    ↪ constante){
  int k = C.size();
  Matrix<lli> T(k,k);
  Matrix<lli> first(k,1);
  for(int i = 0;i<k;i++)T[0][i] = C[i];
  for(int i = 0,col=1;i<k && col<k;i++,col++)
    T[col][i]=1;
  if(constante){
    for(int i = 0;i<k;i++)first[i][0]=init[(k-2)-i];
    first[k-1][0]=init[k-1];
  }
  else
    for(int i = 0;i<k;i++)first[i][0]=init[(k-1)-i];
  if(constante)
    T^=((n-k)+1);
  else
    T^=(n-k);
  Matrix<lli> sol = T*first;
  return sol[0][0];
  //Example Tribonacci F(i) = 1*F(i-1) + 1*F(i-2) + 1*F(i-3) + (c=
      ↪ 0)
  // vector<lli>C(3);
  // C[0] =1;
  // C[1] =1;
  // C[2] =1;
  // vector<lli> ini(3);
  // ini[0] =1;
  // ini[1] =1;
  // ini[2] =2;
  // cout<<Linear_recurrence(C,ini,nth,false)<<endl;
}
```

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}\left(N^3\right)$

bd5cec, 15 lines

```
double det(vector<vector<double>>& a) {
 int n = sz(a); double res = 1;
 rep(i,0,n) {
  int b = i;
  rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
  if (i != b) swap(a[i], a[b]), res *= -1;
  res *= a[i][i];
  if (res == 0) return 0;
  rep(j,i+1,n) {
   double v = a[j][i] / a[i][i];
   if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
  }
 }
 return res;
}
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}\left(N^3\right)$

3313dc, 18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
 int n = sz(a); ll ans = 1;
 rep(i,0,n) {
  rep(j,i+1,n) {
   while (a[j][i] != 0) { // gcd step
    ll t = a[i][i] / a[j][i];
    if (t) rep(k,i,n)
     a[i][k] = (a[i][k] - a[j][k] * t) % mod;
    swap(a[i], a[j]);
    ans *= -1;
   }
  }
  ans = ans * a[i][i] % mod;
  if (!ans) return 0;
 }
 return (ans + mod) % mod;
}
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}\left(n^2 m\right)$

44c9ab, 35 lines

```
typedef vector<double> vd;
const double eps = 1e-12;
int solveLinear(vector<vd>& A, vd& b, vd& x) {
 int n = sz(A), m = sz(x), rank = 0, br, bc;
 if (n) assert(sz(A[0]) == m);
 vi col(m); iota(all(col), 0);
 rep(i,0,n) {
  double v, bv = 0;
  rep(r,i,n) rep(c,i,m)
   if ((v = fabs(A[r][c])) > bv)
    br = r, bc = c, bv = v;
  if (bv <= eps) {
   rep(j,i,n) if (fabs(b[j]) > eps) return -1;
   break;
  }
  swap(A[i], A[br]);
  swap(b[i], b[br]);
  swap(col[i], col[bc]);
  rep(j,0,n) swap(A[j][i], A[j][bc]);
  bv = 1/A[i][i];
  rep(j,i+1,n) {
   double fac = A[j][i] * bv;
   b[j] -= fac * b[i];
   rep(k,i+1,m) A[j][k] -= fac*A[i][k];
  }
  rank++;
 }
 x.assign(m, 0);
 for (int i = rank; i--;) {
```

```
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

87bc77, 8 lines

```
#include "SolveLinear.h"
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}\left(n^2 m\right)$

fa2d7a, 32 lines

```
typedef bitset<1000> bs;
int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }
  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}\left(n^3\right)$

ebfff6, 32 lines

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;
  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }
  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

Tridiagonal.h

Description: $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i,\ 1 \le i \le n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1, -1, -1, ..., -1, 1\}, \{0, c_1, c_2, \dots, c_n\}, \{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}).$$

Fails if the solution is not unique. If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$

8f9fa8, 26 lines

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
  int n = sz(b); vi tr(n);
  rep(i,0,n-1) {
    if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
      b[i+1] -= b[i] * diag[i+1] / super[i];
      if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
      diag[i+1] = sub[i]; tr[++i] = 1;
    } else {
      diag[i+1] -= super[i]*sub[i]/diag[i];
      b[i+1] -= b[i]*sub[i]/diag[i];
    }
  }
  for (int i = n; i--;) {
    if (tr[i]) {
      swap(b[i], b[i-1]);
      diag[i-1] = diag[i];
      b[i] /= super[i-1];
    } else {
      b[i] /= diag[i];
      if (i) b[i-1] -= b[i]*super[i-1];
    }
  }
  return b;
}
```

4.4 Fourier transforms

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.
Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

26e1c6, 23 lines

```
#include "FastFourierTransform.h"
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
```

```
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

NumberTheoreticTransform.h

Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For arbitrary modulo, see FFTMod. `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in [0, mod).

Usage: `vector<int> X(n), Y(m);`
```
for(int i = 0; i < n; i++) s[i]== c?X[i] = 1:X[i] = 0;
for(int i = 0; i < m; i++) t[i]== c?Y[i] = 1:Y[i] = 0;
reverse(Y.begin(),Y.end());
mult<998244353, 3>(X, Y);
```
Time: $\mathcal{O}(N \log N)$

f699dc, 61 lines

```
const double PI = acos(-1.0L);
using lli = int64_t;
using comp = complex<long double>;
#define print(A)for(auto c:A)cout<<c<<" ";cout<<endl;
#define printc(A)for(auto c:A)cout<<c.real()<<" ";cout<<endl;
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
#define endl '\n'
typedef vector<comp> vec;
int nearestPowerOfTwo(int n){
  int ans = 1;
  while(ans < n) ans <<= 1;
  return ans;
}
lli powerMod(lli b, lli e, lli m){
  lli ans = 1;
  e %= m-1;
  if(e < 0) e += m-1;
  while(e){
    if(e & 1) ans = ans * b % m;
    e >>= 1;
    b = b * b % m;
  }
  return ans;
}
template<int p, int g>
void ntt(vector<int> & X, int inv){
  int n = X.size();
  for(int i = 1, j = 0; i < n - 1; ++i){
    for(int k = n >> 1; (j ^= k) < k; k >>= 1);
    if(i < j) swap(X[i], X[j]);
  }
  vector<lli> wp(n>>1, 1);
  for(int k = 1; k < n; k <<= 1){
    lli wk = powerMod(g, inv * (p - 1) / (k<<1), p);
    for(int j = 1; j < k; ++j)
      wp[j] = wp[j - 1] * wk % p;
    for(int i = 0; i < n; i += k << 1){
      for(int j = 0; j < k; ++j){
        int u = X[i + j], v = X[i + j + k] * wp[j] % p;
        X[i + j] = u + v < p ? u + v : u + v - p;
        X[i + j + k] = u - v < 0 ? u - v + p : u - v;
      }
    }
  }
  if(inv == -1){
    lli nrev = powerMod(n, p - 2, p);
    for(int i = 0; i < n; ++i)
      X[i] = X[i] * nrev % p;
  }
}
template<int p, int g>
void mult(vector<int> &A, vector<int> &B){
  int sz = A.size() + B.size() - 1;
  int size = nearestPowerOfTwo(sz);
  A.resize(size), B.resize(size);
  ntt<p, g>(A, 1), ntt<p, g>(B, 1);
  for(int i = 0; i < size; i++)
    A[i] = (lli)A[i] * B[i] % p;
  ntt<p, g>(A, -1);
  A.resize(sz);
}
```

FastSubsetTransform.h

Description: Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x\oplus y} a[x] \cdot b[y]$, where $\oplus$ is one of AND, OR, XOR. The size of a must be a power of two.

Time: $\mathcal{O}(N \log N)$

464cf3, 16 lines

```
void FST(vi& a, bool inv) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
      int &u = a[j], &v = a[j + step]; tie(u, v) =
        inv ? pii(v - u, u) : pii(v, u + v); // AND
        inv ? pii(v, u - v) : pii(u + v, u); // OR
        pii(u + v, u - v);         // XOR
    }
  }
  if (inv) for (int& x : a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
  FST(a, 0); FST(b, 0);
  rep(i,0,sz(a)) a[i] *= b[i];
  FST(a, 1); return a;
}
```

FWHT.cpp

Description: Gets an xor convolution. if you have two arrays like (1,2,2) and (3,4,5) ,all possible xor results in (1,1,2,4,5,6,6,7,7)

2e0eac, 71 lines

```
const int N = 3e5 + 9, mod = 1e9 + 7;
int POW(long long n, long long k) {
  int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
const int inv2 = (mod + 1) >> 1;
#define M (1 << 20)
#define OR 0
#define AND 1
#define XOR 2
struct FWHT{
  int P1[M], P2[M];
  void wt(int *a, int n, int flag = XOR) {
    if (n == 0) return;
    int m = n / 2;
    wt(a, m, flag); wt(a + m, m, flag);
    for (int i = 0; i < m; i++){
      int x = a[i], y = a[i + m];
      if (flag == OR) a[i] = x, a[i + m] = (x + y) % mod;
      if (flag == AND) a[i] = (x + y) % mod, a[i + m] = y;
      if (flag == XOR) a[i] = (x + y) % mod, a[i + m] = (x - y + mod)
        ↪ % mod;
    }
  }
  void iwt(int* a, int n, int flag = XOR) {
    if (n == 0) return;
    int m = n / 2;
    iwt(a, m, flag); iwt(a + m, m, flag);
    for (int i = 0; i < m; i++){
      int x = a[i], y = a[i + m];
      if (flag == OR) a[i] = x, a[i + m] = (y - x + mod) % mod;
      if (flag == AND) a[i] = (x - y + mod) % mod, a[i + m] = y;
      if (flag == XOR) a[i] = 1LL * (x + y) * inv2 % mod, a[i + m] = 1
        ↪ LL * (x - y + mod) * inv2 % mod; // replace inv2 by >>1
        ↪ if not required
```

```
    }
  }
  vector<int> multiply(int n, vector<int> A, vector<int> B, int flag =
      ↪ XOR) {
    assert(__builtin_popcount(n) == 1);
    A.resize(n); B.resize(n);
    for (int i = 0; i < n; i++) P1[i] = A[i];
    for (int i = 0; i < n; i++) P2[i] = B[i];
    wt(P1, n, flag); wt(P2, n, flag);
    for (int i = 0; i < n; i++) P1[i] = 1LL * P1[i] * P2[i] % mod;
    iwt(P1, n, flag);
    return vector<int> (P1, P1 + n);
  }
  vector<int> pow(int n, vector<int> A, long long k, int flag = XOR)
      ↪ {
    assert(__builtin_popcount(n) == 1);
    A.resize(n);
    for (int i = 0; i < n; i++) P1[i] = A[i];
    wt(P1, n, flag);
    for(int i = 0; i < n; i++) P1[i] = POW(P1[i], k);
    iwt(P1, n, flag);
    return vector<int> (P1, P1 + n);
  }
} t;
int32_t main() {
  int n; cin >> n;
  vector<int> a(M, 0);
  for(int i = 0; i < n; i++) {
    int k; cin >> k; a[k]++;
  }
  vector<int> v = t.pow(M, a, n, AND);
  int ans = 1;
  for(int i = 1; i < M; i++) ans += v[i] > 0;
  cout << ans << '\n';;
  return 0;
}
```

4.5 XOR

BasisXor.cpp

Description: Structure to form a basis in Z2 that allows compute things around xor because xor is a sum in Z2 like the maxxor possible , minimum , number of diferent xor´s, kth possible xor

Time: $\mathcal{O}(\log N)$

76e346, 29 lines

```
struct Basis {
  vector<int> a;
  void insert(int x) {
    for (auto &i: a) x = min(x, x ^i);
    if (!x) return;
    for (auto &i: a) if ((i ^x) < i) i ^= x;
    a.push_back(x);
    sort(a.begin(), a.end());
  }
  bool can(int x) {
    for (auto &i: a) x = min(x, x ^i);
    return !x;
  }
  int maxxor(int x = 0) {
    for (auto &i: a) x = max(x, x ^i);
    return x;
  }
  int minxor(int x = 0) {
    for (auto &i: a) x = min(x, x ^i);
    return x;
  }
  int kth(int k) { // 1st is 0
    int sz = (int)a.size();
    if (k > (1LL << sz)) return -1;
    k--; int ans = 0;
    for (int i = 0; i < sz; i++) if (k >> i & 1) ans ^= a[i];
    return ans;
  }
} t;
```

Number theory (5)

5.1 Modular arithmetic

ModInverse.h

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

259876, 15 lines

```
int inverse(int a, int m){
  int x, y ;
   if isPrime(m)return mod_pow(a,m-2,m);
  if(gcd( a, m, x, y ) != 1) return 0; // not all numbers has inverse
      ↪ modulo m
  return (x%m + m) % m;
}
/** All inverse (1 to p-1)%p p is prime*/
vector<int> allinverse(int p){
  vector<int> ans(p);
  ans[1] = 1;
  for(int i = 2;i<p:i++){
    ans[i] = p-(p/i)*ans[p%i]%p;
  }
  return ans;
}
```

ModPow.h

dba434, 10 lines

```
const int mod = 1e9+7;
int modpow(int a,int b){
  int x = 1;
  while(b){
    if(b&1) (x*=a)%=mod;
    (a*=a)%=mod;
    b>>=1;
  }
  return x;
}
```

ModLog.h

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: $\mathcal{O}\left(\sqrt{m}\right)$

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;
}
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.

f(a, b, c, n) $= \sum_{i=0}^{\text{to}-1} (ki+c)\%m.$

Time: $\log(m)$, with a large constant.

5c5bc5, 14 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}
ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModFloorDivision.h

Description: Sum of aritmetic floor division

f(a, b, c, n) $= \sum_{i=0}^{n} \lfloor \frac{(ai+b)}{c} \rfloor.$

Time: $\log(a)$,

f58bf7, 16 lines

```
int f(int a, int b, int c, int n){
  int m = (a*n + b)/c;
  if(n==0 || m==0) return b/c;
  if(n==1) return b/c + (a+b)/c;
  if(a<c && b<c) return m*n - f(c, c-b-1, a, m-1);
  else return (a/c)*n*(n+1)/2 + (b/c)*(n+1) + f(a%c, b%c, c, n);
```

```
}
// $\sum_{k=1}^{n} \lfloor \frac{n}{k} \rfloor$
int floor_sum(int n) {
 int sum = 0;
 for (int i = 1, last; i <= n; i = last + 1) {
  last = n / (n / i);
  sum += (n / i) * (last - i + 1);
 }
 return sum;
}
```

ModSqrt.h

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

"ModPow.h" 19a793, 24 lines

```
ll sqrt(ll a, ll p) {
 a %= p; if (a < 0) a += p;
 if (a == 0) return 0;
 assert(modpow(a, (p-1)/2, p) == 1); // else no solution
 if (p % 4 == 3) return modpow(a, (p+1)/4, p);
 // $\frac{a^{n+3}}{8}$ or $\frac{2^{n+3}}{8} * \frac{2^{n-1}}{4}$ 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;
 }
}
```

LinealDiophantine.cpp

Description: A Linear Diophantine Equation (in two variables) is an equation of the general form: $ax + by = c$ where a,b,c are given integers, and , are unknown integers.

5af435, 63 lines

```
int gcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
   }
  int x1, y1;
  int d = gcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
  g = gcd(abs(a), abs(b), x0, y0);
  if (c % g) {
    return false;
   }
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
void shift_solution(int & x, int & y, int a, int b, int cnt) {
  x += cnt * b;
  y -= cnt * a;
}
int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
  int x, y, g;
  if (!find_any_solution(a, b, c, x, y, g))
    return -1;
  a /= g;
  b /= g;
  int sign_a = a > 0 ? +1 : -1;
  int sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx)
    shift_solution(x, y, a, b, sign_b);
  if (x > maxx)
    return -1;
  int lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx)
    shift_solution(x, y, a, b, -sign_b);
  int rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny)
    shift_solution(x, y, a, b, -sign_a);
  if (y > maxy)
    return -1;
  int lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy)
    shift_solution(x, y, a, b, sign_a);
  int rx2 = x;
  if (lx2 > rx2)
    swap(lx2, rx2);
  int lx = max(lx1, lx2);
  int rx = min(rx1, rx2);
  if (lx > rx)
    return -1;
  return lx;
}
```

5.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s also other fast sieves for diferent purposes

6e889d, 107 lines

```
const int LIM = 1e7;
bitset<LIM> isPrime;
vector<int> eratosthenes() {
 const int S = (int)round(sqrt(LIM)), R = LIM / 2;
 vector<int> 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.push_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;
  rep(i,0,min(S, R - L))
   if (!block[i]) pr.push_back((L + i) * 2 + 1);
 }
 for (int i : pr) isPrime[i] = 1;
 return pr;
}
// More easy linear sieve also gets sieve of function $\mu$
const int N = 1000007;
vector<int> m(N+1);
void criba(){
 vector<int> lp(N+1);
 vector<int> primes;
 m[1]= 1;
 for(int i = 2;i<=N;i++){
  if(lp[i]== 0){
   primes.push_back(i);
   lp[i]= i;
   m[i] = -1;
  }
  for(int j = 0;j<primes.size()&& primes[j]<=lp[i] && primes[j]*i<=N;j++){
   lp[primes[j]*i] = primes[j];
   if(lp[i]==primes[j])m[primes[j]*i]= 0;
   else m[primes[j]*i] = m[primes[j]]*m[i];
```

```cpp
    }
  }
}
// Greatest prime sieve
vector<int> gp;
void greatestPrimeSieve(int n){
  gp.resize(n + 1, 1);
  gp[0] = gp[1] = 0;
  for(int i = 2; i <= n; ++i) gp[i] = i;
  for(int i = 2; i <= n; i++)
    if(gp[i] == i)
      for(int j = i; j <= n; j += i)
        gp[j] = i;
}
// Segmented sieve , get primes in range[L,R] with complexity O(max(
  ↪ sqrt(R)log(sqrt(R)),R-L)) ??? ;
// Also is one of the fastest sieve get all primes in range [1-n]
  ↪ with n = 1^9 in 8.62s and for
// n = 1^8 in 0.76s
vector<int> PrimesInRange;
void calcPrimes(int l ,int r) {
  auto sum = l<=2?2:0;
  if(l<=2)PrimesInRange.push_back(2);
  int cnt = 1;
  const int S = round(sqrt(r));
  vector<char> sieve(S + 1, true);
  vector<array<int, 2>> cp;
  for (int i = 3; i <= S; i += 2) {
    if (!sieve[i])
      continue;
    cp.push_back({i, (i * i - 1) / 2});
    for (int j = i * i; j <= S; j += 2 * i){
      sieve[j] = false;
    }
  }
  vector<char> block(S);
  int high = (r - 1) / 2;
  int x = l/S;
  int L = (x/2)*S;
  for(auto &i:cp){
    int p = i[0],idx = i[1];
    if(idx>L){
      i[1]-=L;
    }
    else{
      int X = (L-idx)/p;
      if((L-idx)%p)X++;
      if(X>=1 && idx<=L)
        i[1] = (idx+(p*X))-L;
    }
  }
  for (int low =(x/2)*S; low <= high; low += S) {
    fill(block.begin(), block.end(), true);
    for (auto &i : cp) {
      int p = i[0], idx = i[1];
      for (; idx < S; idx += p){
        block[idx] = false;
      }
      i[1] = idx - S;
    }
    if (low == 0)
      block[0] = false;
    for (int i = 0; i < S && low + i <= high; i++){
      if (block[i] && (((low+i)*2)+1)>=l){
        // push the primes here if needed
        ++cnt, sum += (low + i) * 2 + 1;
      }
    }
  };
  // cout << "sum = " << sum << endl;
  // cout << "cnt = " << cnt << endl;
}
```

MillerRabin.h

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 $a^b \mod c$.

573e3b, 23 lines

```cpp
typedef unsigned long long ull;
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;
}
```

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"MillerRabin.h" a33cf6, 18 lines

```cpp
ull pollard(ull n) {
  auto f = [n](ull x) { return modmul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

fastPrimeCount.cpp

Description: Count the number of primes up to n^{12}

Usage: `k = 0; gen(); lehmer(n) // k = 0 is count of primes`
`for sum of primes -> k = 1; count_primes(n);`

Time: $\mathcal{O}\left(n^{\frac{2}{3}}\right)$ count of primes for $n^{12} \sim 5.15s$ for sum of primes $n^{11} \sim 4.39s$

c65a97, 123 lines

```cpp
// If sum of primes is needed use int128
#define int __int128
#define MAXN 100
#define MAXM 100007
#define MAXP 10000007
int prime_cnt[MAXP];
int prime_sum[MAXP];
long long dp[MAXN][MAXM];
//Function to print __int128
std::ostream&
operator<<( std::ostream& dest, __int128_t value ){
  std::ostream::sentry s( dest );
  if ( s ) {
    __uint128_t tmp = value < 0 ? -value : value;
    char buffer[ 128 ];
    char* d = std::end( buffer );
    do
    {
      -- d;
      *d = "0123456789"[ tmp % 10 ];
      tmp /= 10;
    } while ( tmp != 0 );
```

```
    if ( value < 0 ) {
      -- d;
      *d = '-';
    }
    int len = std::end( buffer ) - d;
    if ( dest.rdbuf()->sputn( d, len ) != len ) {
      dest.setstate( std::ios_base::badbit );
    }
  }
  return dest;
}
vector<int> primes;
bitset<MAXP> is_prime;
// modify k to calc sum of primes p^k with p^k <= n
int k = 0;
int F(int n){
  return pow(n,k);
};
int pref(int n){
  if(k == 0)return n;
  if(k == 1)return (n*(n+1))/2;
  if(k == 2)return (n*(n+1)*(2*n+1))/6;
  return 1;
}
vector<int> lp(MAXP+1);
void gen(){
  lp.assign(MAXP,0);
  primes.clear();
  for(int i= 2;i<=MAXP;i++){
    if(lp[i]==0)lp[i] = i,primes.push_back(i);
    for(int j = 0;j<primes.size() && primes[j]<=lp[i] && primes[j]*i<=MAXP;j++)
      lp[primes[j]*i] = primes[j];
  }
  for (int i = 2; i < MAXP; i++){
    prime_cnt[i] = prime_cnt[i-1] + (lp[i]==i);
    prime_sum[i] = prime_sum[i-1] + (lp[i]==i?F(i):0);
  }
  for (int m = 0; m < MAXM; m++) dp[0][m] = pref(m);
  for (int n = 1; n < MAXN; n++){
    for (int m = 0; m < MAXM; m++){
      dp[n][m] = dp[n - 1][m] - (dp[n - 1][m/primes[n - 1]]*F(primes[n-1]));
    }
  }
}
int phi(int m, int n){
  if (n == 0) return pref(m);
  if (m < MAXM && n < MAXN) return dp[n][m];
  if (primes[n - 1] * primes[n - 1] >= m && m < MAXP) return prime_sum[m] - pref(n) + 1;
  return phi(m, n - 1) - (phi(m/primes[n - 1], n - 1)*F(primes[n-1]));
}
/*- for some reason this not work for sum of power primes or for k>=1 use count primes instead */
int lehmer(int m){
  if (m < MAXP) return prime_sum[m];
  int s = sqrt(0.5 + m), y = cbrt(0.5 + m);
  int a = prime_cnt[y];
  int res = phi(m, a);
  for (int i = a; primes[i] <= s; i++){
    int x = lehmer(m/primes[i]);
    int y = lehmer(primes[i-1]);
    res = res - ((lehmer(m / primes[i]) - lehmer(primes[i-1]))*F(primes[i])) - F(primes[i]);
  }
  a = prime_sum[s];
  return (res+a)-1;
}
/*+ Use this function if k>=1 */
int count_primes(int n) {
  vector<int> v;
  v.reserve((int)sqrt(n) * 2 + 20);
  int sq; {
    int k = 1;
    for (; k * k <= n; ++k) {
      v.push_back(k);
    }
    --k;
    sq = k;
    if (k * k == n) --k;
    for (; k >= 1; --k) {
      v.push_back(n / k);
    }
  }
  vector<int> s(v.size());
  for (int i = 0; i < s.size(); ++i)
    s[i] = pref(v[i]) - 1;
  auto geti = [&](int x) {
    if (x <= sq) return (int)x - 1;
    else return (int)(v.size() - (n / x));
  };
  for (int p = 2; p * p <= n; ++p) {
    if (s[p - 1] != s[p - 2]) {
      int sp = s[p - 2];
      int p2 = p * p;
      for (int i = (int)v.size() - 1; i >= 0; --i) {
        if (v[i] < p2) {
          break;
        }
        s[i] -= (s[geti(v[i] / p)] - sp) * F(p);
      }
    }
  }
  return s.back();
}
```

5.3 Divisibility

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 $a \pmod{b}$.

1ea825, 5 lines

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

FastCountDivisors.cpp

Description: Count the number of divisors of a large number

Time: $\mathcal{O}\left(n^{\frac{1}{3}}\right)$

"MillerRabin", "primes" 13e60e, 93 lines

```
/*+  Need primes[],lp[],N= 10^6 */
#define lli long long
bool isSquare(lli val){
  lli lo = 1, hi = val;
  while(lo <= hi){
    lli mid = lo + (hi - lo) / 2;
    lli tmp = (val / mid) / mid; // be careful with overflows!!
    if(tmp == 0)hi = mid - 1;
    else if(mid * mid == val)return true;
    else if(mid * mid < val)lo = mid + 1;
  }
  return false;
}
lli countDivisors(lli n) {
  lli ans = 1;
  for(int i = 0; i < primes.size(); i++){
    if(n == 1)break;
    int p = primes[i];
    if(n % p == 0){ // checks whether p is a divisor of n
      int num = 0;
      while(n % p == 0){
        n /= p;
        ++num;
      }
      // p^num divides initial n but p^{num+1} does not divide initial val
      // => p can be taken 0 to num times => num + 1 possibilities!!
      ans *= num + 1;
    }
  }
  if(n == 1)return ans; // first case
```

```cpp
  else if(isPrime(n))return ans * 2; // second case
  else if(isSquare(n))return ans * 3; // third case but with p == q
  else return ans * 4; // third case with p != q
}
using uint32 = unsigned int;
using uint64 = unsigned long long;
using uint128 = __uint128_t;
// compute \sum_{i=1}^n \sigma(i) in O(n^{1/3}) time.
// it is also equal to \sum_{i=1}^{n} floor(n / i)
// takes ~100 ms for n = 1e18
uint128 sum_sigma0(uint64 n) {
  auto out = [n] (uint64 x, uint32 y) {
    return x * y > n;
  };
  auto cut = [n] (uint64 x, uint32 dx, uint32 dy) {
    return uint128(x) * x * dy >= uint128(n) * dx;
  };
  const uint64 sn = sqrtl(n);
  const uint64 cn = pow(n, 0.34); //cbrtl(n);
  uint64 x = n / sn;
  uint32 y = n / x + 1;
  uint128 ret = 0;
  stack<pair<uint32, uint32>> st;
  st.emplace(1, 0);
  st.emplace(1, 1);
  while (true) {
    uint32 lx, ly;
    tie(lx, ly) = st.top();
    st.pop();
    while (out(x + lx, y - ly)) {
    ret += x * ly + uint64(ly + 1) * (lx - 1) / 2;
    x += lx, y -= ly;
    }
    if (y <= cn) break;
    uint32 rx = lx, ry = ly;
    while (true) {
    tie(lx, ly) = st.top();
    if (out(x + lx, y - ly)) break;
    rx = lx, ry = ly;
    st.pop();
    }
    while (true) {
    uint32 mx = lx + rx, my = ly + ry;
    if (out(x + mx, y - my)) {
      st.emplace(lx = mx, ly = my);
    }
    else {
      if (cut(x + mx, lx, ly)) break;
      rx = mx, ry = my;
    }
    }
    }
    for (--y; y > 0; --y) ret += n / y;
    return ret * 2 - sn * sn;
}
auto ans = sum_sigma0(n);
string s = "";
while (ans > 0) {
  s += char('0' + ans % 10);
  ans /= 10;
}
reverse(s.begin(), s.end());
cout << s << '\n';
```

CRT.h

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 $0 \le x < \mathrm{lcm}(m, n)$. Assumes $mn < 2^{62}$. given a set of congruence equations -> $a \equiv a_1 (mod p_1)$ $a \equiv a_2 (mod p_2)$ ... $a \equiv a_k (mod p_k)$ Return a if p_i are pairwise coprimes
Time: $\log(n)$

"ModInverse.h", "euclid.h" bce171, 40 lines

```cpp
int crt(int a, int m, int b, int n) {
  if (n > m) swap(a, b), swap(m, n);
  int 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;
}
int lcm(int a,int b){
  return a*b/__gcd(a,b);
}
vector<int>nums;
vector<int>rem;
int CRT() {
    int prod = 1;
    for (int i = 0; i < nums.size(); i++)
      prod *= nums[i];
    int result = 0;
    for (int i = 0; i < nums.size(); i++) {
      int pp = prod / nums[i];
      result += rem[i] * inverse(pp, nums[i]) * pp;
    }
    return result % prod;
}
/*+ general CRT if pi,p2,p3 no coprimes, return 0 if no solution */
inline int normalize(int x, int mod) { x %= mod; if (x < 0) x += mod;
    return x; }
vector<int> a;
vector<int> p;
int LCM;
int CRT(int &ans){
    int t =a.size();
    ans = a[0];
    LCM = p[0];
    for(int i = 1; i < t; i++){
      int x1,d= gcd(LCM, p[i],x1,d);
      if((a[i] - ans) % d != 0) return 0;
      ans = normalize(ans + x1 * (a[i] - ans) / d % (p[i] / d) * LCM,
          LCM * p[i] / d);
      LCM = lcm(LCM, p[i]); // you can save time by replacing above
          LCM * n[i] /d by LCM = LCM * n[i] / d
    }
    return 1;
}
```

5.4 SOS

SOSConvolutions.cpp

113 lines

```cpp
#include<bits/stdc++.h>
using namespace std;

const int N = 3e5 + 9, mod = 998244353;

// s' $ s defines all subsets of s
namespace SOS {
const int B = 20; // Every input vector must need to be of size 1<<B
// z(f(s))=\sum_{s' $ s}{f(s')}
// O(B * 2 ^B)
// zeta transform is actually SOS DP
vector<int> zeta_transform(vector<int> f) {
  for (int i = 0; i < B; i++) {
    for (int mask = 0; mask < (1 << B); mask++) {
      if ((mask & (1 << i)) != 0) {
        f[mask] += f[mask ^(1 << i)];// you can change the operator
            from + to min/gcd to find min/gcd of all f[submasks]
      }
    }
  }
  return f;
}
// mu(f(s))=\sum_{s' $ s}{(-1)^|s\s'| * f(s')}
// O(B * 2 ^B)
vector<int> mobius_transform(vector<int> f) {
  for (int i = 0; i < B; i++) {
    for (int mask = 0; mask < (1 << B); mask++) {
      if ((mask & (1 << i)) != 0) {
        f[mask] -= f[mask ^(1 << i)];
      }
    }
  }
  return f;
}
vector<int> inverse_zeta_transform(vector<int> f) {
```

```
  return mobius_transform(f);
}
vector<int> inverse_mobius_transform(vector<int> f) {
  return zeta_transform(f);
}
// z(f(s))=\sum_{s' is supermask of s}{f(s')}
// O(B * 2 ^B)
// zeta transform is actually SOS DP
vector<int> zeta_transform_for_supermasks(vector<int> f) {
  for (int i = 0; i < B; i++) {
    for (int mask = (1 << B) - 1 ; mask >= 0 ; mask--) {
      if ((mask & (1 << i)) == 0) f[mask] += f[mask ^(1 << i)] ;
    }
  }
  return f;
}
// f*g(s)=sum_{s' $ s}  {f(s')*g(s\s')}
// O(B * B * 2 ^B)
vector<int> subset_sum_convolution(vector<int> f, vector<int> g) {
  vector< vector<int> > fhat(B + 1, vector<int> (1 << B, 0));
  vector< vector<int> > ghat(B + 1, vector<int> (1 << B, 0));
  // Make fhat[][] = {0} and ghat[][] = {0}
  for (int mask = 0; mask < (1 << B); mask++) {
    fhat[__builtin_popcount(mask)][mask] = f[mask];
    ghat[__builtin_popcount(mask)][mask] = g[mask];
  }
  // Apply zeta transform on fhat[][] and ghat[][]
  for (int i = 0; i <= B; i++) {
    for (int j = 0; j <= B; j++) {
      for (int mask = 0; mask < (1 << B); mask++) {
        if ((mask & (1 << j)) != 0) {
          fhat[i][mask] += fhat[i][mask ^(1 << j)];
          if (fhat[i][mask] >= mod) fhat[i][mask] -= mod;
          ghat[i][mask] += ghat[i][mask ^(1 << j)];
          if (ghat[i][mask] >= mod) ghat[i][mask] -= mod;
        }
      }
    }
  }
  vector< vector<int> > h(B + 1, vector<int> (1 << B, 0));
  // Do the convolution and store into h[][] = {0}
  for (int mask = 0; mask < (1 << B); mask++) {
    for (int i = 0; i <= B; i++) {
      for (int j = 0; j <= i; j++) {
        h[i][mask] += 1LL * fhat[j][mask] * ghat[i - j][mask] % mod;
        if (h[i][mask] >= mod) h[i][mask] -= mod;
      }
    }
  }
  // Apply inverse SOS dp on h[][]
  for (int i = 0; i <= B; i++) {
    for (int j = 0; j <= B; j++) {
      for (int mask = 0; mask < (1 << B); mask++) {
        if ((mask & (1 << j)) != 0) {
          h[i][mask] -= h[i][mask ^(1 << j)];
          if (h[i][mask] < 0) h[i][mask] += mod;
        }
      }
    }
  }
  vector<int> fog(1 << B, 0);
  for (int mask = 0; mask < (1 << B); mask++) fog[mask] = h[
      ↪ __builtin_popcount(mask)][mask];
  return fog;
}
};

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n;
  cin >> n;
  vector<int> a(1 << 20, 0), b(1 << 20, 0);
  for (int i = 0; i < (1 << n); i++) cin >> a[i];
  for (int i = 0; i < (1 << n); i++) cin >> b[i];
  auto ans = SOS::subset_sum_convolution(a, b);
  for (int i = 0; i < (1 << n); i++) cout << ans[i] << ' ';
  cout << '\n';
  return 0;
}
```

5.4.1 Bézout's identity

For $a \neq$, $b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

phiFunction.h

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 $\Rightarrow a^{p-1} \equiv 1 \pmod{p}$ $\forall a$.

2ab231, 10 lines

```
vector<int> Phi;
void phiSieve(int n){
    Phi.resize(n + 1);
    for(int i = 1; i <= n; ++i)
        Phi[i] = i;
    for(int i = 2; i <= n; ++i)
        if(Phi[i] == i)
            for(int j = i; j <= n; j += i)
                Phi[j] -= Phi[j] / i;
}
```

5.5 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$.
For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.
Time: $\mathcal{O}(\log N)$

dd6c5e, 21 lines

```
typedef double d;
pair<ll, ll> approximate(d x, ll N) {
  ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
  for (;;) {
    ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
       a = (ll)floor(y), b = min(a, lim),
       NP = b*P + LP, NQ = b*Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-convergent that gives us a
      // better approximation; if b = a/2, we *may* have one.
      // Return {P, Q} here for a more canonical approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (d)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

5.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

5.7 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

5.8 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

5.9 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:

$\sum_{d|n} \mu(d) = [n = 1]$ (very useful)

$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$

$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$

Combinatorial (6)

6.1 Permutations

6.1.1 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;
}
```

6.1.2 Cycles

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

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

6.1.3 Derangements

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

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

6.1.4 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

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

6.2 Partitions and subsets

6.2.1 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

6.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.

6.2.3 Binomials

multinomial.h

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

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;
}
```

BinomialCoeficients.cpp

Description: A few ways to calc a binomial coeficient with diferent complexityes
Time: Varios complexities

e85f96, 82 lines

```
long binomial_Coeff_without_MOD(int n,int r){
    long ans = 1;
    for(int i = 1;i<=min(n-k,k);i++){
      ans = (ans* (n-(i-1)))/i;
    }
    return ans;
}
/* O(n) solutions
    Based in the prof of C(n,k) = C(n-1,k-1) + C(n-1,k)
    Also calc all C(n,i) for 0<=i<=n
*/
long binomial_Coeff(int n,int m){
  int i,j;
  long bc[MAXN][MAXN];
  for (i=0; i<=n; i++) bc[i][0] = 1;
  for (j=0; j<=n; j++) bc[j][j] = 1;
  for (i=1; i<=n; i++)
    for (j=1; j<i; j++)
      bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
  return bc[n][m];
}
/*
    O(k) solution
    Only calc C(n,k)
*/
int binomial_Coeff_2(int n, int k) {
    int res = 1;
    if ( k > n - k )
      k = n - k;
    for (int i = 0; i < k; ++i){
      res *= (n - i);
      res /= (i + 1);
    }
    return res;
}
/* Factorial modulo P */
int factmod(int n, int p) {
    int res = 1;
    while (n > 1) {
      res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
      for (int i = 2; i <= n%p; ++i)
        res = (res * i) % p;
      n /= p;
    }
    return res % p;
}
/** O(1) binomial coeficient with precalc in O(n) */
const int M = 1e6;
```

```
const lli mod = 986444681;
vector<lli> fact(M+1, 1), inv(M+1, 1), invfact(M+1, 1);
lli ncr(lli n, lli r) {
 if(r < 0 || r > n) return 0;
 return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}
void calc() {
  for(int i = 2; i <= M; ++i) {
  fact[i] = (lli)fact[i-1] * i % mod;
  inv[i] = mod - (lli)inv[mod % i] * (mod / i) % mod;
  invfact[i] = (lli)invfact[i-1] * inv[i] % mod;
 }
}
/** Lucas Theorem: Computes C(N,R)%p in O(log(n)) if P is prime */
/** call calc() first */
lli Lucas(lli N,lli R) {
 if(R<0 || R>N)
  return 0;
 if(R==0 || R==N)
  return 1ll;
 if(N >= mod)
  return (1ll*Lucas(N/mod,R/mod)*Lucas(N%mod,R%mod))%mod;
 return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}
/* Using calc() we can also calculate P(n,k) (permutations) */
lli permutation(int n,int k) {
  return (1ll*fact[n]* invfact[n-k])%mod;
}
/** Cayley's formula: Computes all posibles trees whit n nodes */
lli cayley(int n ,int k) {
  if(n-k-1<0)
    return (1ll*k*modpow(n,mod-2))%mod;
  return (1ll*k*modpow(n,n-k-1))%mod;
}
```

6.3 General purpose numbers

6.3.1 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:

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

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

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

6.3.3 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. $\pi(j) > j$.

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

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

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

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.5 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,

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

6.3.6 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: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

6.3.7 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.

Graph (7)

7.1 Fundamentals

BellmanFord.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max |w_i| < 2^{63}$.
Time: $\mathcal{O}(VE)$

830a8f, 21 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; } };
struct Node { ll dist = inf; int prev = -1; };
void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
 nodes[s].dist = 0;
 sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });
 int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
 rep(i,0,lim) for (Ed ed : eds) {
  Node cur = nodes[ed.a], &dest = nodes[ed.b];
  if (abs(cur.dist) == inf) continue;
  ll d = cur.dist + ed.w;
  if (d < dest.dist) {
   dest.prev = ed.a;
   dest.dist = (i < lim-1 ? d : -inf);
  }
 }
 rep(i,0,lim) for (Ed e : eds) {
  if (nodes[e.a].dist == -inf)
   nodes[e.b].dist = -inf;
 }
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] =$ `inf` if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$

ccbb1e, 17 lines

```
const int inf = 1LL << 62;
void floydWarshall(vector<vector<int>>& m) {
 int n = sz(m);
 for(int i = 0;i<n;i++) m[i][i] = min(m[i][i], 0LL);
 for(int k = 0;k<n;k++)
  for(int i = 0;i<n;i++)
   for(int j = 0;j<n;j++)
    if (m[i][k] != inf && m[k][j] != inf) {
     auto newDist = max(m[i][k] + m[k][j], -inf);
     m[i][j] = min(m[i][j], newDist);
```

```
      }
  for(int k = 0;k<n;k++)
    if(m[k][k] < 0)
      for(int i = 0;i<n;i++)
        for(int j =0;j<n;j++)
          if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Time: $\mathcal{O}(|V| + |E|)$

3966d8, 71 lines

```
const int maxn = 100007;
vector<int> graph[maxn];
set<int> graphL[maxn];
vector<int> inDegree(maxn,0);
void add_edge(int u,int v){
  inDegree[v]++;
  graph[u].push_back(v);
}
int n;
vector<int> topoSort(){
  vector<int> ans;
  priority_queue<int,vector<int>,greater<int>> q; // priority queue if you need a small lexicografic order
  // queue<int> q;
  for(int i = 0;i<n;i++)
    if(inDegree[i] == 0)
      q.push(i);
  while(!q.empty()){
    int u = q.top();
    // int u = q.front(); For a normal queue
    q.pop();
    ans.push_back(u);
    for(auto v:graph[u]){
      inDegree[v]--;
      if(inDegree[v] == 0){
        q.push(v);
      }
    }
  }
  return ans;
}
// Function to get all topolical sorts
int ALLTPS(stack<int>& s,int *recStack,vector<int>& res,int& c){
    int flag = 0;
  for(int i = 0;i<NODOS; i++){
    if(vis[i] == -1&& indegree[i] == 0){
      for(int u:grafo[i]){
        indegree[u]--;
      }
    vis[i] = 1;
    recStack[i] = 1;
    res.push_back(i);
    if(ALLTPS(s,recStack,res,c)==1)
      return 1;
    if(c ==1)
      return 2;
    vis[i] = 0;
    res.erase(res.end()-1);
      for(int u:grafo[i]){
        indegree[u]++;
      }
    flag =1 ;
    }
  }
  if(flag == 0){
    if(res.size() <NODOS)
      return 1;
    for (int i = 0; i < res.size(); i++)
    cout << res[i]+1 << " ";
    c++;
  }
    return 0;
}
int AlltopoSort(vector<int> graph[], int N){
    stack<int> s;
  int recS[N];
  vector<int> ATP;
  int c = 0;
    if(ALLTPS(s,recS,ATP,c) == 2 )
      return 1;
    return 0;
}
```

Dijkstra.cpp

Description: Calculates shortest paths from s in a graph
Time: $\mathcal{O}(V \log E)$

850899, 39 lines

```
const int INF = 1e9;
const int MAX = 1440007;
int D[MAX];
int P[MAX];
int N;
vector<pair<int,int>> graph[MAX];
void add_edge(int u,int v,int cost){
    graph[u].push_back( { v,cost } );
    graph[v].push_back( { u,cost } );
}
vector<int> restore_path(int s, int t, vector<int> const& p) {
    vector<int> path;
    for (int v = t; v != s; v = p[v])
        path.push_back(v);
    path.push_back(s);
    reverse(path.begin(), path.end());
    return path;
}
void dijkstra(int n,int Source){
    set<pair<int,int> > s;
    for(int i = 0; i < n; ++i)
        D[i] = INF;
    D[Source] = 0;
    s.insert(make_pair(D[0], Source));
    while (!s.empty()) {
        int v = s.begin()->second;
        s.erase(s.begin());
        for(auto c:e[v]){
            int u = c.first;
            int w = c.second;
            if (D[v] + w < D[u]) {
                s.erase(make_pair(D[u], u));
                D[u] = D[v]+ w;
                p[u] = v;
                s.insert(make_pair(D[u], u));
            }
        }
    }
}
```

kShortestPaths.cpp

Description: Calculates shortest paths from s in a graph
Usage: adj.resize(n + 1);rev.resize(n + 1); adj[u].pb(new Edge(v, w));
rev[v].pb(new Edge(u, w));
adj[u].back()->rev = rev[v].back();
rev[v].back()->rev = adj[u].back();
vector<int> res = k_shortest_paths(1, n, k); 1 indexed
Time: ??

db636c, 74 lines

```
const int inf = 1e18;
struct Edge {
  int to, w;
  Edge *rev;
  Edge (int to, int w) : to(to), w(w) { }
};
pair<vector<int>, vector<Edge*>> dijkstra (vector<vector<Edge*>> gra, int s) {
  vector<int> dis(gra.size(), inf);
  vector<Edge*> par(gra.size(), nullptr);
  priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
  pq.emplace(0, s);
  dis[s] = 0;
  while (pq.size()) {
    auto [d, u] = pq.top();
    pq.pop();
```

```
    if (dis[u] < d) continue;
    for (auto *e : gra[u]) {
      ll w = d + e->w;
      if (w < dis[e->to]) {
        par[e->to] = e->rev;
        pq.emplace(dis[e->to] = w, e->to);
      }
    }
  }
  return {dis, par};
}
vector<vector<Edge*>> adj, rev;
vector<int> k_shortest_paths (int s, int t, int k) {
  auto [dis, par] = dijkstra(rev, t);
  vector<int> res;
  priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<
      ↪ int,int>>> pq;
  pq.emplace(dis[s], s);
  while (k && pq.size()) {
    auto [d, u] = pq.top();
    pq.pop();
    res.push_back(d);
    k--;
    while (1) {
      for (Edge *e : adj[u]) {
        int v = e->to;
        if (e != par[u]) {
          ll w = d - dis[u] + e->w + dis[v];
          pq.emplace(w, v);
        }
      }
      if (!par[u])
        break;
      u = par[u]->to;
    }
  }
  while (k) {
    res.push_back(-1);
    k--;
  }
  return res;
}
void main_() {
  int n, m, k;
  cin >> n >> m >> k;
  adj.resize(n + 1);
  rev.resize(n + 1);
  for(int i = 0;i<m;i++){
    int u, v, w;
    cin >> u >> v >> w;
    adj[u].pb(new Edge(v, w));
    rev[v].pb(new Edge(u, w));
    adj[u].back()->rev = rev[v].back();
    rev[v].back()->rev = adj[u].back();
  }
  vector<int> res = k_shortest_paths(1, n, k);
  for (ll r : res)
    cout << r << " ";
  cout << endl;
}
```

7.2 Network flow

PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$

0ae1d4, 45 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }
  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);
    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0) // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlow.h

Description: Min-cost max-flow. cap[i][j] != cap[j][i] is allowed; double edges are not. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.

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

6890f1, 69 lines

```
struct Edge{
    int from, to, capacity, cost;
};
vector<vector<int>> adj, cost, capacity;
const int INF = 1e9;
void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}
int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
    N+=7;
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
```

```
    adj[e.to].push_back(e.from);
    cost[e.from][e.to] = e.cost;
    cost[e.to][e.from] = -e.cost;
    capacity[e.from][e.to] = e.capacity;
  }
  int flow = 0;
  int cost = 0;
  vector<int> d, p;
  while (flow < K) {
    shortest_paths(N, s, d, p);
    if (d[t] == INF)
      break;
    // find max flow on that path
    int f = K - flow;
    int cur = t;
    while (cur != s) {
      f = min(f, capacity[p[cur]][cur]);
      cur = p[cur];
    }
    // apply flow
    flow += f;
    cost += f * d[t];
    cur = t;
    while (cur != s) {
      capacity[p[cur]][cur] -= f;
      capacity[cur][p[cur]] += f;
      cur = p[cur];
    }
  }
  if (flow < K)
    return -1;
  else
    return cost;
}
```

EdmondsKarp.h

Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.

Usage: `graph.clear();graph.resize(n);capacity.resize(n,vector<int> (n,0));`
`capacity[u][v] = x;`
`graph[u].push_back(v);`

d20920, 71 lines

```
vector<vector<int>> capacity;
vector<vector<int>> graph;
const int INF = 1e9;
int bfs(int s, int t, vector<int>& parent) {
  fill(parent.begin(), parent.end(), -1);
  parent[s] = -2;
  queue<pair<int, int>> q;
  q.push({s, INF});
  while (!q.empty()) {
    int cur = q.front().first;
    int flow = q.front().second;
    q.pop();
    for (int next : graph[cur]) {
      if (parent[next] == -1 && capacity[cur][next]) {
        parent[next] = cur;
        int new_flow = min(flow, capacity[cur][next]);
        if (next == t)
          return new_flow;
        q.push({next, new_flow});
      }
    }
  }
  return 0;
}
int maxflow(int s, int t,int n) {
  int flow = 0;
  vector<int> parent(n);
  int new_flow;
  while (new_flow = bfs(s, t, parent)) {
    flow += new_flow;
    int cur = t;
    while (cur != s) {
      int prev = parent[cur];
      capacity[prev][cur] -= new_flow;
      capacity[cur][prev] += new_flow;
      cur = prev;
    }
  }
  return flow;
}
struct edge{
  int u,v,c;
};
void get_min_cut(int S,int n){
  vector<bool> cut(n);
  int idx = 0;
  cut[S] = 1;
  queue<int> q;
  q.push(S);
  while(!q.empty()){
    int u = q.front();
    q.pop();
    for(auto v:graph[u]){
      if(capacity[u][v])
        q.push(v),cut[v] = 1;
    }
  }
  for(auto c:cut)cout<<c<<" ";
  cout<<endl;
  vector<edge> edges;
  for(int i = 0;i<n;i++){
    if(cut[i]){
      for(auto v:graph[i]){
        if(!cut[v])
          edges.push_back({i,v,capacity[v][i]});
      }
    }
  }
  for(auto c:edges)
    cout<<c.u<<" "<<c.v<<" "<<c.c<<endl;
}
```

Dinic.h

Description: Flow algorithm with complexity $O(VE\log U)$ where $U = \max|\text{cap}|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.

c3e9a0, 56 lines

```
template<typename flow_type>
struct dinic{
 struct edge{
  size_t src, dst, rev;
  flow_type flow, cap;
 };
 int n;
 vector<vector<edge>> adj;
 dinic(int n) : n(n), adj(n), level(n), q(n), it(n) { }
 void add_edge(size_t src, size_t dst, flow_type cap, flow_type rcap
     ↪ = 0){
  adj[src].push_back({src, dst, adj[dst].size(), 0, cap});
  if (src == dst) adj[src].back().rev++;
  adj[dst].push_back({dst, src, adj[src].size() - 1, 0, rcap});
 }
 vector<int> level, q, it;
 bool bfs(int source, int sink){
  fill(level.begin(), level.end(), -1);
  for (int qf = level[q[0] = sink] = 0, qb = 1; qf < qb; ++qf){
   sink = q[qf];
   for (edge &e : adj[sink]){
    edge &r = adj[e.dst][e.rev];
    if (r.flow < r.cap && level[e.dst] == -1)
     level[q[qb++] = e.dst] = 1 + level[sink];
   }
  }
  return level[source] != -1;
 }
 flow_type augment(int source, int sink, flow_type flow){
  if (source == sink) return flow;
  for (; it[source] != adj[source].size(); ++it[source]){
   edge &e = adj[source][it[source]];
   if (e.flow < e.cap && level[e.dst] + 1 == level[source]){
    flow_type delta = augment(e.dst, sink,
         min(flow, e.cap - e.flow));
    if (delta > 0){
     e.flow += delta;
```

```
      adj[e.dst][e.rev].flow -= delta;
      return delta;
     }
    }
   }
   return 0;
  }
  flow_type max_flow(int source, int sink) {
   for (int u = 0; u < n; ++u)
    for (edge &e : adj[u]) e.flow = 0;
   flow_type flow = 0;
   flow_type oo = numeric_limits<flow_type>::max();
   while (bfs(source, sink)) {
    fill(it.begin(), it.end(), 0);
    for (flow_type f; (f = augment(source, sink, oo)) > 0;)
     flow += f;
   }
   return flow;
  }
};
```

MinCut.h

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity. Check EdmonsKarp for an implementation.

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}(V^3)$

f206fe, 21 lines

```
pair<int, vector<int>> globalMinCut(vector<vector<int>> mat) {
 pair<int, vector<int>> best = {INT_MAX, {}};
 int n = sz(mat);
 vector<vector<int>> co(n);
 for(int i = 0;i<n;i++) co[i] = {i};
 for(int ph = 1;ph<n;ph++) {
  vi w = mat[0];
  size_t s = 0, t = 0;
  for(int it = 0;it<n-ph;it++){ // O(V^2) -> O(E log V) with prio. queue
   w[t] = INT_MIN;
   s = t, t = max_element(all(w)) - w.begin();
   for(int i = 0;i<n;i++) w[i] += mat[t][i];
  }
  best = min(best, {w[t] - mat[t][t], co[t]});
  co[s].insert(co[s].end(), co[t].begin(),co[t].end());
  for(int i = 0;i<n;i++)mat[s][i] += mat[t][i];
  for(int i = 0;i<n;i++)mat[i][s] = mat[s][i];
  mat[0][t] = INT_MIN;
 }
 return best;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h" 0418b3, 13 lines

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
 vector<Edge> tree;
 vi par(N);
 rep(i,1,N) {
  PushRelabel D(N); // Dinic also works
  for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
  tree.push_back({i, par[i], D.calc(i, par[i])});
  rep(j,i+1,N)
   if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
 }
 return tree;
}
```

7.3 Matching

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

9d4e7d, 52 lines

```
struct Bipartite_Matching {
  vector<vector<int>> es;
  vector<int> d, match;
  vector<bool> used, used2;
  const int n, m;
  Bipartite_Matching(int n, int m) : es(n), d(n), match(m), used(n), used2(n), n(n), m(m) {}
  void add_edge(int u, int v) {
    es[u].push_back(v);
  }
  void bfs() {
    fill(begin(d), end(d), -1);
    queue<int> que;
    for(int i = 0; i < n; i++) {
      if(!used[i]) que.emplace(i), d[i] = 0;
    }
    while(!que.empty()) {
      int i = que.front(); que.pop();
      for(auto &e: es[i]) {
        int j = match[e];
        if(j != -1 && d[j] == -1) {
          que.emplace(j), d[j] = d[i]+1;
        }
      }
    }
  }
  bool dfs(int now) {
    used2[now] = true;
    for(auto &e: es[now]) {
      int u = match[e];
      if(u == -1 || (!used2[u] && d[u] == d[now]+1 && dfs(u))) {
        match[e] = now, used[now] = true;
        return true;
      }
    }
    return false;
  }
  int bipartite_matching() {
    fill(begin(match), end(match), -1), fill(begin(used), end(used), false);
    int ret = 0;
    while(true) {
      bfs();
      fill(begin(used2), end(used2), false);
      int flow = 0;
      for(int i = 0; i < n; i++) {
        if(!used[i] && dfs(i)) flow++;
      }
      if(flow == 0) break;
      ret += flow;
    }
    return ret;
  }
};
```

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); dfsMatching(g, btoa);`
Time: $\mathcal{O}(VE)$

522b98, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
 if (btoa[j] == -1) return 1;
 vis[j] = 1; int di = btoa[j];
 for (int e : g[di])
  if (!vis[e] && find(e, g, btoa, vis)) {
   btoa[e] = di;
   return 1;
  }
 return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
 vi vis;
 rep(i,0,sz(g)) {
```

```
  vis.assign(sz(btoa), 0);
  for (int j : g[i])
   if (find(j, g, btoa, vis)) {
    btoa[j] = i;
    break;
   }
 }
 return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set. A vertex cover is a set of vertex such that every edge has an endpoint to one of the vertex in the set

bdeffe, 21 lines

```
#include "DFSMatching.h"
vi cover(vector<vi>& g, int n, int m) {
 vi match(m, -1);
 int res = dfsMatching(g, match);
 vector<bool> lfound(n, true), seen(m);
 for (int it : match) if (it != -1) lfound[it] = false;
 vi q, cover;
 rep(i,0,n) if (lfound[i]) q.push_back(i);
 while (!q.empty()) {
  int i = q.back(); q.pop_back();
  lfound[i] = 1;
  for (int e : g[i]) if (!seen[e] && match[e] != -1) {
   seen[e] = true;
   q.push_back(match[e]);
  }
 }
 rep(i,0,n) if (!lfound[i]) cover.push_back(i);
 rep(i,0,m) if (seen[i]) cover.push_back(n+i);
 assert(sz(cover) == res);
 return cover;
}
```

WeightedMatching.h

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.
Time: $\mathcal{O}(N^2M)$

c97b86, 31 lines

```
pair<int, vector<int>> hungarian(const vector<vector<int>> &a) {
 if (a.empty()) return {0, {}};
 int n = sz(a) + 1, m = sz(a[0]) + 1;
 vector<int> u(n), v(m), p(m), ans(n - 1);
 for(int i = 1;i<n;i++){
  p[0] = i;
  int j0 = 0; // add "dummy" worker 0
  vector<int> dist(m, INT_MAX), pre(m, -1);
  vector<bool> done(m + 1);
  do { // dijkstra
   done[j0] = true;
   int i0 = p[j0], j1, delta = INT_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
}
```

GeneralMatching.h

Description: Matching for general graphs.
Time: $\mathcal{O}(NM)$

e91247, 57 lines

```
const int maxn = 507;
vector<int> graph[maxn];
vector<int> Blossom(int n) {
  int timer = -1;
  vector<int> mate(n, -1), label(n), parent(n), orig(n), aux(n, -1),
      ↪ q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv;
      mate[pv] = v;
      v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1;
          parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        }
        else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a);
          blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

7.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: scc(graph, [&](vi& v) { ... }) visits all components in reverse topological order. comp[i] holds the component index of a node (a component only has edges to components with lower index). ncomps will contain the number of components.
Time: $\mathcal{O}(E+V)$

76b5c9, 23 lines

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
 int low = val[j] = ++Time, x; z.push_back(j);
 for (auto e : g[j]) if (comp[e] < 0)
  low = min(low, val[e] ?: dfs(e,g,f));
 if (low == val[j]) {
  do {
```

```
      x = z.back(); z.pop_back();
      comp[x] = ncomps;
      cont.push_back(x);
    } while (x != j);
   f(cont); cont.clear();
   ncomps++;
  }
 return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
 int n = sz(g);
 val.assign(n, 0); comp.assign(n, -1);
 Time = ncomps = 0;
 rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

BridgeTree(BiconnectedComponents).h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.

ca0b4e, 62 lines

```
const int maxn = 200007;
set<int> graph[maxn];
set<int> graph2[maxn];
vector<int> low(maxn),d(maxn),label(maxn),bridge(maxn),vis(maxn),parent(maxn);
lli idx;
void dfs(lli u,lli p = -1){
   d[u] =idx++;
   low[u] = d[u];
   vis[u] = true;
   parent[u] = p;
   for(auto v:graph[u]){
      if(v == p)continue;
      if(!vis[v]){
         dfs(v,u);
         if(low[v]>d[u]) bridge[v] = true;
      }
      low[u] = min(low[u],low[v]);
   }
}
void dfs_label(lli u){
   vis[u] = 1;
   label[u] = idx;
   for(auto v : graph[u])
      if(!vis[v])
         dfs_label(v);
}
int main(){__
 int t= 1,n,u,v;
   cin>>t;
   while(t--){
      cin>>n;
      for(lli i = 0;i<n;i++)graph[i].clear(),graph2[i].clear(),bridge[i] = false,vis[i] = false;
      for(lli i = 0;i<n;i++){
         cin>>u>>v;
         u--,v--;
         graph[u].insert(v);
         graph[v].insert(u);
         graph2[u].insert(v);
         graph2[v].insert(u);
      }
      dfs(0);
      lli root;
      set<lli> cycle;
      vector<pair<lli,lli>> bridges;
      for(lli i = 0;i<n;i++){
         vis[i] = false;
         if(bridge[i]){
            graph[i].erase(parent[i]);
            graph[parent[i]].erase(i);
            if(i)bridges.push_back( { i,parent[i] } );
         }
      }
      idx = 0;
      for(lli i = 0;i<n;i++){
         if(!vis[i]){
            dfs_label(i);
            idx++;
         }
      }
   }
   return 0;
}
```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, Here is an example of such a 2-SAT problem. Find an assignment of a,b,c such that the following formula is true: so that an expression of the type $(a \lor \neg b) \land (\neg a \lor b) \land (\neg a \lor \neg b) \land \ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).

d15d85, 47 lines

```
const int maxn = 2000007;
vector<int> g[maxn];
vector<int> gt[maxn];
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;
void dfs1(int v) {
   used[v] = true;
   for (int u : g[v]) {
      if (!used[u])
         dfs1(u);
   }
   order.push_back(v);
}
void dfs2(int v, int cl) {
   comp[v] = cl;
   for (int u : gt[v]) {
      if (comp[u] == -1)
         dfs2(u, cl);
   }
}
bool solve_2SAT(int n) {
   used.assign(n, false);
   for (int i = 0; i < n; ++i) {
      if (!used[i])
         dfs1(i);
   }
   comp.assign(n, -1);
   for (int i = 0, j = 0; i < n; ++i) {
      int v = order[n - i - 1];
      if (comp[v] == -1)
         dfs2(v, j++);
   }
   assignment.assign(n / 2, false);
   for (int i = 0; i < n; i += 2) {
      if (comp[i] == comp[i + 1])
         return false;
      assignment[i / 2] = comp[i] > comp[i + 1];
   }
   return true;
}
void add_edge(int u,int v,bool negU,bool negV){
   g[(u*2)+(negU?0:1)].push_back((v*2)+(negV?1:0));
   gt[(v*2)+(negV?1:0)].push_back((u*2)+(negU?0:1));
   g[(v*2)+(negV?0:1)].push_back((u*2)+(negU?1:0));
   gt[(u*2)+(negU?1:0)].push_back((v*2)+(negV?0:1));
}
```

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time: $\mathcal{O}(V+E)$

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
 int n = sz(gr);
 vi D(n), its(n), eu(nedges), ret, s = { src } ;
 D[src]++; // to allow Euler paths, not just cycles
 while (!s.empty()) {
  int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
  if (it == end){ ret.push_back(x); s.pop_back(); continue; }
```

```
  tie(y, e) = gr[x][it++];
  if (!eu[e]) {
   D[x]--, D[y]++;
   eu[e] = 1; s.push_back(y);
  }}
 for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
 return {ret.rbegin(), ret.rend()};
}
```

DSUTree.cpp

Description: A trick to get information about subtree , like how many nodes in my subtree has color c

78408d, 43 lines

```
void dfs_size(int v, int p) {
 sz[v] = 1;
 for (auto u : adj[v]) {
  if (u != p) {
   dfs_size(u, v);
   sz[v] += sz[u];
  }
 }
}
void dfs(int v, int p, bool keep) {
 int Max = -1, bigchild = -1;
 for (auto u : adj[v]) {
  if (u != p && Max < sz[u]) {
   Max = sz[u];
   bigchild = u;
  }
 }
 for (auto u : adj[v]) {
  if (u != p && u != bigchild) {
   dfs(u, v, 0);
  }
 }
 if (bigchild != -1) {
  dfs(bigchild, v, 1);
  swap(vec[v], vec[bigchild]);
 }
 vec[v].push_back(v);
 cnt[color[v]]++;
 for (auto u : adj[v]) {
  if (u != p && u != bigchild) {
   for (auto x : vec[u]) {
    cnt[color[x]]++;
    vec[v].push_back(x);
   }
  }
 }
 // there are cnt[c] vertex in subtree v color with c
 if (keep == 0) {
  for (auto u : vec[v]) {
   cnt[color[u]]--;
  }
 }
}
```

blockCutTree.cpp

Description: Decompose the tree aroun articilation points

e3fda9, 79 lines

```
const int N = 4e5 + 9;
int T, low[N], dis[N], art[N], sz;
vector<int> g[N], bcc[N], st;
void dfs(int u, int pre = 0) {
 low[u] = dis[u] = ++T;
 st.push_back(u);
 for(auto v: g[u]) {
  if(!dis[v]) {
   dfs(v, u);
   low[u] = min(low[u], low[v]);
   if(low[v] >= dis[u]) {
    sz ++;
    int x;
    do {
     x = st.back();
     st.pop_back();
     bcc[x].push_back(sz);
    } while(x ^v);
    bcc[u].push_back(sz);
   }
  } else if(v != pre) low[u] = min(low[u], dis[v]);
 }
}
int dep[N], par[N][20], cnt[N], id[N];
vector<int> bt[N];
void dfs1(int u, int pre = 0) {
 dep[u] = dep[pre] + 1;
 cnt[u] = cnt[pre] + art[u];
 par[u][0] = pre;
 for(int k = 1; k <= 18; k++) par[u][k] = par[par[u][k - 1]][k - 1];
 for(auto v: bt[u]) if(v != pre) dfs1(v, u);
}
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 dist(int u, int v) {
 int lc = lca(u, v);
 return cnt[u] + cnt[v] - 2 * cnt[lc] + art[lc];
}
int32_t main() {
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 int n, m;
 cin >> n >> m;
 while(m--) {
  int u, v;
  cin >> u >> v;
  g[u].push_back(v);
  g[v].push_back(u);
 }
 dfs(1);
 for(int u = 1; u <= n; u++) {
  if(bcc[u].size() > 1) { //AP
   id[u] = ++sz;
   art[id[u]] = 1; //if id in BCT is an AP on real graph or not
   for(auto v: bcc[u]) {
    bt[id[u]].push_back(v);
    bt[v].push_back(id[u]);
   }
  } else if(bcc[u].size() == 1) id[u] = bcc[u][0];
 }
 dfs1(1);
 int q;
 cin >> q;
 while(q--) {
  int u, v;
  cin >> u >> v;
  int ans;
  if(u == v) ans = 0;
  else ans = dist(id[u], id[v]) - art[id[u]] - art[id[v]];
  cout << ans << '\n';//number of articulation points in the path
      from u to v except u and v
  //u and v are in the same bcc if ans == 0
 }
 return 0;
}
```

7.5 Coloring

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D, computes a $(D+1)$-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

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

ec3455, 31 lines

```
vector<int> edgeColoring(int N, vector<pair<int,int>> edges) {
 vector<int> cc(N + 1), ret(sz(edges)), fan(N), free(N), loc;
 for (pii e : edges) ++cc[e.first], ++cc[e.second];
 int u, v, ncols = *max_element(all(cc)) + 1;
 vector<vector<int>> adj(N, vector<int>(ncols, -1));
 for (pii e : edges) {
```

```
  tie(u, v) = e;
  fan[0] = v;
  loc.assign(ncols, 0);
  int at = u, end = u, d, c = free[u], ind = 0, i = 0;
  while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
   loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
  cc[loc[d]] = c;
  for (int cd = d; at != -1; cd ^= c ^d, at = adj[at][cd])
   swap(adj[at][cd], adj[end = at][cd ^c ^d]);
  while (adj[fan[i]][d] != -1) {
   int left = fan[i], right = fan[++i], e = cc[i];
   adj[u][e] = left;
   adj[left][e] = u;
   adj[right][e] = -1;
   free[right] = e;
  }
  adj[u][d] = fan[i];
  adj[fan[i]][d] = u;
  for (int y : {fan[0], u, end})
   for (int& z = free[y] = 0; adj[y][z] != -1; z++);
 }
 for(int i = 0;i<edges.size();i++)
  for (tie(u, v) = edges[i]; adj[u][ret[i]] != v;) ++ret[i];
 return ret;
}
```

7.6 Trees

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.

Time: $\mathcal{O}(N \log N + Q)$

91b6d0, 47 lines

```
const int maxn = 100007;
const int mxlog = 25;
vector<int> graph[maxn];
int parent[mxlog][maxn];
vector<int> deep(maxn);
int N;
void add_edge(int u,int v){
  graph[u].push_back(v);
  graph[v].push_back(u);
}
void dfs(int u,int p = -1,int d = 0){
  deep[u] = d;
  parent[0][u] = p;
  for(auto v:graph[u]){
    if(v== p)continue;
    dfs(v,u,d+1);
  }
}
void build(){
  for(int i = 0;i<N;i++)for(int j = 0;j<mxlog;j++)parent[j][i] = -1;
  for(int i = 0;i<N;i++)deep[i]= -1;
  dfs(0);
  for(int i = 0;i<N;i++)
    if(deep[i]== -1)dfs(i);
  for(int i = 1;i<mxlog;i++){
    for(int u = 0;u<N;u++){
      if(parent[i-1 ][u]!= -1)
      parent[i][u] = parent[i-1][parent[i-1][u]];
    }
  }
}
int lca(int u ,int v){
  if(deep[u]>deep[v])swap(u,v);
  int diff = deep[v]-deep[u];
  for(int i = mxlog-1;i>=0;i--){
    if(diff & (1<<i))
      v = parent[i][v];
  }
  if(u == v)return u;
  for(int i = mxlog-1;i>=0;i--){
    if(parent[i][u]!= parent[i][v]){
      u = parent[i][u];
      v = parent[i][v];
    }
  }
  return parent[0][u];
}
```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges.

Time: $\mathcal{O}\left((\log N)^2\right)$ per query in a path

"SegmentTree.cpp" d18d4e, 61 lines

```
const int maxn = 100007;
vector<pair<int,int>> graph[maxn];
void add_edge(int u,int v,int c){
  graph[u].push_back( { v,c } );
  graph[v].push_back( { u,c } );
}
vector<int> p(maxn),head(maxn),stpos(maxn),lvl(maxn),sz(maxn),val(
    maxn);
vector<int> heavy(maxn,-1);
int cn = 0;
void dfs(int u ,int pr = -1,int lev = 0){
  lvl[u] = lev;
  sz[u]= 1;
  int mx= 0;
  p[u] = pr;
  for(auto v:graph[u]){
    if(v.x == pr)continue;
      val[v.x] = v.y;
      dfs(v.x,u,lev+1);
      if(sz[v.x]>mx){
        mx = sz[v.x];
        heavy[u] = v.x;
      }
      sz[u]+=sz[v.x];
  }
}
void HLD(int u,int ch,int n){
  head[u] = ch;
  stpos[u] = cn++;
  for(int i=0, currpos = 0; i < n; ++i)
  if(p[i] == -1 || heavy[p[i]] != i)
   for(int j = i; j != -1; j = heavy[j])
    {
      head[j] = i;
      stpos[j] = currpos;
      currpos++;
         }
}
int query(int a,int b,int n){
  int res = 0;
  while(head[a]!= head[b]){
    if(lvl[head[a]]< lvl[head[b]])
      swap(a,b);
    res += query(1,0,n-1,stpos[head[a]],stpos[a]);
    a = p[head[a]];
  }
  if(lvl[a]> lvl[b])
    swap(a,b);
  res+=query(1,0,n-1,stpos[a],stpos[b]);
  return res;
}
int update(int a,int b,int val, int n){
  while(head[a] != head[b]){
    if(lvl[head[a]] < lvl[head[b]])
      swap(a,b);
    update(1,0,n-1,stpos[head[a]],stpos[a],val);
    a = p[head[a]];
  }
  if(lvl[a]> lvl[b])
    swap(a,b);
  update(1,0,n-1,stpos[a],stpos[b],val);
}
```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.

Time: All operations take amortized $\mathcal{O}(\log N)$.

5909e2, 88 lines

```cpp
struct Node { // Splay tree. Root's pp contains tree's parent.
 Node *p = 0, *pp = 0, *c[2];
 bool flip = 0;
 Node() { c[0] = c[1] = 0; fix(); }
 void fix() {
  if (c[0]) c[0]->p = this;
  if (c[1]) c[1]->p = this;
  // (+ update sum of subtree elements etc. if wanted)
 }
 void pushFlip() {
  if (!flip) return;
  flip = 0; swap(c[0], c[1]);
  if (c[0]) c[0]->flip ^= 1;
  if (c[1]) c[1]->flip ^= 1;
 }
 int up() { return p ? p->c[1] == this : -1; }
 void rot(int i, int b) {
  int h = i ^b;
  Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
  if ((y->p = p)) p->c[up()] = y;
  c[i] = z->c[i ^1];
  if (b < 2) {
   x->c[h] = y->c[h ^1];
   z->c[h ^1] = b ? x : this;
  }
  y->c[i ^1] = b ? this : x;
  fix(); x->fix(); y->fix();
  if (p) p->fix();
  swap(pp, y->pp);
 }
 void splay() {
  for (pushFlip(); p; ) {
   if (p->p) p->p->pushFlip();
   p->pushFlip(); pushFlip();
   int c1 = up(), c2 = p->up();
   if (c2 == -1) p->rot(c1, 2);
   else p->p->rot(c2, c1 != c2);
  }
 }
 Node* first() {
  pushFlip();
  return c[0] ? c[0]->first() : (splay(), this);
 }
};
struct LinkCut {
 vector<Node> node;
 LinkCut(int N) : node(N) { }
 void link(int u, int v) { // add an edge (u, v)
  assert(!connected(u, v));
  makeRoot(&node[u]);
  node[u].pp = &node[v];
 }
 void cut(int u, int v) { // remove an edge (u, v)
  Node *x = &node[u], *top = &node[v];
  makeRoot(top); x->splay();
  assert(top == (x->pp ?: x->c[0]));
  if (x->pp) x->pp = 0;
  else {
   x->c[0] = top->p = 0;
   x->fix();
  }
 }
 bool connected(int u, int v) { // are u, v in the same tree?
  Node* nu = access(&node[u])->first();
  return nu == access(&node[v])->first();
 }
 void makeRoot(Node* u) {
  access(u);
  u->splay();
  if(u->c[0]) {
   u->c[0]->p = 0;
   u->c[0]->flip ^= 1;
   u->c[0]->pp = u;
   u->c[0] = 0;
   u->fix();
  }
 }
 Node* access(Node* u) {
  u->splay();
  while (Node* pp = u->pp) {
   pp->splay(); u->pp = 0;
   if (pp->c[1]) {
    pp->c[1]->p = 0; pp->c[1]->pp = pp; }
   pp->c[1] = u; pp->fix(); u = pp;
  }
  return u;
 }
};
```

CentroidDecomposition.cpp

Description: Decomposes a tree into centroids and create a new tree with the centroids

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

b5bb88, 65 lines

```cpp
const int maxn = 100005;
vector<int> graph[maxn];
vector<int> parent(maxn,-1);
vector<int> depth(maxn,-1);
vector<int> best(maxn,1e16);
int P[maxn][25];
bitset<maxn> cent;
int sz[maxn];
void dfs (int u, int p = -1,int d = 0) {
    sz[u] = 1;
    P[u][0] = p;
    depth[u] = d;
    for (int v : graph[u]) {
        if(v ==p)continue;
        dfs(v,u,d+1);
        sz[u] += sz[v];
    }
}
void build(int n) {
    for(int i = 0;i<n;i++)
        for(int j = 0;j<25;j++)
            P[i][j] = -1;
    dfs(0);
    for(int i = 1;i<25;i++)
        for(int u = 0;u<n;u++)
            if(P[u][i-1]!=-1)
                P[u][i] = P[P[u][i-1]][i-1];
}
int lca(int u,int v) {
    if(depth[u]<depth[v])swap(u,v);
    int diff = depth[u]-depth[v];
    for(int i = 24;i>=0;i--) {
        if((diff>>i)&1) {
            u = P[u][i];
        }
    }
    if(u==v)return u;
    for(int i= 24;i>=0;i--) {
        if(P[u][i]!=P[v][i]) {
            u = P[u][i];
            v = P[v][i];
        }
    }
    return P[u][0];
}
int descomp (int u) {
    int tam = 1;
    for (int v : graph[u])
        if (!cent[v])
            tam += sz[v];
    while (1) {
        int idx = -1;
        for (int v : graph[u])
            if (!cent[v] && 2 * sz[v] > tam)
                idx = v;
        if (idx == -1)break;<
        sz[u] = tam - sz[idx];
        u = idx;
    }
    cent[u] = 1;
```

```cpp
  for (int v : graph[u])
    if (!cent[v])
      parent[descomp(v)] = u;
  return u;
}
```

7.7 Math

7.7.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--`, `mat[b][b]++` (and `mat[b][a]--`, `mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

7.7.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k).$$

Kirchhoff Matrix Tree Theorem

Count the number of spanning trees in a graph, as the determinant of the Laplacian matrix of the graph.

Laplacian Matrix :
Given a simple graph G with n vertices, its Laplacian matrix $L_{n\times n}$ is defined as

$$L = D - A$$

The elements of L are given by

$$L_{i,j} = \begin{cases} deg(v_i) & \text{if } i == j \\ -1 & \text{if } i \neq j \text{and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

define $\tau(G)$ as number of spanning trees of a grap G

$$\tau(G) = \det L_{n-1\times n-1}$$

Where $L_{n-1\times n-1}$ is a laplacian matrix deleting any row and any column

$$\det \begin{pmatrix} deg(v_1) & L_{1,2} & \cdots & L_{1,n-1} \\ L_{2,1} & deg(v_2) & \cdots & L_{2,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ L_{n-1,1} & L_{n-1,2} & \cdots & deg(v_{n-1}) \end{pmatrix}$$

Generalization for a multigraph $K_n^m \pm G$

define $\tau(K_n^m \pm G)$ as number of spanning trees of a grap $K_n^m \pm G$

$$\tau(K_n^m \pm G) = n * (nm)^{n-p-2} \det(B)$$

where $B = mnI_p + \alpha * L(G)$is a $p \times p$ matrix, $\alpha = \pm$ according $(K_n^m \pm G)$, and $L(G)$ is the Kirchhoff matrix of G

Geometry (8)

8.1 all

AllGeometry.cpp 920 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
using ld = long double;
const ld eps = 1e-9, inf = numeric_limits<ld>::max(), pi = acos(-1);
// For use with integers, just set eps=0 and everything remains the same
bool geq(ld a, ld b){return a-b >= -eps;} //a >= b
bool leq(ld a, ld b){return b-a >= -eps;} //a <= b
bool ge(ld a, ld b){return a-b > eps;} //a > b
bool le(ld a, ld b){return b-a > eps;} //a < b
bool eq(ld a, ld b){return abs(a-b) <= eps;} //a == b
bool neq(ld a, ld b){return abs(a-b) > eps;} //a != b

struct point{
 ld x, y;
 point(): x(0), y(0){}
 point(ld x, ld y): x(x), y(y){}

 point operator+(const point & p) const{return point(x + p.x, y + p.y);}
 point operator-(const point & p) const{return point(x - p.x, y - p.y);}
 point operator*(const ld & k) const{return point(x * k, y * k);}
 point operator/(const ld & k) const{return point(x / k, y / k);}

 point operator+=(const point & p){*this = *this + p; return *this;}
 point operator-=(const point & p){*this = *this - p; return *this;}
 point operator*=(const ld & p){*this = *this * p; return *this;}
 point operator/=(const ld & p){*this = *this / p; return *this;}

 point rotate(const ld & a) const{return point(x*cos(a) - y*sin(a), x*sin(a) + y*cos(a));}
 point perp() const{return point(-y, x);}
 ld ang() const{
  ld a = atan2l(y, x); a += le(a, 0) ? 2*pi : 0; return a;
 }
 ld dot(const point & p) const{return x * p.x + y * p.y;}
 ld cross(const point & p) const{return x * p.y - y * p.x;}
 ld norm() const{return x * x + y * y;}
 ld length() const{return sqrtl(x * x + y * y);}
 point unit() const{return (*this) / length();}

 bool operator==(const point & p) const{return eq(x, p.x) && eq(y, p.y);}
 bool operator!=(const point & p) const{return !(*this == p);}
 bool operator<(const point & p) const{return le(x, p.x) || (eq(x, p.x) && le(y, p.y));}
 bool operator>(const point & p) const{return ge(x, p.x) || (eq(x, p.x) && ge(y, p.y));}
 bool half(const point & p) const{return le(p.cross(*this), 0) || (eq(p.cross(*this), 0) && le(p.dot(*this), 0));}
};

istream &operator>>(istream &is, point & p){return is >> p.x >> p.y;}
ostream &operator<<(ostream &os, const point & p){return os << "(" << p.x << ", " << p.y << ")";}

int sgn(ld x){
 if(ge(x, 0)) return 1;
 if(le(x, 0)) return -1;
 return 0;
}

void polarSort(vector<point> & P, const point & o, const point & v){
 //sort points in P around o, taking the direction of v as first angle
 sort(P.begin(), P.end(), [&](const point & a, const point & b){
  return point((a - o).half(v), 0) < point((b - o).half(v), (a - o).cross(b - o));
 });
}

bool pointInLine(const point & a, const point & v, const point & p){
 //line a+tv, point p
 return eq((p - a).cross(v), 0);
}

bool pointInSegment(const point & a, const point & b, const point & p){
 //segment ab, point p
 return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
}

int intersectLinesInfo(const point & a1, const point & v1, const point & a2, const point & v2){
 //lines a1+tv1 and a2+tv2
 ld det = v1.cross(v2);
```

```cpp
	if(eq(det, 0)){
		if(eq((a2 - a1).cross(v1), 0)){
			return -1; //infinity points
		}else{
			return 0; //no points
		}
	}else{
		return 1; //single point
	}
}

point intersectLines(const point & a1, const point & v1, const point & a2, const point & v2){
	//lines a1+tv1, a2+tv2
	//assuming that they intersect
	ld det = v1.cross(v2);
	return a1 + v1 * ((a2 - a1).cross(v2) / det);
}

int intersectLineSegmentInfo(const point & a, const point & v, const point & c, const point & d){
	//line a+tv, segment cd
	point v2 = d - c;
	ld det = v.cross(v2);
	if(eq(det, 0)){
		if(eq((c - a).cross(v), 0)){
			return -1; //infinity points
		}else{
			return 0; //no point
		}
	}else{
		return sgn(v.cross(c - a)) != sgn(v.cross(d - a)); //1: single point, 0: no point
	}
}

int intersectSegmentsInfo(const point & a, const point & b, const point & c, const point & d){
	//segment ab, segment cd
	point v1 = b - a, v2 = d - c;
	int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a));
	if(t == u){
		if(t == 0){
			if(pointInSegment(a, b, c) || pointInSegment(a, b, d) || pointInSegment(c, d, a) || pointInSegment(c, d, b)){
				return -1; //infinity points
			}else{
				return 0; //no point
			}
		}else{
			return 0; //no point
		}
	}else{
		return sgn(v2.cross(a - c)) != sgn(v2.cross(b - c)); //1: single point, 0: no point
	}
}

ld distancePointLine(const point & a, const point & v, const point & p){
	//line: a + tv, point p
	return abs(v.cross(p - a)) / v.length();
}

ld perimeter(vector<point> & P){
	int n = P.size();
	ld ans = 0;
	for(int i = 0; i < n; i++){
		ans += (P[i] - P[(i + 1) % n]).length();
	}
	return ans;
}

ld area(vector<point> & P){
	int n = P.size();
	ld ans = 0;
	for(int i = 0; i < n; i++){
		ans += P[i].cross(P[(i + 1) % n]);
	}
	return abs(ans / 2);
}

vector<point> convexHull(vector<point> P){
	sort(P.begin(), P.end());
	vector<point> L, U;
	for(int i = 0; i < P.size(); i++){
		while(L.size() >= 2 && leq((L[L.size() - 2] - P[i]).cross(L[L.size() - 1] - P[i]), 0)){
			L.pop_back();
		}
		L.push_back(P[i]);
	}
	for(int i = P.size() - 1; i >= 0; i--){
		while(U.size() >= 2 && leq((U[U.size() - 2] - P[i]).cross(U[U.size() - 1] - P[i]), 0)){
			U.pop_back();
		}
		U.push_back(P[i]);
	}
	L.pop_back();
	U.pop_back();
	L.insert(L.end(), U.begin(), U.end());
	return L;
}

bool pointInPerimeter(const vector<point> & P, const point & p){
	int n = P.size();
	for(int i = 0; i < n; i++){
		if(pointInSegment(P[i], P[(i + 1) % n], p)){
			return true;
		}
	}
	return false;
}

bool crossesRay(const point & a, const point & b, const point & p){
	return (geq(b.y, p.y) - geq(a.y, p.y)) * sgn((a - p).cross(b - p)) > 0;
}

int pointInPolygon(const vector<point> & P, const point & p){
	if(pointInPerimeter(P, p)){
		return -1; //point in the perimeter
	}
	int n = P.size();
	int rays = 0;
	for(int i = 0; i < n; i++){
		rays += crossesRay(P[i], P[(i + 1) % n], p);
	}
	return rays & 1; //0: point outside, 1: point inside
}

//point in convex polygon in O(log n)
//make sure that P is convex and in ccw
//before the queries, do the preprocess on P:
// rotate(P.begin(), min_element(P.begin(), P.end()), P.end());
// int right = max_element(P.begin(), P.end()) - P.begin();
//returns 0 if p is outside, 1 if p is inside, -1 if p is in the perimeter
int pointInConvexPolygon(const vector<point> & P, const point & p, int right){
	if(p < P[0] || P[right] < p) return 0;
	int orientation = sgn((P[right] - P[0]).cross(p - P[0]));
	if(orientation == 0){
		if(p == P[0] || p == P[right]) return -1;
		return (right == 1 || right + 1 == P.size()) ? -1 : 1;
	}else if(orientation < 0){
		auto r = lower_bound(P.begin() + 1, P.begin() + right, p);
		int det = sgn((p - r[-1]).cross(r[0] - r[-1])) - 1;
		if(det == -2) det = 1;
		return det;
	}else{
		auto l = upper_bound(P.rbegin(), P.rend() - right - 1, p);
```

```cpp
    int det = sgn((p - l[0]).cross((l == P.rbegin() ? P[0] : l[-1]) - l[0])) - 1;
    if(det == -2) det = 1;
    return det;
  }
}

vector<point> cutPolygon(const vector<point> & P, const point & a, const point & v){
  //returns the part of the convex polygon P on the left side of line a+tv
  int n = P.size();
  vector<point> lhs;
  for(int i = 0; i < n; ++i){
    if(geq(v.cross(P[i] - a), 0)){
      lhs.push_back(P[i]);
    }
    if(intersectLineSegmentInfo(a, v, P[i], P[(i+1)%n]) == 1){
      point p = intersectLines(a, v, P[i], P[(i+1)%n] - P[i]);
      if(p != P[i] && p != P[(i+1)%n]){
        lhs.push_back(p);
      }
    }
  }
  return lhs;
}

point centroid(vector<point> & P){
  point num;
  ld den = 0;
  int n = P.size();
  for(int i = 0; i < n; ++i){
    ld cross = P[i].cross(P[(i + 1) % n]);
    num += (P[i] + P[(i + 1) % n]) * cross;
    den += cross;
  }
  return num / (3 * den);
}

vector<pair<int, int>> antipodalPairs(vector<point> & P){
  vector<pair<int, int>> ans;
  int n = P.size(), k = 1;
  auto f = [&](int u, int v, int w){return abs((P[v%n]-P[u%n]).cross(P[w%n]-P[u%n]));};
  while(ge(f(n-1, 0, k+1), f(n-1, 0, k))) ++k;
  for(int i = 0, j = k; i <= k && j < n; ++i){
    ans.emplace_back(i, j);
    while(j < n-1 && ge(f(i, i+1, j+1), f(i, i+1, j)))
      ans.emplace_back(i, ++j);
  }
  return ans;
}

pair<ld, ld> diameterAndWidth(vector<point> & P){
  int n = P.size(), k = 0;
  auto dot = [&](int a, int b){return (P[(a+1)%n]-P[a]).dot(P[(b+1)%n]-P[b]);};
  auto cross = [&](int a, int b){return (P[(a+1)%n]-P[a]).cross(P[(b+1)%n]-P[b]);};
  ld diameter = 0;
  ld width = inf;
  while(ge(dot(0, k), 0)) k = (k+1) % n;
  for(int i = 0; i < n; ++i){
    while(ge(cross(i, k), 0)) k = (k+1) % n;
    //pair: (i, k)
    diameter = max(diameter, (P[k] - P[i]).length());
    width = min(width, distancePointLine(P[i], P[(i+1)%n] - P[i], P[k]));
  }
  return {diameter, width};
}

pair<ld, ld> smallestEnclosingRectangle(vector<point> & P){
  int n = P.size();
  auto dot = [&](int a, int b){return (P[(a+1)%n]-P[a]).dot(P[(b+1)%n]-P[b]);};
  auto cross = [&](int a, int b){return (P[(a+1)%n]-P[a]).cross(P[(b+1)%n]-P[b]);};
  ld perimeter = inf, area = inf;
  for(int i = 0, j = 0, k = 0, m = 0; i < n; ++i){
    while(ge(dot(i, j), 0)) j = (j+1) % n;
    if(!i) k = j;
    while(ge(cross(i, k), 0)) k = (k+1) % n;
    if(!i) m = k;
    while(le(dot(i, m), 0)) m = (m+1) % n;
    //pairs: (i, k) , (j, m)
    point v = P[(i+1)%n] - P[i];
    ld h = distancePointLine(P[i], v, P[k]);
    ld w = distancePointLine(P[j], v.perp(), P[m]);
    perimeter = min(perimeter, 2 * (h + w));
    area = min(area, h * w);
  }
  return {area, perimeter};
}

ld distancePointCircle(const point & c, ld r, const point & p){
  //point p, circle with center c and radius r
  return max((ld)0, (p - c).length() - r);
}

point projectionPointCircle(const point & c, ld r, const point & p){
  //point p (outside the circle), circle with center c and radius r
  return c + (p - c).unit() * r;
}

pair<point, point> pointsOfTangency(const point & c, ld r, const point & p){
  //point p (outside the circle), circle with center c and radius r
  point v = (p - c).unit() * r;
  ld d2 = (p - c).norm(), d = sqrt(d2);
  point v1 = v * (r / d), v2 = v.perp() * (sqrt(d2 - r*r) / d);
  return {c + v1 - v2, c + v1 + v2};
}

vector<point> intersectLineCircle(const point & a, const point & v, const point & c, ld r){
  //line a+tv, circle with center c and radius r
  ld h2 = r*r - v.cross(c - a) * v.cross(c - a) / v.norm();
  point p = a + v * v.dot(c - a) / v.norm();
  if(eq(h2, 0)) return {p}; //line tangent to circle
  else if(le(h2, 0)) return {}; //no intersection
  else{
    point u = v.unit() * sqrt(h2);
    return {p - u, p + u}; //two points of intersection (chord)
  }
}

vector<point> intersectSegmentCircle(const point & a, const point & b, const point & c, ld r){
  //segment ab, circle with center c and radius r
  vector<point> P = intersectLineCircle(a, b - a, c, r), ans;
  for(const point & p : P){
    if(pointInSegment(a, b, p)) ans.push_back(p);
  }
  return ans;
}

pair<point, ld> getCircle(const point & m, const point & n, const point & p){
  //find circle that passes through points p, q, r
  point c = intersectLines((n + m) / 2, (n - m).perp(), (p + n) / 2, (p - n).perp());
  ld r = (c - m).length();
  return {c, r};
}

vector<point> intersectionCircles(const point & c1, ld r1, const point & c2, ld r2){
  //circle 1 with center c1 and radius r1
  //circle 2 with center c2 and radius r2
  point d = c2 - c1;
  ld d2 = d.norm();
  if(eq(d2, 0)) return {}; //concentric circles
```

```cpp
  ld pd = (d2 + r1*r1 - r2*r2) / 2;
  ld h2 = r1*r1 - pd*pd/d2;
  point p = c1 + d*pd/d2;
  if(eq(h2, 0)) return {p}; //circles touch at one point
  else if(le(h2, 0)) return {}; //circles don't intersect
  else{
    point u = d.perp() * sqrt(h2/d2);
    return {p - u, p + u};
  }
}

int circleInsideCircle(const point & c1, ld r1, const point & c2, ld r2){
  //test if circle 2 is inside circle 1
  //returns "-1" if 2 touches internally 1, "1" if 2 is inside 1, "0" if they overlap
  ld l = r1 - r2 - (c1 - c2).length();
  return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

int circleOutsideCircle(const point & c1, ld r1, const point & c2, ld r2){
  //test if circle 2 is outside circle 1
  //returns "-1" if they touch externally, "1" if 2 is outside 1, "0" if they overlap
  ld l = (c1 - c2).length() - (r1 + r2);
  return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

int pointInCircle(const point & c, ld r, const point & p){
  //test if point p is inside the circle with center c and radius r
  //returns "0" if it's outside, "-1" if it's in the perimeter, "1" if it's inside
  ld l = (p - c).length() - r;
  return (le(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}

vector<vector<point>> tangents(const point & c1, ld r1, const point & c2, ld r2, bool inner){
  //returns a vector of segments or a single point
  if(inner) r2 = -r2;
  point d = c2 - c1;
  ld dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr*dr;
  if(eq(d2, 0) || le(h2, 0)) return {};
  point v = d*dr/d2;
  if(eq(h2, 0)) return {{c1 + v*r1}};
  else{
    point u = d.perp()*sqrt(h2)/d2;
    return {{c1 + (v - u)*r1, c2 + (v - u)*r2}, {c1 + (v + u)*r1, c2 + (v + u)*r2}};
  }
}

ld signed_angle(const point & a, const point & b){
  return sgn(a.cross(b)) * acosl(a.dot(b) / (a.length() * b.length()));
}

ld intersectPolygonCircle(const vector<point> & P, const point & c, ld r){
  //Gets the area of the intersection of the polygon with the circle
  int n = P.size();
  ld ans = 0;
  for(int i = 0; i < n; ++i){
    point p = P[i], q = P[(i+1)%n];
    bool p_inside = (pointInCircle(c, r, p) != 0);
    bool q_inside = (pointInCircle(c, r, q) != 0);
    if(p_inside && q_inside){
      ans += (p - c).cross(q - c);
    }else if(p_inside && !q_inside){
      point s1 = intersectSegmentCircle(p, q, c, r)[0];
      point s2 = intersectSegmentCircle(c, q, c, r)[0];
      ans += (p - c).cross(s1 - c) + r*r * signed_angle(s1 - c, s2 - c);
    }else if(!p_inside && q_inside){
      point s1 = intersectSegmentCircle(c, p, c, r)[0];
      point s2 = intersectSegmentCircle(p, q, c, r)[0];
      ans += (s2 - c).cross(q - c) + r*r * signed_angle(s1 - c, s2 - c);
    }else{
      auto info = intersectSegmentCircle(p, q, c, r);
      if(info.size() <= 1){
        ans += r*r * signed_angle(p - c, q - c);
      }else{
        point s2 = info[0], s3 = info[1];
        point s1 = intersectSegmentCircle(c, p, c, r)[0];
        point s4 = intersectSegmentCircle(c, q, c, r)[0];
        ans += (s2 - c).cross(s3 - c) + r*r * (signed_angle(s1 - c, s2 - c) + signed_angle(s3 - c, s4 - c));
      }
    }
  }
  return abs(ans)/2;
}

pair<point, ld> mec2(vector<point> & S, const point & a, const point & b, int n){
  ld hi = inf, lo = -hi;
  for(int i = 0; i < n; ++i){
    ld si = (b - a).cross(S[i] - a);
    if(eq(si, 0)) continue;
    point m = getCircle(a, b, S[i]).first;
    ld cr = (b - a).cross(m - a);
    if(le(si, 0)) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  ld v = (ge(lo, 0) ? lo : le(hi, 0) ? hi : 0);
  point c = (a + b) / 2 + (b - a).perp() * v / (b - a).norm();
  return {c, (a - c).norm()};
}

pair<point, ld> mec(vector<point> & S, const point & a, int n){
  random_shuffle(S.begin(), S.begin() + n);
  point b = S[0], c = (a + b) / 2;
  ld r = (a - c).norm();
  for(int i = 1; i < n; ++i){
    if(ge((S[i] - c).norm(), r)){
      tie(c, r) = (n == S.size() ? mec(S, S[i], i) : mec2(S, a, S[i], i));
    }
  }
  return {c, r};
}

pair<point, ld> smallestEnclosingCircle(vector<point> S){
  assert(!S.empty());
  auto r = mec(S, S[0], S.size());
  return {r.first, sqrt(r.second)};
}

bool comp1(const point & a, const point & b){
  return le(a.y, b.y);
}
pair<point, point> closestPairOfPoints(vector<point> P){
  sort(P.begin(), P.end(), comp1);
  set<point> S;
  ld ans = inf;
  point p, q;
  int pos = 0;
  for(int i = 0; i < P.size(); ++i){
    while(pos < i && geq(P[i].y - P[pos].y, ans)){
      S.erase(P[pos++]);
    }
    auto lower = S.lower_bound({P[i].x - ans - eps, -inf});
    auto upper = S.upper_bound({P[i].x + ans + eps, -inf});
    for(auto it = lower; it != upper; ++it){
      ld d = (P[i] - *it).length();
      if(le(d, ans)){
        ans = d;
        p = P[i];
        q = *it;
      }
    }
```

```cpp
   S.insert(P[i]);
  }
  return {p, q};
 }

struct vantage_point_tree {
  struct node
  {
   point p;
   ld th;
   node *l, *r;
  } *root;

  vector<pair<ld, point>> aux;

  vantage_point_tree(vector<point> &ps) {
   for(int i = 0; i < ps.size(); ++i)
    aux.push_back( { 0, ps[i] } );
   root = build(0, ps.size());
  }

  node *build(int l, int r) {
   if(l == r)
    return 0;
   swap(aux[l], aux[l + rand() % (r - l)]);
   point p = aux[l++].second;
   if(l == r)
    return new node( { p } );
   for(int i = l; i < r; ++i)
    aux[i].first = (p - aux[i].second).dot(p - aux[i].second);
   int m = (l + r) / 2;
   nth_element(aux.begin() + l, aux.begin() + m, aux.begin() + r);
   return new node( { p, sqrt(aux[m].first), build(l, m), build(m, r)
       ↪ } );
  }

  priority_queue<pair<ld, node*>> que;

  void k_nn(node *t, point p, int k) {
   if(!t)
    return;
   ld d = (p - t->p).length();
   if(que.size() < k)
    que.push( { d, t } );
   else if(ge(que.top().first, d)) {
    que.pop();
    que.push( { d, t } );
   }
   if(!t->l && !t->r)
    return;
   if(le(d, t->th)) {
     k_nn(t->l, p, k);
     if(leq(t->th - d, que.top().first))
      k_nn(t->r, p, k);
    } else {
     k_nn(t->r, p, k);
     if(leq(d - t->th, que.top().first))
      k_nn(t->l, p, k);
    }
   }

  vector<point> k_nn(point p, int k) {
   k_nn(root, p, k);
   vector<point> ans;
   for(; !que.empty(); que.pop())
    ans.push_back(que.top().second->p);
   reverse(ans.begin(), ans.end());
   return ans;
  }
 };

vector<point> minkowskiSum(vector<point> A, vector<point> B) {
  int na = (int)A.size(), nb = (int)B.size();
  if(A.empty() || B.empty()) return { };

  rotate(A.begin(), min_element(A.begin(), A.end()), A.end());
  rotate(B.begin(), min_element(B.begin(), B.end()), B.end());

  int pa = 0, pb = 0;
  vector<point> M;

  while(pa < na && pb < nb) {
   M.push_back(A[pa] + B[pb]);
   ld x = (A[(pa + 1) % na] - A[pa]).cross(B[(pb + 1) % nb] - B[pb]);
   if(leq(x, 0)) pb++;
   if(geq(x, 0)) pa++;
  }

  while(pa < na) M.push_back(A[pa++] + B[0]);
  while(pb < nb) M.push_back(B[pb++] + A[0]);

  return M;
 }

//Delaunay triangulation in O(n log n)
const point inf_pt(inf, inf);

struct QuadEdge {
  point origin;
  QuadEdge* rot = nullptr;
  QuadEdge* onext = nullptr;
  bool used = false;
  QuadEdge* rev() const { return rot->rot; }
  QuadEdge* lnext() const { return rot->rev()->onext->rot; }
  QuadEdge* oprev() const { return rot->onext->rot; }
  point dest() const { return rev()->origin; }
 };

QuadEdge* make_edge(const point & from, const point & to) {
  QuadEdge* e1 = new QuadEdge;
  QuadEdge* e2 = new QuadEdge;
  QuadEdge* e3 = new QuadEdge;
  QuadEdge* e4 = new QuadEdge;
  e1->origin = from;
  e2->origin = to;
  e3->origin = e4->origin = inf_pt;
  e1->rot = e3;
  e2->rot = e4;
  e3->rot = e2;
  e4->rot = e1;
  e1->onext = e1;
  e2->onext = e2;
  e3->onext = e4;
  e4->onext = e3;
  return e1;
 }

void splice(QuadEdge* a, QuadEdge* b) {
  swap(a->onext->rot->onext, b->onext->rot->onext);
  swap(a->onext, b->onext);
 }

void delete_edge(QuadEdge* e) {
  splice(e, e->oprev());
  splice(e->rev(), e->rev()->oprev());
  delete e->rot;
  delete e->rev()->rot;
  delete e;
  delete e->rev();
 }

QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
  QuadEdge* e = make_edge(a->dest(), b->origin);
  splice(e, a->lnext());
  splice(e->rev(), b);
  return e;
 }

bool left_of(const point & p, QuadEdge* e) {
  return ge((e->origin - p).cross(e->dest() - p), 0);
 }

bool right_of(const point & p, QuadEdge* e) {
```

```cpp
	return le((e->origin - p).cross(e->dest() - p), 0);
}

ld det3(ld a1, ld a2, ld a3, ld b1, ld b2, ld b3, ld c1, ld c2, ld c3) {
	return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) + a3 * (b1 * c2 - c1 * b2);
}

bool in_circle(const point & a, const point & b, const point & c, const point & d) {
	ld det = -det3(b.x, b.y, b.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm());
	det += det3(a.x, a.y, a.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm());
	det -= det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), d.x, d.y, d.norm());
	det += det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), c.x, c.y, c.norm());
	return ge(det, 0);
}

pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<point> & P) {
	if(r - l + 1 == 2){
		QuadEdge* res = make_edge(P[l], P[r]);
		return {res, res->rev()};
	}
	if(r - l + 1 == 3){
		QuadEdge *a = make_edge(P[l], P[l + 1]), *b = make_edge(P[l + 1], P[r]);
		splice(a->rev(), b);
		int sg = sgn((P[l + 1] - P[l]).cross(P[r] - P[l]));
		if(sg == 0)
			return {a, b->rev()};
		QuadEdge* c = connect(b, a);
		if(sg == 1)
			return {a, b->rev()};
		else
			return {c->rev(), c};
	}
	int mid = (l + r) / 2;
	QuadEdge *ldo, *ldi, *rdo, *rdi;
	tie(ldo, ldi) = build_tr(l, mid, P);
	tie(rdi, rdo) = build_tr(mid + 1, r, P);
	while(true){
		if(left_of(rdi->origin, ldi)){
			ldi = ldi->lnext();
			continue;
		}
		if(right_of(ldi->origin, rdi)){
			rdi = rdi->rev()->onext;
			continue;
		}
		break;
	}
	QuadEdge* basel = connect(rdi->rev(), ldi);
	auto valid = [&basel](QuadEdge* e){ return right_of(e->dest(), basel); };
	if(ldi->origin == ldo->origin)
		ldo = basel->rev();
	if(rdi->origin == rdo->origin)
		rdo = basel;
	while(true){
		QuadEdge* lcand = basel->rev()->onext;
		if(valid(lcand)){
			while(in_circle(basel->dest(), basel->origin, lcand->dest(), lcand->onext->dest())){
				QuadEdge* t = lcand->onext;
				delete_edge(lcand);
				lcand = t;
			}
		}
		QuadEdge* rcand = basel->oprev();
		if(valid(rcand)){
			while(in_circle(basel->dest(), basel->origin, rcand->dest(), rcand->oprev()->dest())){
				QuadEdge* t = rcand->oprev();
				delete_edge(rcand);
				rcand = t;
			}
		}
		if(!valid(lcand) && !valid(rcand))
			break;
		if(!valid(lcand) || (valid(rcand) && in_circle(lcand->dest(), lcand->origin, rcand->origin, rcand->dest())))
			basel = connect(rcand, basel->rev());
		else
			basel = connect(basel->rev(), lcand->rev());
	}
	return {ldo, rdo};
}

vector<tuple<point, point, point>> delaunay(vector<point> & P){
	sort(P.begin(), P.end());
	auto res = build_tr(0, (int)P.size() - 1, P);
	QuadEdge* e = res.first;
	vector<QuadEdge*> edges = {e};
	while(le((e->dest() - e->onext->dest()).cross(e->origin - e->onext->dest()), 0))
		e = e->onext;
	auto add = [&P, &e, &edges](){
		QuadEdge* curr = e;
		do{
			curr->used = true;
			P.push_back(curr->origin);
			edges.push_back(curr->rev());
			curr = curr->lnext();
		}while(curr != e);
	};
	add();
	P.clear();
	int kek = 0;
	while(kek < (int)edges.size())
		if(!(e = edges[kek++])->used)
			add();
	vector<tuple<point, point, point>> ans;
	for(int i = 0; i < (int)P.size(); i += 3){
		ans.emplace_back(P[i], P[i + 1], P[i + 2]);
	}
	return ans;
}

struct circ{
	point c;
	ld r;
	circ() {}
	circ(const point & c, ld r): c(c), r(r) {}
	set<pair<ld, ld>> ranges;

	void disable(ld l, ld r){
		ranges.emplace(l, r);
	}

	auto getActive() const{
		vector<pair<ld, ld>> ans;
		ld maxi = 0;
		for(const auto & dis : ranges){
			ld l, r;
			tie(l, r) = dis;
			if(l > maxi){
				ans.emplace_back(maxi, l);
			}
			maxi = max(maxi, r);
		}
		if(!eq(maxi, 2*pi)){
			ans.emplace_back(maxi, 2*pi);
		}
		return ans;
	}
};

ld areaUnionCircles(const vector<circ> & circs){
```

```
 vector<circ> valid;
 for(const circ & curr : circs){
  if(eq(curr.r, 0)) continue;
  circ nuevo = curr;
  for(circ & prev : valid){
   if(circleInsideCircle(prev.c, prev.r, nuevo.c, nuevo.r)){
    nuevo.disable(0, 2*pi);
   }else if(circleInsideCircle(nuevo.c, nuevo.r, prev.c, prev.r)){
    prev.disable(0, 2*pi);
   }else{
    auto cruce = intersectionCircles(prev.c, prev.r, nuevo.c, nuevo.r);
    if(cruce.size() == 2){
     ld a1 = (cruce[0] - prev.c).ang();
     ld a2 = (cruce[1] - prev.c).ang();
     ld b1 = (cruce[1] - nuevo.c).ang();
     ld b2 = (cruce[0] - nuevo.c).ang();
     if(a1 < a2){
      prev.disable(a1, a2);
     }else{
      prev.disable(a1, 2*pi);
      prev.disable(0, a2);
     }
     if(b1 < b2){
      nuevo.disable(b1, b2);
     }else{
      nuevo.disable(b1, 2*pi);
      nuevo.disable(0, b2);
     }
    }
   }
  }
  valid.push_back(nuevo);
 }
 ld ans = 0;
 for(const circ & curr : valid){
  for(const auto & range : curr.getActive()){
   ld l, r;
   tie(l, r) = range;
   ans += curr.r*(curr.c.x * (sin(r) - sin(l)) - curr.c.y * (cos(r) - cos(l))) + curr.r*curr.r*(r-l);
  }
 }
 return ans/2;
};

struct plane{
 point a, v;
 plane(): a(), v(){}
 plane(const point& a, const point& v): a(a), v(v){}

 point intersect(const plane& p) const{
  ld t = (p.a - a).cross(p.v) / v.cross(p.v);
  return a + v*t;
 }

 bool outside(const point& p) const{ // test if point p is strictly outside
  return le(v.cross(p - a), 0);
 }

 bool inside(const point& p) const{ // test if point p is inside or in the boundary
  return geq(v.cross(p - a), 0);
 }

 bool operator<(const plane& p) const{ // sort by angle
  auto lhs = make_tuple(v.half({1, 0}), ld(0), v.cross(p.a - a));
  auto rhs = make_tuple(p.v.half({1, 0}), v.cross(p.v), ld(0));
  return lhs < rhs;
 }

 bool operator==(const plane& p) const{ // paralell and same directions, not really equal
  return eq(v.cross(p.v), 0) && ge(v.dot(p.v), 0);
 }
};

vector<point> halfPlaneIntersection(vector<plane> planes){
 planes.push_back({{0, -inf}, {1, 0}});
 planes.push_back({{inf, 0}, {0, 1}});
 planes.push_back({{0, inf}, {-1, 0}});
 planes.push_back({{-inf, 0}, {0, -1}});
 sort(planes.begin(), planes.end());
 planes.erase(unique(planes.begin(), planes.end()), planes.end());
 deque<plane> ch;
 deque<point> poly;
 for(const plane& p : planes){
  while(ch.size() >= 2 && p.outside(poly.back())) ch.pop_back(), poly.pop_back();
  while(ch.size() >= 2 && p.outside(poly.front())) ch.pop_front(), poly.pop_front();
  if(p.v.half({1, 0}) && poly.empty()) return {};
  ch.push_back(p);
  if(ch.size() >= 2) poly.push_back(ch[ch.size()-2].intersect(ch[ch.size()-1]));
 }
 while(ch.size() >= 3 && ch.front().outside(poly.back())) ch.pop_back(), poly.pop_back();
 while(ch.size() >= 3 && ch.back().outside(poly.front())) ch.pop_front(), poly.pop_front();
 poly.push_back(ch.back().intersect(ch.front()));
 return vector<point>(poly.begin(), poly.end());
}

vector<point> halfPlaneIntersectionRandomized(vector<plane> planes){
 point p = planes[0].a;
 int n = planes.size();
 random_shuffle(planes.begin(), planes.end());
 for(int i = 0; i < n; ++i){
  if(planes[i].inside(p)) continue;
  ld lo = -inf, hi = inf;
  for(int j = 0; j < i; ++j){
   ld A = planes[j].v.cross(planes[i].v);
   ld B = planes[j].v.cross(planes[j].a - planes[i].a);
   if(ge(A, 0)){
    lo = max(lo, B/A);
   }else if(le(A, 0)){
    hi = min(hi, B/A);
   }else{
    if(ge(B, 0)) return {};
   }
   if(ge(lo, hi)) return {};
  }
  p = planes[i].a + planes[i].v*lo;
 }
 return {p};
}
```

8.2 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 << ")"; }
};
```

8.3 Misc. Point Set Problems

ClosestPair.h

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

ac9ec9, 18 lines

```
#include "Point.h"
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;
}
```

kdTree.h

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

08fbca, 55 lines

```
#include "Point.h"
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);
  }
};
```

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

e8d11f, 83 lines

```
#include "Point.h"
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;
}
```

8.4 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: $\mathcal{O}(n^2)$

c114bf, 46 lines

```
#include "Point3D.h"
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);
}
```

Strings (9)

KMP.h

Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s

$$0 \ldots x$$

itself (abacaba -> 0010123). Can be used to find all occurrences of a string.
Time: $\mathcal{O}(n)$

a933ca, 29 lines

```
vector<int> p_function(const string& v){
    int n = v.size();
```

```
  vector<int> p(n);
  for(int i = 1; i < n; i++) {
    int j = p[i - 1];
    while(j > 0 && v[j] != v[i]) {
      j = p[j - 1];
    }
    if(v[j] == v[i])
      j++;
    p[i] = j;
  }
  return p;
}
vector<int> match(const string& s, const string& pat) {
 int n = pat.size();
 int m = s.size();
 if(m<n) {
  cout<<endl;
  continue;
 }
 string match = pat+"#"+s;
 vector<int> kmp =p_function(match);
 vector<int> ans(m-n+1);
 for(int i = 0; i < m - n + 1; i++)
  if(kmp[2 * n + i] == n)
   ans[i] = 1;
 for(int i = 0;i<ans.size();i++)if(ans[i])cout<<i<<" ";
}
```

Zfunc.h

Description: z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$

2895e4, 13 lines

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

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).
Time: $\mathcal{O}(N)$

d4fde9, 15 lines

```
vector<vector<int>> manacher(const string& s) {
 n = s.size();
 vector<vector<int>> p(2,vector<int>(n,0));
  for(int z=1,l=0,r=0;z<2;z++,l=0,r=0)
    for(int i=0;i<n;i++)
    {
      if(i<r) p[z][i]=min(r-i+!z,p[z][l+r-i+!z]);
      int L=i-p[z][i], R=i+p[z][i]-!z;
      cout<<L<<" "<<R<<" "<<(L-1>=0) <<" "<<(R+1<n)<<endl;
      while(L-1>=0 && R+1<n && s[L-1]==s[R+1]) p[z][i]++,L--,R++;
      if(R>r) l=L,r=R;
    }
  for(int i = 0;i<n;i++)cout<<p[0][i]<<" "<<p[1][i]<<endl;
 return p;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.
Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
Time: $\mathcal{O}(N)$

975539, 9 lines

```
int minRotation(string s) {
 int a=0, N=sz(s); s += s;
 for(int b = 0;b<N;b++)
  for(int k = 0;k<N;k++) {
   if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
   if (s[a+k] > s[b+k]) {a = b; break; }
  }
 return a;
}
```

SuffixArray.h

Description: Builds suffix array for a string. `sa[i]` is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size $n+1$, and `sa[0] = n`. The `lcp` array contains longest common prefixes for neighbouring strings in the suffix array: `lcp[i] = lcp(sa[i], sa[i-1])`, `lcp[0] = 0`. The input string must not contain any zero bytes.
Time: $\mathcal{O}(n \log n)$

237f0d, 70 lines

```
void radix_sort(vector<int> &P,vector<int> &c) {
  int n = P.size();
  vector<int> cnt(n);
  for(auto d:c)
    cnt[d]++;
  vector<int> pos(n);
  vector<int> nP(n);
  pos[0]= 0;
  for(int i = 1;i<n;i++)
    pos[i] = pos[i-1]+cnt[i-1];
  for(auto d:P) {
    int i = c[d];
    nP[pos[i]] = d;
    pos[i]++;
  }
  P = nP;
}
// SuffixArray and LCP (Longest common preffix)
void suffixArray(string s) {
 s+=char(31);
 int N;
 cin>>N;
 vector<int> nums(N);
 for(auto &c:nums)cin>>c;
 int n = s.size();
 vector<int>c(n);
 vector<int>p(n);
 vector<pair<char,int>> a(n);
 for(int i = 0;i<n;i++)a[i] = {s[i],i};
 sort(a.begin(),a.end());
 for(int i = 0;i<n;i++)
  p[i] = a[i].second;
 c[p[0]] = 0;
 for(int i = 1;i<n;i++) {
  if(a[i].first == a[i-1].first)
   c[p[i]] = c[p[i-1]];
  else c[p[i]] = c[p[i-1]]+1;
 }
 int k = 0;
 while((1<<k)<n) {
  for(int i = 0 ;i<n;i++)
   p[i] = ((p[i]-(1<<k))+n)%n;
  radix_sort(p,c);
  vector<int> nC(n);
  nC[p[0]] = 0;
  for(int i = 1;i<n;i++) {
   pair<int,int> prev = {c[p[i-1]],c[(p[i-1]+ (1<<k))%n]};
   pair<int,int> now = {c[p[i]],c[(p[i]+ (1<<k))%n]};
   if(prev == now)
    nC[p[i]] = nC[p[i-1]];
   else nC[p[i]] = nC[p[i-1]]+1;
  }
  c = nC;
  k++;
 }
 // LCP O(n)
 k = 0;
 vector<int> lcp(n);
 for(int i = 0;i<n-1;i++) {
  int x = c[i];
  int j = p[x-1];
  while(s[i+k] == s[j+k])k++;
  lcp[x] = k;
  k = max(k-1,0ll);
 }
 for(int i = 0;i<N;i++)cout<<p[nums[i]+1]<<" ";
 cout<<endl;
```

```
  for(int i = 0;i<n;i++)cout<<lcp[i]<<" "<<p[i]<<" "<<s.substr(p[i],n
      ↪ -p[i])<<endl;
  cout<<endl;
}
```

SuffixTree.h

Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).
Time: $\mathcal{O}(26N)$

797ab6, 59 lines

```
const int inf = 1e9;
const int maxn = 1e6 ;
char s[maxn];
map<int, int> to[maxn];
int len[maxn], start[maxn], link[maxn];
int node, remaind;
int sz = 1, n = 0;
int make_node(int _pos, int _len){
  start[sz] = _pos;
  len [sz] = _len;
  return sz++;
}
void go_edge(){
  while(remaind > len[to[node][s[n - remaind]]]){
    node = to[node][s[n - remaind]];
    remaind -= len[node];
  }
}
void add_letter(int c){
  s[n++] = c;
  remaind++;
  int last = 0;
  while(remaind > 0){
    go_edge();
    int edge = s[n - remaind];
    int &v = to[node][edge];
    int t = s[start[v] + remaind - 1];
    if(v == 0){
      v = make_node(n - remaind, inf);
      link[last] = node;
      last = 0;
    }
    else if(t == c){
      link[last] = node;
      return;
    }
    else{
      int u = make_node(start[v], remaind - 1);
      to[u][c] = make_node(n - 1, inf);
      to[u][t] = v;
      start[v] += remaind - 1;
      len [v] -= remaind - 1;
      v = u;
      link[last] = u;
      last = u;
    }
    if(node == 0)
      remaind--;
    else
      node = link[node];
  }
}
bool dfsForPrint(int node,char edge){
  if(node != 0)
    cout<<edge<<" "<<node<<" "<<len[node]<<" "<<start[node]<<endl;
  for(auto c:to[node])
    dfsForPrint(c.second,c.first);
  return 0 ;
}
```

Hashing-codeforces.h

Description: Various self-explanatory methods for string hashing. Use on Codeforces, which lacks 64-bit support and where solutions can be hacked.

0207eb, 46 lines

```
typedef uint64_t ull;
static int C; // initialized below
// Arithmetic mod two primes and 2^32 simultaneously.
// "typedef uint64_t H;" instead if Thue-Morse does not apply.
template<int M, class B>
struct A {
  int x; B b; A(int x=0) : x(x), b(x) {}
  A(int x, B b) : x(x), b(b) {}
  A operator+(A o){int y = x+o.x; return{y - (y>=M)*M, b+o.b};}
  A operator-(A o){int y = x-o.x; return{y + (y< 0)*M, b-o.b};}
  A operator*(A o) { return {(int)(1LL*x*o.x % M), b*o.b}; }
  explicit operator ull() { return x ^(ull) b << 21; }
};
typedef A<1000000007, A<1000000009, unsigned>> H;
struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};
vector<H> getHashes(string& str, int length) {
  if (sz(str) < length) return {};
  H h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}
H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
#include <sys/time.h>
int main() {
  timeval tp;
  gettimeofday(&tp, 0);
  C = (int)tp.tv_usec; // (less than modulo)
  assert((ull)(H(1)*2+1-3) == 0);
  // ...
}
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(−, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
Considerations: The exit links are a compression of links , with exit links go directly to the next node that is a end of some word
Usage: `new_node(-1,'#'); //Init`
Time: construction takes $\mathcal{O}(kN)$, where N = sum of length of patterns and K is the size of alphabet. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$.

7e300b, 64 lines

```
const int k =200;
// to use all ASCII use an space as start or 'A' with k= 60 if it is
    ↪ only lower/upper case english letters
// Use arrays instead of struct because is much fate r in that way,
    ↪ tested in I love string with struct didn´t pass
char start = ' ';
const int maxn = 200000;
int go_state[maxn][k];
bool leaf[maxn];
int link[maxn];
int exitLink[maxn];
vector<int> ids[maxn];
char pch[maxn];
int p[maxn];
string word[maxn];
int key = 0;
int new_node(int v,char c){
  p[key] =v;
  pch[key] = c;
```

```
    return key++;
}
void add_string(string s,int id){
    int v = 0;
    for(auto c:s){
        int id = c-start;
        if(go_state[v][id]==-1)go_state[v][id] = new_node(v,c);
        v = go_state[v][id];
    }
    leaf[v] = true;
    word[v] = s;
    ids[v].push_back(id);
}
int go(int v,char c);
int get_link(int v){
    if(link[v] ==-1){
        if(v ==0 || p[v] == 0)
            link[v] = 0;
        else
            link[v] = go(get_link(p[v]),pch[v]);
    }
    return link[v];
}
int go(int v,char c){
    int id = c-start;
    if(go_state[v][id]==-1)
        go_state[v][id] = v==0?0:go(get_link(v),c);
    return go_state[v][id];
}
int get_exit_link(int v){
    if(v==0)return exitLink[v] = 0;
    if(exitLink[v]==-1){
        exitLink[v] = get_exit_link(link[v]);
    }
    return leaf[v]?v:exitLink[v];
}
void clean(int N){
    for(int i = 0;i<N;i++){
        exitLink[i] = -1;
        link[i] = -1;
        leaf[i] = false;
        ids[i].clear();
        word[i] = "";
        for(int j = 0;j<k;j++)
            go_state[i][j] = -1;
    }
}
```

palindromicTree.h

Description: A nice data structure allowing to solve some interesting problems involving palindromes.

8feb90, 51 lines

```
const int MAXN = 105000;
struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};
int len;
char s[MAXN];
node tree[MAXN];
int num;        // node 1 - root with len -1, node 2 - root with len 0
int suff;       // max suffix palindrome
long long ans;
bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';
    while (true) {
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }
    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;
    if (tree[num].len == 1) {
        tree[num].sufflink = 2;
        tree[num].num = 1;
        return true;
    }
    while (true) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }
    tree[num].num = 1 + tree[tree[num].sufflink].num;
    return true;
}
void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}
```

Various (10)

10.1 Misc

various.cpp

168 lines

```
int lcs(string & a, string & b){
  int m = a.size(), n = b.size();
  vector<vector<int>> aux(m + 1, vector<int>(n + 1));
  for(int i = 1; i <= m; ++i){
    for(int j = 1; j <= n; ++j){
      if(a[i - 1] == b[j - 1])
        aux[i][j] = 1 + aux[i - 1][j - 1];
      else
        aux[i][j] = max(aux[i - 1][j], aux[i][j - 1]);
    }
  }
  return aux[m][n];
}

//0:saturday, 1:sunday, ..., 6:friday
int dayOfWeek(int d, int m, lli y){
  if(m == 1 || m == 2){
    m += 12;
    --y;
  }
  int k = y % 100;
  lli j = y / 100;
  return (d + 13*(m+1)/5 + k + k/4 + j/4 + 5*j) % 7;
}

//cout for __int128
ostream &operator<<(ostream &os, const __int128 & value){
  char buffer[64];
  char *pos = end(buffer) - 1;
  *pos = '\0';
  __int128 tmp = value < 0 ? -value : value;
  do{
    --pos;
    *pos = tmp % 10 + '0';
    tmp /= 10;
  }while(tmp != 0);
  if(value < 0){
    --pos;
    *pos = '-';
  }
  return os << pos;
}

//cin for __int128
istream &operator>>(istream &is, __int128 & value){
  char buffer[64];
```

```
 is >> buffer;
 char *pos = begin(buffer);
 int sgn = 1;
 value = 0;
 if(*pos == '-'){
  sgn = -1;
  ++pos;
 }else if(*pos == '+'){
  ++pos;
 }
 while(*pos != '\0'){
  value = (value << 3) + (value << 1) + (*pos - '0');
  ++pos;
 }
 value *= sgn;
 return is;
}

int LevenshteinDistance(string & a, string & b){
 int m = a.size(), n = b.size();
 vector<vector<int>> aux(m + 1, vector<int>(n + 1));
 for(int i = 1; i <= m; ++i)
  aux[i][0] = i;
 for(int j = 1; j <= n; ++j)
  aux[0][j] = j;
 for(int j = 1; j <= n; ++j)
  for(int i = 1; i <= m; ++i)
   aux[i][j] = min({aux[i-1][j] + 1, aux[i][j-1] + 1, aux[i-1][j-1] + (a[i-1] != b[j-1])});
 return aux[m][n];
}
// far pair of points manhatan
lli minz= 2e9,maxz = -2e9,miny = 2e9,maxy = -2e9;
for(int i = 0;i<n;i++){
  cin>>x>>y;
  minz = min(minz,x+y);
  maxz = max(maxz,x+y);
  miny = min(miny,x-y);
  maxy = max(maxy,x-y);
}
cout<<max(maxz-minz,maxy-miny)<<endl;

// DIgit DP template
int a,b,k;
int DP[20][2][200][200];
vector<int> Num;
int go(int pos,int f,int sum,int rem){
  if(pos == Num.size()){
    if(sum%k == 0 && rem%k == 0)
      return 1;
    return 0;
  }

  int &x = DP[pos][f][sum][rem];
  if(x!= -1)return x;
  int res = 0;
  int LIM ;
  if(!f)LIM = Num[pos];
  else LIM = 9;
  for(int i =0;i<=LIM;i++){
    int nf = f;
    if(i<LIM)nf = 1;
    res +=go(pos+1,nf,(sum+i)%k,(rem*10+i)%k);
  }
  return x = res;
}

int solve(int n){
  Num.clear();
  while(n){
    Num.push_back(n%10);
    n/=10;
  }
  memset(DP,-1,sizeof(DP));
  reverse(Num.begin(),Num.end());
  return go(0,0,0,0);
}

//DP de corte template
const int maxn = 2007;
const int maxk = 22;
int DP[maxn][maxk];
const int inf = 1000000000;
int n;
vector<int> ac(maxn);
int cost(int i,int j){
  int sum;
  if(i)
    sum = ac[j]-ac[i-1];
  else
    sum = ac[j];
  if(sum%10<5)sum-=sum%10;
  else sum+=10-sum%10;
  return sum;
}
// return min or max of proffit cutting an array in exactly k parts
// Complexity O(n^2 *k * cost)
int solve(int idx,int k){
  if(k==0 && idx==n)return 0;
  else if(k == 0 && idx<n)return inf+1;
  int &x = DP[idx][k];
  if(x!= inf)return x;
  for(int i = idx;i<n;i++)
    x = min(x,cost(idx,i)+solve(i+1,k-1));

  return x;
}

//Random number generation in C++11
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

//Generate a random integer in [a, b], you can also use long long int
int aleatorio_int(int a, int b){
 uniform_int_distribution<int> dist(a, b);
 return dist(rng);
}

//Generate a random double in [a, b], you can also use long double
double aleatorio_double(double a, double b){
 uniform_real_distribution<double> dist(a, b);
 return dist(rng);
}
```

10.2 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

edce47, 22 lines

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
 if (L == R) return is.end();
 auto it = is.lower_bound({L, R}), before = it;
 while (it != is.end() && it->first <= R) {
  R = max(R, it->second);
  before = it = is.erase(it);
 }
 if (it != is.begin() && (--it)->second >= L) {
  L = min(L, it->first);
  R = max(R, it->second);
  is.erase(it);
 }
 return is.insert(before, {L,R});
}
void removeInterval(set<pii>& is, int L, int R) {
 if (L == R) return;
 auto it = addInterval(is, L, R);
 auto r2 = it->second;
 if (it->first == L) is.erase(it);
 else (int&)it->second = L;
 if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

9e9d8d, 19 lines

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
 vi S(sz(I)), R;
 iota(all(S), 0);
 sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
 T cur = G.first;
 int at = 0;
 while (cur < G.second) { // (A)
  pair<T, int> mx = make_pair(cur, -1);
  while (at < sz(I) && I[S[at]].first <= cur) {
   mx = max(mx, make_pair(I[S[at]].second, S[at]));
   at++;
  }
  if (mx.second == -1) return {};
  cur = mx.first;
  R.push_back(mx.second);
 }
 return R;
}
```

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

753a4c, 19 lines

```
template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
 if (p == q) return;
 if (from == to) {
  g(i, to, p);
  i = to; p = q;
 } else {
  int mid = (from + to) >> 1;
  rec(from, mid, f, g, i, p, f(mid));
  rec(mid+1, to, f, g, i, p, q);
 }
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
 if (to <= from) return;
 int i = from; auto p = f(i), q = f(to-1);
 rec(from, to-1, f, g, i, p, q);
 g(i, to, q);
}
```

10.3 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$

9155b4, 11 lines

```
template<class F>
int ternSearch(int a, int b, F f) {
 assert(a <= b);
 while (b - a >= 5) {
  int mid = (a + b) / 2;
  if (f(mid) < f(mid+1)) a = mid; // (A)
  else b = mid+1;
 }
 rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
 return a;
}
```

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$

2932a0, 17 lines

```
template<class I> vi lis(const vector<I>& S) {
 if (S.empty()) return {};
 vi prev(sz(S));
 typedef pair<I, int> p;
 vector<p> res;
 rep(i,0,sz(S)) {
  // change 0 -> i for longest non-decreasing subsequence
  auto it = lower_bound(all(res), p{S[i], 0});
  if (it == res.end()) res.emplace_back(), it = res.end()-1;
  *it = {S[i], i};
  prev[i] = it == res.begin() ? 0 : (it-1)->second;
 }
 int L = sz(res), cur = res.back().second;
 vi ans(L);
 while (L--) ans[L] = cur, cur = prev[cur];
 return ans;
}
```

10.4 Dynamic programming

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i, j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b, c) \leq f(a, d)$ and $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$ for all $a \leq b \leq c \leq d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \leq k < hi(i)}(f(i, k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}((N + (hi - lo)) \log N)$

d38d2b, 17 lines

```
struct DP { // Modify at will:
 int lo(int ind) { return 0; }
 int hi(int ind) { return ind; }
 ll f(int ind, int k) { return dp[ind][k]; }
 void store(int ind, int k, ll v) { res[ind] = pii(k, v); }
 void rec(int L, int R, int LO, int HI) {
  if (L >= R) return;
  int mid = (L + R) >> 1;
  pair<ll, int> best(LLONG_MAX, LO);
  rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
   best = min(best, make_pair(f(mid, k), k));
  store(mid, best.second, best.first);
  rec(L, mid, LO, best.second+1);
  rec(mid+1, R, best.second, HI);
 }
 void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

10.5 Optimization tricks

10.5.1 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.

10.5.2 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).

FastMod.h

Description: Compute $a\%b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to $a \pmod{b}$ in the range $[0, 2b)$.

751a02, 8 lines

```
typedef unsigned long long ull;
struct FastMod {
 ull b, m;
 FastMod(ull b) : b(b), m(-1ULL / b) {}
```

```
  ull reduce(ull a) { // a % b + (0 or b)
    return a - (ull)((__uint128_t(m) * a) >> 64) * b;
  }
};
```

FastInput.h

Description: Read an integer from stdin. Usage requires your program to pipe in input from file.
Usage: `./a.out < input.txt`
Time: About 5x as fast as cin/scanf.

7b3c70, 16 lines

```
inline char gc() { // like getchar()
  static char buf[1 << 16];
  static size_t bc, be;
  if (bc >= be) {
    buf[0] = 0, bc = 0;
    be = fread(buf, 1, sizeof(buf), stdin);
  }
  return buf[bc++]; // returns 0 on EOF
}
int readInt() {
  int a, c;
  while ((a = gc()) < 40);
  if (a == '-') return -readInt();
  while ((c = gc()) >= 48) a = a * 10 + c - 480;
  return a - 48;
}
```

Techniques (A)

techniques.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
```