"Alexandru Ioan Cuza" University of Iasi

Endgame

Cristian Vintur, Denis Banu, Matei Chiriac

ACM-ICPC World Finals 2021

November 2022

Contest (1)

templateCristi.cpp
39 lines

```cpp
//https://gitlab.com/cristi.vintur/cp-templates
#include <bits/stdc++.h>
using namespace std;
using uint = unsigned int;
using ll = long long;
using ld = long double;
using ull = unsigned long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
#define dbg(...) cerr << #__VA_ARGS__ << " ->", debug_out(
     __VA_ARGS__)
#define dbg_p(x) cerr<<#x": "<<(x).first<<' '<<(x).second<<endl
#define dbg_v(x, n) {cerr<<#x"[]: ";for(long long _=0;_<n;++_)
     cerr<<(x)[_]<<' ';cerr<<endl;}
#define all(v) v.begin(), v.end()
#define fi first
#define se second
void debug_out() { cerr << endl; }
template <typename Head, typename... Tail> void debug_out(Head
     H, Tail... T) { cerr << " " << H; debug_out(T...);}

template<typename T1, typename T2>
ostream& operator <<(ostream &out, const pair<T1, T2> &item) {
  out << '(' << item.first << ", " << item.second << ')';
  return out;
}

template <typename T>
ostream& operator <<(ostream &out, const vector<T>& v) {
  for(const auto &item : v) out << item << ' ';
  return out;
}

//const int N = ;

int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  return 0;
}
```

templateMatei.txt
126 lines

```
Ordered set

#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
     tree_order_statistics_node_update

using namespace __gnu_pbds;

typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;

Random

#include <algorithm>
#include <chrono>
#include <iostream>
#include <random>
#include <vector>

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

Turn stack DS into queue DS

When provided a B update (add B), we just push it to the top of
      S.
When provided an A
update: If S already had A on top, we just pop. Otherwise, we
     begin the following process (which I'll call "fixing"):
pop from S and save all the elements we popped, until we popped
      an equal amount of A's and B's,
or until no A's remain in the stack; empty stack
or only B's remain (we can keep an index of this position in
     the stack, which will only increase).
Then, push back all the elements we popped, where we first push
      all B's,
then all A's (we use commutativity here).
Since the top of S had a B, and we were asked to pop an
     existing A, after fixing,
the topmost element will be an A which we pop.

struct queue_dsu
{
  int bottom;
  vector<pos> act;
  stack_dsu s;

  void init(int n)
  {
    s.init(n);
  }
  void fmerge(int x, int y)
  {
    act.push_back(pos(x, y));
    s.fmerge(x, y);
  }

  void move_bottom()
  {
    while(bottom<act.size() && act[bottom].t=='B')
      bottom++;
  }
  void reverse_upd()
  {
    for(int i=0;i<act.size();i++)
    {
      s.pop();
      act[i].t='A';
    }

    reverse(act.begin(), act.end());

    for(int i=0;i<act.size();i++)
            s.fmerge(act[i].x, act[i].y);

        bottom=0;
    }
    void fix()
    {
        if(act.empty() || act.back().t=='A')
            return;

        move_bottom();
        vector<pos> va, vb;
        vb.push_back(act.back());
        act.pop_back();
        s.pop();

        while(va.size()!=vb.size() && act.size()>bottom)
        {
            if(act.back().t=='A')
                va.push_back(act.back());
            else
                vb.push_back(act.back());

            act.pop_back();
            s.pop();
        }

        reverse(va.begin(), va.end());
        reverse(vb.begin(), vb.end());

        for(auto it : vb)
        {
            act.push_back(it);
            s.fmerge(it.x, it.y);
        }
        for(auto it : va)
        {
            act.push_back(it);
            s.fmerge(it.x, it.y);
        }
        move_bottom();
    }
    void pop()
    {
        move_bottom();
        if(bottom==act.size())
            reverse_upd();

        fix();
        act.pop_back();
        s.pop();
    }
};
```

gen.py
21 lines

```python
import sys

from random import randint, choice, shuffle, uniform, sample
from math import gcd, sqrt
from string import ascii_lowercase, ascii_uppercase

chars = ['0', '1']

def random_string(n, chars = ascii_lowercase):
  return "".join([choice(chars) for _ in range(n)])

def print_list(l, sep = " "):
  print(sep.join([str(item) for item in l]))

def gen_graph_edges(n, m):
  edges = []
  for i in range(1, n):
    edges.append((randint(0, i - 1), i))
  for nr in range(n, m + 1):
    edges.append((randint(0, n - 1), randint(0, n - 1)))
  return edges
```

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 datastructures 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 team mate.
Ask the team mate 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 team mate 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 team mates think about your algorithm?

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

stresstest.sh 31 lines

```bash
#/bin/bash

i=0
while true
do
  #python3 gen.py >in
  #./gen >in
  ./generators/graph >in
  ./c <in >out
  ./d <in >ok
  #python3 verif.py

  #if [ $? -eq 1 ]; then
  #   echo $?
  #   exit 1
  #fi

  if ! diff out ok; then
    echo $?
    exit 1
  fi

  #if ((i == 1000)); then
  # exit 0
  #fi

  let i=i+1
  if ((i % 1 == 0)); then
    echo $i
  fi
done
```

Mathematics (2)

2.1 Equations

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

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 $a_n=c_1a_{n-1}+\cdots+c_ka_{n-k}$, and $r_1,\ldots,r_k$ are distinct roots of $x^k+c_1x^{k-1}+\cdots+c_k$, there are $d_1,\ldots,d_k$ s.t.

$$a_n=d_1r_1^n+\cdots+d_kr_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n=(d_1n+d_2)r^n$.

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: $\dfrac{a+b}{a-b}=\dfrac{\tan\dfrac{\alpha+\beta}{2}}{\tan\dfrac{\alpha-\beta}{2}}$

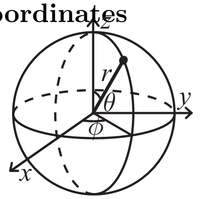
2.4.2 Quadrilaterals

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



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

2.5 Linear algebra

2.5.1 Matrix inverse

The inverse of a 2x2 matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

In general:

$$A^{-1} = \frac{1}{det(A)}A^*$$

where $A^*_{i,j} = (-1)^{i+j} * \Delta_{i,j}$ and $\Delta_{i,j}$ is the determinant of matrix A crossing out line i and column j.

2.6 Derivatives/Integrals

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

Integration by parts:

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

2.7 Sums

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

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

2.8 Series

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

2.9 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 $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x-\mathbb{E}(X))^2 p_X(x)$ 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,

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

2.9.1 Discrete distributions

Binomial distribution

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

$$p(k) = \binom{n}{k}p^k(1-p)^{n-k}$$
$$\mu = np, \sigma^2 = np(1-p)$$

Bin(n,p) is approximately 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 Fs(p), $0 \leq p \leq 1$.

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

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 Po(λ), $\lambda = t\kappa$.

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

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

2.9.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is 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 Exp(λ), $\lambda > 0$.

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

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

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

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

2.10 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)

binomialHeap.cpp
Description: wtf 200 lines

```
const int INF = 2000000001;
const int NMAX = 101;

struct Node {
    int key, degree;
    Node *child, *sibling, *parent;
};

Node* newNode( int val ){
    Node* temp = new Node;
    temp -> key = val;
    temp -> degree = 0;
    temp -> child = temp -> sibling = temp -> parent = NULL;
    return temp;
}

class BinomialHeap{
```

```
list < Node* > H;

list < Node* > :: iterator get_root(){

    list < Node* > :: iterator it, it_max;
    Node* vmax = newNode( -INF );

    for( it = H.begin(); it != H.end(); ++it )
        if( (*it) -> key > vmax -> key ){
            vmax = *it;
            it_max = it;
        }

    return it_max;
}

void delete_root( Node* tree, BinomialHeap& heap ){

    if( tree -> degree == 0 ){
        delete tree;
        return;
    }

    Node* temp = tree;

    tree -> child -> parent = NULL;
    heap.H.push_front( tree -> child );

    tree = tree -> child;
    while( tree -> sibling ){
        tree -> sibling -> parent = NULL;
        heap.H.push_front( tree -> sibling );
        tree = tree -> sibling;
    }
    delete temp;
}

void merge_tree( Node* tree1, Node* tree2 ){

    if( tree1 -> key < tree2 -> key )
        swap ( *tree1, *tree2 );

    tree2 -> sibling = tree1 -> child;
    tree2 -> parent = tree1;
    tree1 -> child = tree2;
    tree1 -> degree++;

}

void adjust(){

    if( H.size() <= 1 ) return;

    list < Node* > :: iterator prev;
    list < Node* > :: iterator curr;
    list < Node* > :: iterator next;
    list < Node* > :: iterator temp;

    prev = curr = H.begin();
    curr++;
    next = curr;
    next++;

    while( curr != H.end() ){

        while ( ( next == H.end() || (*next) -> degree > (*
            curr) -> degree ) && curr != H.end() && (*prev
            ) -> degree == (*curr) -> degree ){
```

```
                merge_tree( *curr, *prev );

                temp = prev;

                if( prev == H.begin() ){
                    prev++;
                    curr++;
                    if( next != H.end() )
                        next++;
                }
                else prev--;

                H.erase( temp );

            }

            prev++;
            if( curr != H.end() ) curr++;
            if( next != H.end() ) next++;
        }
    }
public:

    int top(){
        return (*get_root()) -> key;
    }

    void push( int val ){

        Node *tree = newNode( val );
        H.push_front( tree );
        adjust();
    }

    void heap_union( BinomialHeap& heap2){

        list < Node* > :: iterator it1 = H.begin();
        list < Node* > :: iterator it2 = heap2.H.begin();

        list < Node* > new_heap;

        while( it1 != H.end() && it2 != heap2.H.end() ){
            if( (*it1) -> degree <= (*it2) -> degree ){
                new_heap.push_back( *it1 );
                it1++;
            }
            else{
                new_heap.push_back( *it2 );
                it2++;
            }
        }

        while( it1 != H.end() ){
            new_heap.push_back( *it1 );
            it1++;
        }

        while( it2 != heap2.H.end() ){
            new_heap.push_back( *it2 );
            it2++;
        }

        heap2.H.clear();

        H = new_heap;
        adjust();
    }
    void pop(){

        list < Node* > :: iterator root = get_root();

        BinomialHeap new_heap;
        delete_root( (*root), new_heap );

        H.erase( root );

        heap_union( new_heap );

    }
};

int N, M;
BinomialHeap Heap[NMAX];

int main()
{
    fin >> N >> M;

    int task, h, x, h1, h2;
    for( int i = 1; i <= M; ++i ){
        fin >> task;

        if( task == 1 ){

            fin >> h >> x;
            Heap[h].push( x );

        }
        if( task == 2 ){

            fin >> h;
            fout << Heap[h].top() << '\n';
            Heap[h].pop();
        }
        if( task == 3 ){

            fin >> h1 >> h2;

            Heap[h1].heap_union( Heap[h2] );
        }
    }

    return 0;
}
```

bit.cpp

Description: wtf

20 lines

```
template<class T = int>
struct BIT {
  vector<T> bit;

  BIT(int n) : bit(n + 1, 0) {}

  void update(int p, T val) {
    for(; p < bit.size(); p += p & (-p)) bit[p] += val;
  }

  T query(int p) {
    T res;
    for(res = 0; p; p &= p - 1) res += bit[p];
    return res;
  }

  T query(int l, int r) {
    return query(bit, r) - query(bit, l - 1);
  }
};
```

cht.cpp

Description: Add lines of the form ax+b and query maximum. Lines should be sorted in increasing order of slope

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

32 lines

```
struct Hull {
  int nr = 0;
  vector<pll> v;

  void add(pll line) {
    line.se = -line.se;

    while(nr >= 2 && ccw(v[nr - 2], v[nr - 1], line) <= 0) --nr
        , v.pop_back();
    v.push_back(line);
    ++nr;
  }

  ll query(ll x) const {
    int l, r, mid;
    for(l = 0, r = nr - 1; l < r; ) {
      mid = (l + r) / 2;
      // care at overflow !!
      if(v[mid + 1].se - v[mid].se < x * (v[mid + 1].fi - v[mid
          ].fi)) l = mid + 1;
      else r = mid;
    }
    // while(p + 1 < nr && eval(v[p + 1], x) < eval(v[p], x))
        ++p;
    return eval(v[l], x);
  }

  ll eval(pll line, ll x) const {
    return line.fi * x - line.se;
  }

  ld ccw(const pll &a, const pll &b, const pll &c) const {
    return 1.0 * (b.fi - a.fi) * (c.se - a.se) - 1.0 * (c.fi -
        a.fi) * (b.se - a.se);
  }
};
```

convexhulltrick.cpp

Description: Add lines of the form ax+b and query maximum. Lines should be sorted in increasing order of slope

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

39 lines

```
template<class T = pll, class U = ll>
struct hull {
  struct frac {
    ll x, y;
    frac(ll _x, ll _y) : x(_x), y(_y) {
      if(y < 0) x = -x, y = -y;
    }
    bool operator <(const frac &other) const {
      return 1.0 * x * other.y < 1.0 * other.x * y;
    }
  };
  frac inter(T l1, T l2) { return { l2.se - l1.se, l1.fi - l2.
      fi }; }

  int nr = 0;
  vector<T> v;
  void add(T line) {
    // change signs for min
    if(!v.empty() && v.back().fi == line.fi) {
      if(v.back().se < line.se) v.back() = line;
      return;
    }
    while(nr >= 2 && inter(line, v[nr - 2]) < inter(v[nr - 1],
        v[nr - 2])) --nr, v.pop_back();
    v.push_back(line);
    ++nr;
  }
  U query(ll x) {
    int l, r, mid;
    for(l = 0, r = nr - 1; l < r; ) {
      mid = (l + r) / 2;
      if(inter(v[mid], v[mid + 1]) < frac(x, 1)) l = mid + 1;
      else r = mid;
    }
    // while(p + 1 < nr && eval(v[p + 1], x) < eval(v[p], x))
        ++p;
    return v[l];
  }
  ll eval(T line, ll x) {
    return line.fi * x + line.se;
  }
};
```

ConvexTree.h

Description: Container where you can add lines of the form a * x + b, and query maximum values at points x. Useful for dynamic programming. To change to minimum, either change the sign of all comparisons, the initialization of T and max to min, or just add lines of form (-a)*X + (-b) instead and negate the result.

Time: $\mathcal{O}(\log(kMax - kMin))$

<bits/stdc++.h> 50 lines

```
using int64 = int64_t;

struct Line {
  int a; int64 b;
  int64 Eval(int x) { return 1LL * a * x + b; }
};
const int64 kInf = 2e18; // Maximum abs(A * x + B)
const int kMin = -1e9, kMax = 1e9; // Bounds of query (x)

struct ConvexTree {
  struct Node { int l, r; Line line; };
  vector<Node> T = { Node{0, 0, {0, -kInf}} };
  int root = 0;

  int update(int node, int b, int e, Line upd) {
    if (node == 0) {
      T.push_back(Node{0, 0, upd});
      return T.size() - 1;
    }

    auto& cur = T[node].line;
    if (cur.Eval(b)>=upd.Eval(b) && cur.Eval(e)>=upd.Eval(e))
      return node;
    if (cur.Eval(b)<=upd.Eval(b) && cur.Eval(e)<=upd.Eval(e))
      return cur = upd, node;

    int m = (b + e) / 2;
    if (cur.Eval(b) < upd.Eval(b)) swap(cur, upd);
    if (cur.Eval(m) >= upd.Eval(m)) {
      int res = update(T[node].r, m + 1, e, upd);
      T[node].r = res; // DO NOT ATTEMPT TO OPTIMIZE
    } else {
      swap(cur, upd);
      int res = update(T[node].l, b, m, upd);
      T[node].l = res; // DO NOT ATTEMPT TO OPTIMIZE
    }
    return node;
  }
  void AddLine(Line l) { root = update(root, kMin, kMax, l); }

  int64 query(int node, int b, int e, int x) {
    int64 ans = T[node].line.Eval(x);
    if (node == 0) return ans;
    int m = (b + e) / 2;
    if (x <= m) ans = max(ans, query(T[node].l, b, m, x));
    if (x > m) ans = max(ans, query(T[node].r, m + 1, e, x));
    return ans;
  }
  int64 QueryMax(int x) { return query(root, kMin, kMax, x); }
};
```

dynamicCHT.cpp

Description: wtf

43 lines

```
const ll is_query = -(1LL << 62);

struct Line {
    ll m, b;
    mutable function<const Line *()> succ;

    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};

struct DynamicHull : public multiset<Line> {
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) *
             (y->m - x->m);
    }

    void insert_line(ll m, ll b) {
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y)
             ; };
        if (bad(y)) {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y))) erase(next(y))
             ;
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }

    ll eval(ll x) {
        auto l = *lower_bound((Line) {x, is_query});
        return l.m * x + l.b;
    }
};
```

FenwickTree.h

Description: Adds a value to a (half-open) range and computes the sum on a (half-open) range. Beware of overflows!

Time: Both operations are $\mathcal{O}(\log N)$.

27 lines

```
#define int long long
struct FenwickTree {
  int n;
  vector<int> T1, T2;

  FenwickTree(int n) : n(n), T1(n + 1, 0), T2(n + 1, 0) {}

  void Update(int b, int e, int val) {
    if (e != -1)
      return Update(e, -1, -val), Update(b, -1, +val);

    int c1 = val, c2 = val * (b - 1);
    for (int pos = b + 1; pos <= n; pos += (pos & -pos)) {
      T1[pos] += c1; T2[pos] += c2;
    }
  }

  int Query(int b, int e) {
    if (b != 0) return Query(0, e) - Query(0, b);

    int ans = 0;
    for (int pos = e; pos; pos -= (pos & -pos)) {
      ans += T1[pos] * (e - 1) - T2[pos];
    }
    return ans;
  }
};
```

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call FakeUpdate() before Init()).

Time: $\mathcal{O}\left(\log^2 N\right)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

"FenwickTree.h" 32 lines

```
struct Fenwick2D {
  vector<vector<int>> ys;
  vector<vector<int>> T;
  Fenwick2D(int n) : ys(n + 1) {}

  void FakeUpdate(int x, int y) {
    for (++x; x < (int)ys.size(); x += (x & -x))
      ys[x].push_back(y);
  }
  void Init() {
    for (auto& v : ys) {
      sort(v.begin(), v.end());
      T.emplace_back(v.size());
    }
  }
  int ind(int x, int y) {
    auto it = lower_bound(ys[x].begin(), ys[x].end(), y);
    return distance(ys[x].begin(), it);
  }
  void Update(int x, int y, int val) {
    for (++x; x < (int)ys.size(); x += (x & -x))
    for (int i = ind(x,y); i < (int)T[x].size(); i += (i & -i))
      trees[x][i] = trees[x][i] + val;
  }
  int Query(int x, int y) {
    int sum = 0;
    for (; x > 0; x -= (x & -x))
    for (int i = ind(x,y); i > 0; i -= (i & -i))
      sum = sum + T[x][i];
    return sum;
  }
};
```

implicitTreapsMaxValeriu.cpp

Description: None

Usage: ask Djok

<bits/stdc++.h> 140 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 200005;

int i, n, q, a[N], x, y, z;

struct node;
typedef node* ln;

struct node
{
    int pr;

    int v;
    int dp;

    int id,sz;

    ln l, r;

    node (int v=0) : pr(rand() * rand() * rand()),v(v),l(0),r
         (0) { upd(); }

    void upd()
    {
        dp = v;
        if (l) dp=max(dp,l->dp);
        if (r) dp=max(dp,r->dp);

        sz = 1;
        if (l) sz+=l->sz;
        id = sz;
        if (r) sz+=r->sz;
    }
};

ln root;

void split ( ln t, int x, ln &l, ln &r)
{
    l=r=0;
    if (!t) return;
    if (t->id <= x)
    {
        split(t->r, x - t->id, t->r, r);
        l = t;
    } else
    {
        split(t->l, x, l, t->l);
        r = t;
    }

    t->upd();
}

ln merge(ln l, ln r)
{
    if (!l || !r) return (l?l:r);

    if (l->pr > r->pr)
    {
        l->r = merge(l->r, r);
        l->upd();
        return l;
    } else
    {
        r->l = merge(l,r->l);
        r->upd();
        return r;
    }
}

void insert(int x, int p)
{
    ln l,r;
    split(root,p,l,r);
    root = merge(merge(l,new node(x)),r);
}

void erase(int p)
{
    ln l,r,t;
    split(root,p,l,r);
    split(r,1,r,t);
    root = merge(l,t);
}

int query(int x, int y)
{
    ln l,t,r;
    split(root, x, l, t);
    split(t, y-x+1, t, r);

    int m = t->dp;

    root = merge(merge(l,t),r);
    return m;
}

void show(ln t)
{
    if (!t) return;
    show(t->l);
    cout<<' '<<t->v;
    show(t->r);
}

int getPoz(int p)
{
    ln l,r,t;
    split(root,p,l,r);
    split(r,1,r,t);
    int ans = r->v;
    r = merge(r,t);
    root = merge(l, r);
    return ans;
}

int main() {
  srand(time(0));
  root = 0;
  scanf("%d %d", &n, &q);
  for(i = 0; i < n; ++i) scanf("%d", a + i), insert(a[i], i);
  while(q--) {
    scanf("%d %d %d", &x, &y, &z);
    if(x == 1) {
      printf("%d\n", query(y - 1, z - 1));
      continue;
    }

    --z; x = getPoz(z);
    erase(z);
    if(y == 1) {
      insert(x, n - 1);
    } else {
      insert(x, 0);
    }
  }
  return 0;
}
```

LazySegmentTree.h

Description: wtf

38 lines

```
struct ST {
  int n;
  vector<int> st, lazy;

  ST(int n) : n(n), st(4 * n), lazy(4 * n) {}

  void push(int node) {
    st[2 * node] += lazy[node];
    lazy[2 * node] += lazy[node];
    st[2 * node + 1] += lazy[node];
    lazy[2 * node + 1] += lazy[node];
    lazy[node] = 0;
  }

  void update(int node, int l, int r, int a, int b, int val) {
    if(a <= l && r <= b) { st[node] += val; lazy[node] += val;
        return; }
    push(node);

    int mid = (l + r) / 2;
    if(a <= mid) update(2 * node, l, mid, a, b, val);
    if(mid + 1 <= b) update(2 * node + 1, mid + 1, r, a, b, val
        );

    st[node] = min(st[2 * node], st[2 * node + 1]);
  }

  int query(int node, int l, int r, int a, int b) {
    if(a <= l && r <= b) return st[node];
    push(node);

    int mid = (l + r) / 2;
    int v1 = (a <= mid ? query(2 * node, l, mid, a, b) : INF);
    int v2 = (mid + 1 <= b ? query(2 * node + 1, mid + 1, r, a,
         b) : INF);
    return min(v1, v2);
  }

  void update(int a, int b, int val) { update(1, 1, n, a, b,
      val); }
  int query(int a, int b) { return query(1, 1, n, a, b); }
};
```

OrderStatisticTree.h

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

16 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
}
```

pairingHeap.cpp

Description: wtf

131 lines

```
const int NMAX = 101;
const int INF = 2000000001;

ifstream fin("mergeheap.in");
ofstream fout("mergeheap.out");

struct Node{
    int key;
    Node *child, *sibling;

    Node( int x ) : key( x ), child( NULL ), sibling( NULL ) {}
};

class PairingHeap{

    Node *root;

    Node* merge_heap( Node* H1, Node* H2 ){

        if( H1 == NULL ){
            H1 = H2;
            return H1;
        }
        if( H2 == NULL ) return H1;

        if( H1 -> key < H2 -> key )
            swap( H1, H2 );

        H2 -> sibling = H1 -> child;
        H1 -> child = H2;

        return H1;
    }

    Node* two_pass_merge( Node *_Node ){

        if( _Node == NULL || _Node -> sibling == NULL )
            return _Node;

        Node *heap_1, *heap_2, *next_pair;

        heap_1 = _Node;
        heap_2 = _Node -> sibling;
        next_pair = _Node -> sibling -> sibling;

        heap_1 -> sibling = heap_2 -> sibling = NULL;

        return merge_heap( merge_heap( heap_1, heap_2 ),
            two_pass_merge( next_pair ) );
    }
public:

    PairingHeap() : root( NULL ) {}

    PairingHeap( int _key ){
        root = new Node( _key );
    }
    PairingHeap( Node* _Node ) : root( _Node ) {}

    int top(){
        return root -> key;
    }

    void merge_heap( PairingHeap H ){

        if( root == NULL ){
            root = H.root;
            return;
        }
        if( H.root == NULL ) return;

        if( root -> key < H.root -> key )
            swap( root, H.root );

        H.root -> sibling = root -> child;
        root -> child = H.root;
        H.root = NULL;
    }

    void push( int _key ){
        merge_heap( PairingHeap( _key ) );
    }

    void pop(){

        Node* temp = root;
        root = two_pass_merge( root -> child );

        delete temp;
    }

    void heap_union( PairingHeap &H ){
        merge_heap( H );
        H.root = NULL;
    }
};

int N, M;

PairingHeap Heap[NMAX];

int main()
{

    fin >> N >> M;

    int task, h, x, h1, h2;
    for( int i = 1; i <= M; ++i ){

        fin >> task;

        if( task == 1 ){
            fin >> h >> x;

            Heap[h].push( x );
        }
        if( task == 2 ){
            fin >> h;

            fout << Heap[h].top() << '\n';
            Heap[h].pop();
        }
        if( task == 3 ){
            fin >> h1 >> h2;

            Heap[h1].heap_union( Heap[h2] );
        }
    }

    return 0;
}
```

RMQ.h

Description: wtf

18 lines

```
struct RMQ {
  vector<vector<int>> rmq;

  void build(const vector<int> &vec) {
    rmq.push_back(vec);

    for(int i = 1; (1 << i) <= vec.size(); ++i) {
      rmq.push_back(vector<int>(vec.size()));
      for(int j = 0; j + (1 << i) - 1 < vec.size(); ++j)
        rmq[i][j] = gcd(rmq[i - 1][j], rmq[i - 1][j + (1 << (i
             - 1))]);
    }
  }

  int query(int l, int r) {
    int d = 31 - __builtin_clz(r - l + 1);
    return gcd(rmq[d][l], rmq[d][r - (1 << d) + 1]);
  }
};
```

SegmentTree.h

Description: wtf

19 lines

```
struct SegTree {
  int n;
  vector<int> st;

  SegTree(int n) : n(n), st(2 * n) {}

  void update(int p, int val) {
    for(st[p += n] = val; p > 1; p >>= 1) st[p >> 1] = max(st[p
         ], st[p ^ 1]);
  }

  int query(int l, int r) {
    int res = 0;
    for(l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if(l & 1) res = max(res, st[l++]);
      if(r & 1) res = max(res, st[--r]);
    }
    return res;
  }
};
```

slopeTrick.cpp

Description: Given an array a, on operation means increase or decrease an element by one What is the minimum number of operations to make it strictly increasing? Remove line "a -= i" for non-decreasing

28 lines

```
int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int n, a;

  cin >> n;

  priority_queue<int> q;

  ll ans = 0;
  for(int i = 0; i < n; ++i) {
    cin >> a;
    a -= i;

    q.push(a);
    q.push(a);

    ans += q.top() - a;

    q.pop();
  }

  cout << ans << '\n';

  return 0;
}
```

Treap.h

Description: wtf

69 lines

```
struct Treap {
  int key, pri, cnt, mn, mx, s;
  Treap *l, *r;

  Treap(int key) : key(key), pri(rand()) {
    cnt = s = 1;
    mn = mx = key;
    l = r = nullptr;
  }
};
using PTreap = Treap*;

void update(PTreap node) {
  // TODO: update node considering children are correct
}

void split(PTreap node, int key, PTreap &l, PTreap &r) {
  if(!node) return void(l = r = nullptr);

  if(key < node->key) split(node->l, key, l, node->l), r = node
       ;
  else split(node->r, key, node->r, r), l = node;

  update(node);
}

void merge(PTreap &node, PTreap l, PTreap r) {
  if(!l || !r) return void(node = (l ? l : r));

  if(l->pri < r->pri) merge(r->l, l, r->l), node = r;
  else merge(l->r, l->r, r), node = l;

  update(node);
}

bool addIfExists(PTreap node, int key) {
  if(!node) return false;
  if(node->key == key) return ++node->cnt, update(node), true;

  auto res = addIfExists(key < node->key ? node->l : node->r,
       key);
  update(node);
  return res;
}

void add(PTreap &node, PTreap item) {
  if(!node) return void(node = item);
  if(item->pri > node->pri) split(node, item->key, item->l,
       item->r), node = item;
  else add(item->key < node->key ? node->l : node->r, item);

  update(node);
}

void erase(PTreap &node, int key) {
  if(!node) return;

  if(node->key == key) {
    --node->cnt;
    if(!node->cnt) merge(node, node->l, node->r);
  } else erase(key < node->key ? node->l : node->r, key);

  if(node) update(node);
}

void print(PTreap node, string indent = "") {
  if(!node) return;
  cout << indent << ' ' << node->key << ' ' << node->cnt << '\n
       ';
  print(node->l, indent + " ");
  print(node->r, indent + " ");
}
```

UnionFind.h

Description: wtf

20 lines

```
struct UnionFind {
  vector<int> fth, sz;

  UnionFind(int n) {
    fth.assign(n, -1);
    sz.assign(n, 1);
  }

  int root(int x) { return fth[x] == -1 ? x : fth[x] = root(fth
       [x]); }

  bool join(int a, int b) {
    a = root(a);
    b = root(b);
    if(a == b) return false;

    if(sz[a] < sz[b]) fth[a] = b, sz[b] += sz[a];
    else fth[b] = a, sz[a] += sz[b];
    return true;
  }
};
```

Numerical (4)

BerlekampMassey.h

Description: Recovers any n-order linear recurrence relation from the first 2*n terms of the recurrence. Very useful for guessing linear recurrences after brute-force / backtracking the first terms. Should work on any field. Numerical stability for floating-point calculations is not guaranteed.

Usage: BerlekampMassey({0, 1, 1, 3, 5, 11}) => {1, 2}

<bits/stdc++.h>, "ModOps.h" 29 lines

```
vector<ModInt> BerlekampMassey(vector<ModInt> s) {
  int n = s.size();
  vector<ModInt> C(n, 0), B(n, 0);
  C[0] = B[0] = 1;

  ModInt b = 1; int L = 0;
  for (int i = 0, m = 1; i < n; ++i) {
    ModInt d = s[i];
    for (int j = 1; j <= L; ++j)
      d = d + C[j] * s[i - j];

    if (d.get() == 0) { ++m; continue; }

    auto T = C; ModInt coef = d * inv(b);
    for (int j = m; j < n; ++j)
      C[j] = C[j] - coef * B[j - m];

    if (2 * L > i) { ++m; continue; }

    L = i + 1 - L; B = T; b = d; m = 1;
  }

  C.resize(L + 1); C.erase(C.begin());
  for (auto& x : C) x = ModInt(0) - x;

  return C;
}
```

Polynomial.h

Description: Different operations on polynomials. Should work on any field.

<bits/stdc++.h> 114 lines

```
using TElem = double;
using Poly = vector<TElem>;

TElem Eval(const Poly& P, TElem x) {
  TElem val = 0;
  for (int i = (int)P.size() - 1; i >= 0; --i)
    val = val * x + P[i];
  return val;
}

// Differentiation
Poly Diff(Poly P) {
  for (int i = 1; i < (int)P.size(); ++i)
    P[i - 1] = i * P[i];
  P.pop_back();
  return P;
}

// Integration
Poly Integrate(Poly p) {
  P.push_back(0);
  for (int i = (int)P.size() - 2; i >= 0; --i)
    P[i + 1] = P[i] / (i + 1);
  P[0] = 0;
  return P;
}

// Division by (X − x0)
Poly DivRoot(Poly P, TElem x0) {
  int n = P.size();
  TElem a = P.back(), b; P.back() = 0;
  for (int i = n--; i--; )
    b = P[i], P[i] = P[i + 1] * x0 + a, a = b;
  P.pop_back();
  return P;
}

// Multiplication modulo X^sz
Poly Multiply(Poly A, Poly B, int sz) {
  static FFTSolver fft;
  A.resize(sz, 0); B.resize(sz, 0);
  auto R = fft.Multiply(A, B);
  R.resize(sz, 0);
  return r;
}

// Scalar multiplication
Poly Scale(Poly P, TElem s) {
  for (auto& x : P)
    x = x * s;
  return P;
}

// Addition modulo X^sz
Poly Add(Poly A, Poly B, int sz) {
  A.resize(sz, 0); B.resize(sz, 0);
  for (int i = 0; i < sz; ++i)
    A[i] = A[i] + B[i];
  return A;
}

// ************************************************
// For Invert, Sqrt, size of argument should be 2^k
// ************************************************

Poly inv_step(Poly res, Poly P, int n) {
  auto res_sq = Multiply(res, res, n);
  auto sub = Multiply(res_sq, P, n);
  res = Add(Scale(res, 2), Scale(sub, -1), n);
  return res;
}

// Inverse modulo X^sz
// EXISTS ONLY WHEN P[0] IS INVERTIBLE
Poly Invert(Poly P) {
  assert(P[0].Get() == 1);
  Poly res(1, 1);          // i.e., P[0]^(−1)

  int n = P.size();
  for (int step = 2; step <= n; step *= 2) {
    res = inv_step(res, P, step);
  }

  // Optional, but highly encouraged
  auto check = Multiply(res, P, n);
  for (int i = 0; i < n; ++i) {
    assert(check[i].Get() == (i == 0));
  }
  return res;
};

// Square root modulo X^sz
// EXISTS ONLY WHEN P[0] HAS SQUARE ROOT
Poly Sqrt(Poly P) {
  assert(P[0].Get() == 1);
  Poly res(1, 1);          // i.e., P[0]^(−1)
  Poly inv(1, 1);          // i.e., P[0]^(1/2)

  int n = P.size();
  for (int step = 2; step <= n; step *= 2) {
    auto now = inv_step(inv, res, step);
    now = Multiply(P, move(now), step);
    res = Add(res, now, step);
    res = Scale(res, (kMod + 1) / 2);
    inv = inv_step(inv, res, step);
  }

  // Optional, but highly encouraged
  auto check = Multiply(res, res, n);
  for (int i = 0; i < n; ++i) {
    assert(check[i].Get() == P[i].Get());
  }
  return res;
}
```

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `Poly p = {2, -3, 1} // x^2 - 3x + 2 = 0`
`auto roots = GetRoots(p, -1e18, 1e18); // {1, 2}`

<bits/stdc++.h>, "Polynomial.h" 26 lines

```
vector<double> GetRoots(Poly p, double xmin, double xmax) {
  if (p.size() == 2) { return {-p.front() / p.back()}; }
  else {
    Poly d = Diff(p);
    vector<double> dr = GetRoots(d, xmin, xmax);
    dr.push_back(xmin - 1);
    dr.push_back(xmax + 1);
    sort(dr.begin(), dr.end());

    vector<double> roots;
    for (auto i = dr.begin(), j = i++; i != dr.end(); j = i++){
      double lo = *j, hi = *i, mid, f;
      bool sign = Eval(p, lo) > 0;
      if (sign ^ (Eval(p, hi) > 0)) {
        // for (int it = 0; it < 60; ++it) {
        while (hi - lo > 1e-8) {
          mid = (lo + hi) / 2, f = Eval(p, mid);
          if ((f <= 0) ^ sign) lo = mid;
          else hi = mid;
        }
        roots.push_back((lo + hi) / 2);
      }
    }
    return roots;
  }
}
```

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

<bits/stdc++.h>, "Polynomial.h" 15 lines

```
Poly Interpolate(vector<TElem> x, vector<TElem> y) {
  int n = x.size();
  Poly res(n), temp(n);
  for (int k = 0; k < n; ++k)
    for (int i = k + 1; i < n; ++i)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  TElem last = 0; temp[0] = 1;
  for (int k = 0; k < n; ++k)
  for (int i = 0; i < n; ++i) {
    res[i] = res[i] + y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] = temp[i] - last * x[k];
  }
  return res;
}
```

LinearRecurrence.h

Description: Generates the k-th term of a n-th order linear recurrence given the first n terms and the recurrence relation. Faster than matrix multiplication. Useful to use along with Berlekamp Massey.
Usage: `LinearRec<double>({0, 1}, {1, 1}).Get(k) gives k-th Fibonacci number (0-indexed)`
Time: $\mathcal{O}(n^2 log(k))$ per query

<bits/stdc++.h> 43 lines

```
template<typename T>
struct LinearRec {
  using Poly = vector<T>;
  int n; Poly first, trans;

  // Recurrence is S[i] = sum(S[i-j-1] * trans[j])
  // with S[0..(n-1)] = first
  LinearRec(const Poly &first, const Poly &trans) :
    n(first.size()), first(first), trans(trans) {}

  Poly combine(Poly a, Poly b) {
    Poly res(n * 2 + 1, 0);
    // You can apply constant optimization here to get a
    // ~10x speedup
    for (int i = 0; i <= n; ++i)
      for (int j = 0; j <= n; ++j)
        res[i + j] = res[i + j] + a[i] * b[j];

    for (int i = 2 * n; i > n; --i)
      for (int j = 0; j < n; ++j)
        res[i - 1 - j] = res[i - 1 - j] + res[i] * trans[j];

    res.resize(n + 1);
    return res;
  }

  // Consider caching the powers for multiple queries
  T Get(int k) {
    Poly r(n + 1, 0), b(r);
    r[0] = 1; b[1] = 1;

    for (++k; k; k /= 2) {
      if (k % 2)
        r = combine(r, b);
      b = combine(b, b);
    }

    T res = 0;
    for (int i = 0; i < n; ++i)
      res = res + r[i + 1] * first[i];
    return res;
  }
};
```

FST.h

Description: Fast Subset transform. Useful for performing the following convolution: R[a op b] += A[a] * B[b], where op is either of AND, OR, XOR. P has to have size $N = 2^n$, for some n.
Time: $\mathcal{O}(N \log N)$

<bits/stdc++.h> 16 lines

```
vector<int> Transform(vector<int> P, bool inv) {
  int n = P.size();
  for (int step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) {
      for (int j = i; j < i + step; ++j) {
        int u = P[j], v = P[j + step];
        tie(P[j], P[j + step]) =
        inv ? make_pair(v - u, u) : make_pair(v, u + v); // AND
        inv ? make_pair(v, u - v) : make_pair(u + v, u); // OR
        make_pair(u + v, u - v);                          // XOR
      }
    }
  }
  // if (inv) for (auto& x : P) x /= n; // XOR only
  return P;
}
```

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: `A = {{1,-1}, {-1,1}, {-1,-2}};`
`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.

68 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
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define sz(x) (int)(x).size()

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;
  }
};
```

SolveLinear.h

Description: Solves $M * x = b$. If there are multiple solutions, returns a solution which has all free variables set to 0. To compute rank, count the number of values in pivot. vector which are not -1. For inverse modulo 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: $\mathcal{O}\left(vars^2 cons\right)$ 79 lines

```
// Transforms a matrix into its row echelon form
// Returns a vector of pivots for each variable
// vars is the number of variables to do echelon for
vector<int> ToRowEchelon(vector<vector<double>> &M, int vars) {
  int n = M.size(), m = M[0].size();
  vector<int> pivots(vars, -1);

  int cur = 0;
  for (int var = 0; var < vars; ++var) {
    if (cur >= n) break;

    for (int con = cur + 1; con < n; ++con)
      if (sgn(M[con][var]) != 0)
        swap(M[con], M[cur]);

    if (sgn(M[cur][var]) != 0) {
      pivots[var] = cur;
      auto aux = M[cur][var];

      for (int i = 0; i < m; ++i)
        M[cur][i] = M[cur][i] / aux;

      for (int con = 0; con < n; ++con) {
        if (con != cur) {
          auto mul = M[con][var];
          for (int i = 0; i < m; ++i) {
            M[con][i] = M[con][i] - mul * M[cur][i];
          }
        }
      }
      ++cur;
    }
  }

  return pivots;
}

// Computes the inverse of a nxn square matrix.
// Returns true if successful
bool Invert(vector<vector<double>> &M) {
  int n = M.size();
  for (int i = 0; i < n; ++i) {
    M[i].resize(2 * n, 0); M[i][n + i] = 1;
  }

  auto pivs = ToRowEchelon(M, n);
  for (auto x : pivs) if (x == -1) return false;

  for (int i = 0; i < n; ++i)
    M[i].erase(M[i].begin(), M[i].begin() + n);
  return true;
}

// Returns the solution of a system
// Will change matrix
// Throws 5 if inconsistent
vector<double> SolveSystem(vector<vector<double>> &M,
                           vector<double>& b) {
  int vars = M[0].size();
  for (int i = 0; i < (int)M.size(); ++i)
    M[i].push_back(b[i]);

  auto pivs = ToRowEchelon(M, vars);
  vector<double> solution(vars);
  for (int i = 0; i < vars; ++i) {
    solution[i] = (pivs[i] == -1) ? 0 : M[pivs[i]][vars];
  }

  // Check feasible (optional)
  for (int i = 0; i < (int)M.size(); ++i) {
    double check = 0;
    for (int j = 0; j < vars; ++j)
      check = check + M[i][j] * solution[j];
    if (sgn(check - M[i][vars]) != 0)
      throw 5;
  }

  return solution;
}
```

Number theory (5)

5.1 General

mathValeriu.h

Description: None
Usage: ask Djok 118 lines

```
bool isPrime(int x) {
  if(x < 2) return 0;
  if(x == 2) return 1;
  if(x % 2 == 0) return 0;
  for(int i = 3; i * i <= x; i += 2)
    if(x % i == 0) return 0;
  return 1;
}

int mul(int a, int b) {
  return (long long)a * b % MOD;
}

int add(int a, int b) {
  a += b;
  if(a >= MOD) return a - MOD;
  return a;
}

int getPw(int a, int b) {
  int ans = 1;
  for(; b > 0; b /= 2) {
    if(b & 1) ans = mul(ans, a);
    a = mul(a, a);
  }
  return ans;
}

long long modInv(long long a, long long m) {
  if(a == 1) return 1;
  return (1 - modInv(m % a, a) * m) / a + m;
}

long long CRT(vector<long long> &r, vector<long long> &p) {
  long long ans = r[0] % p[0], prod = p[0];
  for(int i = 1; i < r.size(); ++i) {
    long long coef = ((r[i] - (ans % p[i]) + p[i]) % p[i]) *
        modInv(prod % p[i], p[i]) % p[i];
    ans += coef * prod;
    prod *= p[i];
  }
  return ans;
}

long long getPhi(long long n) {
  long long ans = n - 1;
  for(int i = 2; i * i <= n; ++i) {
    if(n % i) continue;
    while(n % i == 0) n /= i;
    ans -= ans / i;
  }
  if(n > 1) ans -= ans / n;
  return ans;
}

// fact is a vector with prime divisors of N - 1 (N here is
     modulo) and N is prime
// the idea is that if N is prime, then N-1 is phi(N), which
     means the cycle has length N - 1
// now, lets try to see if X is a generator
// we know that if x ^ phi(N) == 1 then x ^ 2*phi(N) is also ==
      1, and here we get the idea
// if for some divisor of phi(N), x ^ div == 1, then obviously
      X is not a generator
// because the cycle is not of length N
// good luck to understand this after one year :)
bool isGenerator(int x, int n) {
  if(cmmdc(x, n) != 1) return 0;

  for(auto it : fact)
    if(Pow(x, (n - 1) / it, n) == 1)
      return 0;
  return 1;
}

// Lucas Theorem
// calc COMB(N, R) if N and R is VERY VERY BIG and MOD is PRIME
r -= 2; n += m - 2;
while(r > 0 || n > 0) {
  ans = (1LL * ans * comb(n % MOD, r % MOD)) % MOD;
  n /= MOD; r /= MOD;
}

// GAUSS FOR F2 space
// SZ is the size of basis
void gauss(int mask) {
  for(int i = 0; i < n; ++i) {
    if(!(mask & (1 << i))) continue;
    if(!basis[i]) {
      basis[i] = mask;
      ++sz;
      break;
    }
    mask ^= basis[i];
  }
}

// if A is a permutation of B, then A == B mod 9
```

```
bool isSquare(int x) {
  int a = sqrt(x) + 0.5;
  return a * a == x;
}

int getDiscreteLog(int a, int b, int m) {
  if(b == 1) return 0;
  int n = sqrt(m) + 1;
  int an = 1;
  for(int i = 0; i < n; ++i) an = (an * a) % m;
  unordered_map<int, int> vals;
  for(int i = 1, cur = an; i <= n; ++i) {
    if(!vals.count(cur)) vals[cur] = i;
    cur = (cur * an) % m;
  }
  for(int i = 0, cur = b; i <= n; ++i) {
    if(vals.count(cur)) {
      int ans = vals[cur] * n - i;
      return ans;
    }
    cur = (cur * a) % m;
  }
  return -1;
}
```

nrPentagonaleValeriu.h

Description: None
Usage: ask Djok
<bits/stdc++.h> 20 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int MOD = 999123;
const int N = 500005;

int i, j, p[N];

int main() {
  p[0] = 1;
  for(i = 1; i < N; ++i) {
    for(j = 1; j * (3 * j - 1) / 2 <= i; ++j)
      if(j & 1) p[i] = (p[i] + p[i - j * (3 * j - 1) / 2]) %
          MOD;
      else p[i] = (p[i] - p[i - j * (3 * j - 1) / 2] + MOD) %
          MOD;
    for(j = 1; j * (3 * j + 1) / 2 <= i; ++j)
      if(j & 1) p[i] = (p[i] + p[i - j * (3 * j + 1) / 2]) %
          MOD;
      else p[i] = (p[i] - p[i - j * (3 * j + 1) / 2] + MOD) %
          MOD;
  }
  return 0;
}
```

sieve.cpp

Description: wtf
14 lines

```
bool isPrime[VMAX];
vector<int> primes;

void linearSieve(int n) {
  for(int i = 2; i <= n; ++i) isPrime[i] = true;
  for(int i = 2; i <= n; ++i) {
    if(isPrime[i]) primes.push_back(i);
    for(auto p : primes) {
      if(i * p > n) break;
      isPrime[p * i] = false;
      if(i % p == 0) break;
    }
  }
}
```

5.2 Modular arithmetic

ModInverse.h

Description: Pre-computation of modular inverses. Assumes lim < kMod and that kMod is a prime.
"ModOps.h" 7 lines

```
vector<ModInt> ComputeInverses(int lim) {
  vector<ModInt> inv(lim + 1); inv[1] = 1;
  for (int i = 2; i <= lim; ++i) {
    inv[i] = ModInt(0) - ModInt(kMod / i) * inv[kMod % i];
  }
  return inv;
}
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) = $\sum_{i=0}^{to-1}(ki+c)\%m$. divsum is similar but for floored division.
Time: $\log(m)$, with a large constant.
21 lines

```
using ull = unsigned long long;
using ll = long long;

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) {
    ull to2 = (to * k + c) / m;
    res += to * to2;
    res -= DivSum(to2, m-1 - c, m, k) + to2;
  }
  return res;
}

ll ModSum(ull to, ll c, ll k, ll m) {
  c %= m; if (c < 0) c += m;
  k %= m; if (k < 0) k += m;
  return to * c + k * SumSq(to) - m * DivSum(to, c, k, m);
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots.
Time: $\mathcal{O}(\log^2 p)$ worst case, often $\mathcal{O}(\log p)$
"ModPow.h" 30 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);
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1;
  int r = 0;
  while (s % 2 == 0)
    ++r, s /= 2;
  ll n = 2; // find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p);
  ll g = modpow(n, s, p);
  for (;;) {
    ll t = b;
    int m = 0;
    for (; m < r; ++m) {
      if (t == 1) break;
      t = t * t % p;
    }
    if (m == 0) return x;
    ll gs = modpow(g, 1 << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
    r = m;
  }
}
```

5.3 Number theoretic transform

NTT.h

Description: wtf
78 lines

```
template<int P>
struct NTT {
  int root, maxBase;
  std::vector<int> rev, roots{0, 1};

  int power(int base, int e) {
    int res;
    for (res = 1; e > 0; e >>= 1) {
      if (e % 2 == 1) res = 1LL * res * base % P;
      base = 1LL * base * base % P;
    }
    return res;
  }

  void init() {
    for(maxBase = 0; !((P - 1) >> maxBase); ++maxBase);

    for(int root = 3; ; ++root)
      if(power(x, (P - 1) / 2) != 1) {
        return;
      }
  }

  void fft(std::vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
      int k = __builtin_ctz(n) - 1;
      rev.resize(n);
      for (int i = 0; i < n; ++i)
        rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
    }
    for (int i = 0; i < n; ++i)
      if (rev[i] < i) std::swap(a[i], a[rev[i]]);

    if (int(roots.size()) < n) {
      int k = __builtin_ctz(roots.size());
      roots.resize(n);
      while ((1 << k) < n) {
        int e = power(root, (P - 1) >> (k + 1));
        for (int i = 1 << (k - 1); i < (1 << k); ++i) {
          roots[2 * i] = roots[i];
          roots[2 * i + 1] = 1LL * roots[i] * e % P;
        }
        ++k;
      }
    }
    for (int k = 1; k < n; k *= 2) {
      for (int i = 0; i < n; i += 2 * k) {
        for (int j = 0; j < k; ++j) {
          int num = 1LL * a[i + j + k] * roots[k + j] % P;
          a[i + j + k] = (a[i + j] - num + P) % P;
          a[i + j] = (a[i + j] + num) % P;
        }
      }
    }
  }
  void ifft(std::vector<int> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    fft(a);
    int inv = power(n, P - 2);
    for (int i = 0; i < n; ++i)
      a[i] = 1LL * a[i] * inv % P;
  }

  std::vector<int> multiply(std::vector<int> a, std::vector<int
       > b) {
    int sz = 1, tot = a.size() + b.size() - 1;
    while (sz < tot) sz *= 2;
    a.resize(sz);
    b.resize(sz);
    fft(a);
    fft(b);
    for (int i = 0; i < sz; ++i) a[i] = 1LL * a[i] * b[i] % P;
    ifft(a);
    a.resize(tot);
    return a;
  }
};
```

5.4 Fast Fourier Transform

fftValeriu.h

Description: wtf

248 lines

```
struct ftype {
  double a, b;

  ftype(double a = 0, double b = 0) : a(a), b(b) {}
  ftype conj() { return ftype(a, -b); }
  friend ftype operator +(const ftype &x, const ftype &y) {
       return ftype(x.a + y.a, x.b + y.b); }
  friend ftype operator -(const ftype &x, const ftype &y) {
       return ftype(x.a - y.a, x.b - y.b); }
  friend ftype operator *(const ftype &x, const ftype &y) {
       return ftype(x.a * y.a - x.b * y.b, x.a * y.b + x.b * y.
       a); }
  friend ftype operator /(const ftype &x, int y) { return ftype
       (x.a / y, x.b / y); }
};

const double PI = acos(-1);

ftype polar(double ang) { return ftype(cos(ang), sin(ang)); }

int rv(int x, int sz) {
  int ans = 0;
  for(int i = 0; i < sz; ++i)
    if(x & (1 << i)) ans |= (1 << (sz - 1 - i));

  return ans;
}

vector<ftype> fft(vector<ftype> p, bool rev = false) {
  int i, sz, n = p.size();
  for(sz = 0; (1 << sz) < n; ++sz);

  for(int i = 0; i < n; ++i)
    if(i < rv(i, sz)) swap(p[i], p[rv(i, sz)]);

  for(int len = 2; len <= n; len <<= 1) {
    ftype wlen = polar((rev ? -1 : 1) * 2 * PI / len);

    for(int i = 0; i < n; i += len) {
      ftype w = 1;
      for(int j = 0; j < len / 2; ++j) {
        ftype u = p[i + j] + w * p[i + j + len / 2];
        ftype v = p[i + j] - w * p[i + j + len / 2];
        p[i + j] = u;
        p[i + j + len / 2] = v;
        w = w * wlen;
      }
    }
  }

  if(rev) {
    for(auto &val : p) val.a /= p.size(), val.b /= p.size();
  }

  return p;
}

vector<int> multiply(const vector<int> &a, const vector<int> &b
    ) {
  int sz = 2 * max(a.size(), b.size());
  while(__builtin_popcount(sz) != 1) ++sz;

  vector<ftype> na, nc;

  na.resize(sz);
  for(int i = 0; i < a.size(); ++i) na[i].a = a[i];
  for(int i = 0; i < b.size(); ++i) na[i].b = b[i];

  auto r = fft(na);

  for(int i = 0; i < r.size(); ++i) {
    ftype x = r[i];
    ftype y = r[i == 0 ? i : r.size() - i].conj();
    ftype ai = (x + y) / 2;
    ftype bi = (x - y) / 2 * ftype(0, -1);
    nc.push_back(ai * bi);
  }

  auto vc = fft(nc, true);

  vector<int> c;
  for(auto val : vc) c.push_back(round(val.a));

  return c;
}

// Tourist FFT
namespace fft {
  typedef double dbl;

  struct num {
    dbl x, y;
    num() { x = y = 0; }
    num(dbl x, dbl y) : x(x), y(y) { }
  };

  inline num operator+(num a, num b) { return num(a.x + b.x, a.
       y + b.y); }
  inline num operator-(num a, num b) { return num(a.x - b.x, a.
       y - b.y); }
  inline num operator*(num a, num b) { return num(a.x * b.x - a
       .y * b.y, a.x * b.y + a.y * b.x); }
  inline num conj(num a) { return num(a.x, -a.y); }

  int base = 1;
  vector<num> roots = {{0, 0}, {1, 0}};
  vector<int> rev = {0, 1};

  const dbl PI = acosl(-1.0);

  void ensure_base(int nbase) {
    if (nbase <= base) {
      return;
    }
    rev.resize(1 << nbase);
    for (int i = 0; i < (1 << nbase); i++) {
      rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
    }
    roots.resize(1 << nbase);
    while (base < nbase) {
      dbl angle = 2 * PI / (1 << (base + 1));
//        num z(cos(angle), sin(angle));
      for (int i = 1 << (base - 1); i < (1 << base); i++) {
        roots[i << 1] = roots[i];
//          roots[(i << 1) + 1] = roots[i] * z;
        dbl angle_i = angle * (2 * i + 1 - (1 << base));
        roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
      }
      base++;
    }
  }

  void fft(vector<num> &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 = base - 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++) {
          num z = a[i + j + k] * roots[j + k];
          a[i + j + k] = a[i + j] - z;
          a[i + j] = a[i + j] + z;
        }
      }
    }
  }

  vector<num> fa, fb;

  vector<int> multiply(vector<int> &a, vector<int> &b) {
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while ((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if (sz > (int) fa.size()) {
      fa.resize(sz);
    }
    for (int i = 0; i < sz; i++) {
      int x = (i < (int) a.size() ? a[i] : 0);
      int y = (i < (int) b.size() ? b[i] : 0);
      fa[i] = num(x, y);
    }
    fft(fa, sz);
    num r(0, -0.25 / sz);
    for (int i = 0; i <= (sz >> 1); i++) {
      int j = (sz - i) & (sz - 1);
      num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
      if (i != j) {
        fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
      }
      fa[i] = z;
    }
    fft(fa, sz);
    vector<int> res(need);
    for (int i = 0; i < need; i++) {
      res[i] = fa[i].x + 0.5;
    }
    return res;
  }

  vector<int> multiply_mod(vector<int> &a, vector<int> &b, int
      m, int eq = 0) {
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while ((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if (sz > (int) fa.size()) {
      fa.resize(sz);
    }
    for (int i = 0; i < (int) a.size(); i++) {
      int x = (a[i] % m + m) % m;
      fa[i] = num(x & ((1 << 15) - 1), x >> 15);
    }
    fill(fa.begin() + a.size(), fa.begin() + sz, num {0, 0});
    fft(fa, sz);
    if (sz > (int) fb.size()) {
      fb.resize(sz);
    }
    if (eq) {
      copy(fa.begin(), fa.begin() + sz, fb.begin());
    } else {
      for (int i = 0; i < (int) b.size(); i++) {
        int x = (b[i] % m + m) % m;
        fb[i] = num(x & ((1 << 15) - 1), x >> 15);
      }
      fill(fb.begin() + b.size(), fb.begin() + sz, num {0, 0});
      fft(fb, sz);
    }
    dbl ratio = 0.25 / sz;
    num r2(0, -1);
    num r3(ratio, 0);
    num r4(0, -ratio);
    num r5(0, 1);
    for (int i = 0; i <= (sz >> 1); i++) {
      int j = (sz - i) & (sz - 1);
      num a1 = (fa[i] + conj(fa[j]));
      num a2 = (fa[i] - conj(fa[j])) * r2;
      num b1 = (fb[i] + conj(fb[j])) * r3;
      num b2 = (fb[i] - conj(fb[j])) * r4;
      if (i != j) {
        num c1 = (fa[j] + conj(fa[i]));
        num c2 = (fa[j] - conj(fa[i])) * r2;
        num d1 = (fb[j] + conj(fb[i])) * r3;
        num d2 = (fb[j] - conj(fb[i])) * r4;
        fa[i] = c1 * d1 + c2 * d2 * r5;
        fb[i] = c1 * d2 + c2 * d1;
      }
      fa[j] = a1 * b1 + a2 * b2 * r5;
      fb[j] = a1 * b2 + a2 * b1;
    }
    fft(fa, sz);
    fft(fb, sz);
    vector<int> res(need);
    for (int i = 0; i < need; i++) {
      long long aa = fa[i].x + 0.5;
      long long bb = fb[i].x + 0.5;
      long long cc = fa[i].y + 0.5;
      res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
    }
    return res;
  }

  vector<int> square_mod(vector<int> &a, int m) {
    return multiply_mod(a, a, m, 1);
  }
};
```

5.5 Primality

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.
Time: 15 times the complexity of $a^b \mod c$.

"ModMulLL.h" 18 lines

```
using ull = unsigned long long;

bool IsPrime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  for (int i = 0; i < 15; ++i) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = ModPow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = ModMul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
}
```

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run Init(bits), where bits is the length of the numbers you use.
Time: Expected running time should be good enough for 50-bit numbers.

"MullerRabin.h", "Eratosthenes.h", "Euclid.h" 39 lines

```
using ull = unsigned long long;

vector<ull> pr;

ull f(ull a, ull n, ull &has) {
  return (ModMul(a, a, n) + has) % n;
}

vector<ull> Factorize(ull d) {
  vector<ull> res;
  for (size_t i = 0; i < pr.size() && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      while (d % pr[i] == 0) d /= pr[i];
      res.push_back(pr[i]);
    }
  //d is now a product of at most 2 primes.
  if (d > 1) {
    if (prime(d))
      res.push_back(d);
    else while (true) {
      ull has = rand() % 2321 + 47;
      ull x = 2, y = 2, c = 1;
      for (; c==1; c = gcd((y > x ? y - x : x - y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      }
      if (c != d) {
        res.push_back(c); d /= c;
        if (d != c) res.push_back(d);
        break;
      }
    }
  }
  return res;
}

void Init(int bits) {//how many bits do we use?
  pr = Sieve(1 << ((bits + 2) / 3));
}
```

5.6 Divisibility

euclid.h

Description: Finds the Greatest Common Divisor to the integers a and b. Euclid also finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$.

8 lines

```
using ll = long long;

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

5.6.1 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 totient* or *Euler's phi* function is defined as $\phi(n) :=$ # of positive integers $\leq n$ that are coprime with n. The *cototient* is $n - \phi(n)$. $\phi(1) = 1$, 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 $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$.

11 lines

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

void ComputePhi() {
  for (int i = 0; i < kLim; ++i)
    phi[i] = (i % 2) ? i : i / 2;
  for (int i = 3; i < kLim; i += 2)
    if (phi[i] == i)
      for (int j = i; j < kLim; j += i)
        (phi[j] /= i) *= i - 1;
}
```

5.7 Chinese remainder theorem

CRT.h

Description: wtf

33 lines

```
void extendedGCD(int a, int b, int &x, int &y, int mod) {
  if(b == 0) { x = 1; y = 0; return; }

  int xx, yy;
  extendedGCD(b, a % b, xx, yy, mod);

  x = yy;
  y = (xx - 1LL * a / b * yy) % mod;
  if(y < 0) y += mod;
}

int fastCrt(const vector<pii> &eqs) {
  int currm = 1, currr = 0;

  for(const auto &[m, r] : eqs) {
    if(currm == 1) {
      currm = m;
      currr = r;
      continue;
    }

    int x, y;
    extendedGCD(currm, m, x, y, currm * m);

    int a = (1LL * r * x) % m;
    int b = (1LL * currr * y) % currm;

    currr = (1LL * a * currm + 1LL * b * m) % (currm * m);
    currm *= m;
  }

  return currr;
}
```

5.8 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

5.9 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

5.10 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 $n < 1e19$.

Combinatorial (6)

6.1 The Twelvefold Way

Counts the # of functions $f : N \to K$, $|N| = n$, $|K| = k$. The elements in N and K can be distinguishable or indistinguishable, while f can be injective (one-to-one) of surjective (onto).

N	K	none	injective	surjective
dist	dist	k^n	$\frac{k!}{(k-n)!}$	$k!S(n,k)$
indist	dist	$\binom{n+k-1}{n}$	$\binom{k}{n}$	$\binom{n-1}{n-k}$
dist	indist	$\sum_{t=0}^{k} S(n,t)$	$[n \le k]$	$S(n,k)$
indist	indist	$\sum_{t=1}^{k} p(n,t)$	$[n \le k]$	$p(n,k)$

Here, $S(n,k)$ is the Stirling number of the second kind, and $p(n,k)$ is the partition number.

6.2 Permutations

6.2.1 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

6.2.2 Cycles

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

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

6.2.3 Derangements

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

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

derangements.h

Description: Generates the i:th derangement of S_n (in lexicographical order).

38 lines

```
template <class T, int N>
struct derangements {
  T dgen[N][N], choose[N][N], fac[N];
  derangements() {
    fac[0] = choose[0][0] = 1;
    memset(dgen, 0, sizeof(dgen));
    rep(m,1,N) {
      fac[m] = fac[m-1] * m;
      choose[m][0] = choose[m][m] = 1;
      rep(k,1,m)
        choose[m][k] = choose[m-1][k-1] + choose[m-1][k];
    }
  }
  T DGen(int n, int k) {
    T ans = 0;
    if (dgen[n][k]) return dgen[n][k];
    rep(i,0,k+1)
      ans += (i&1?-1:1) * choose[k][i] * fac[n-i];
    return dgen[n][k] = ans;
  }
  void generate(int n, T idx, int *res) {
    int vals[N];
    rep(i,0,n) vals[i] = i;
    rep(i,0,n) {
      int j, k = 0, m = n - i;
      rep(j,0,m) if (vals[j] > i) ++k;
      rep(j,0,m) {
        T p = 0;
        if (vals[j] > i) p = DGen(m-1, k-1);
        else if (vals[j] < i) p = DGen(m-1, k);
        if (idx <= p) break;
        idx -= p;
      }
      res[i] = vals[j];
      memmove(vals + j, vals + j + 1, sizeof(int)*(m-j-1));
    }
  }
};
```

6.2.4 Involutions

An involution is a permutation with maximum cycle length 2, and it is its own inverse.

$$a(n) = a(n-1) + (n-1)a(n-2)$$

$$a(0) = a(1) = 1$$

1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152

6.2.5 Stirling numbers of the first kind

$$s(n,k) = (-1)^{n-k}c(n,k)$$

$c(n,k)$ is the unsigned Stirling numbers of the first kind, and they count the number of permutations on n items with k cycles.

$$s(n,k) = s(n-1,k-1) - (n-1)s(n-1,k)$$

$$s(0,0) = 1, s(n,0) = s(0,n) = 0$$

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k)$$
$$c(0,0) = 1, c(n,0) = c(0,n) = 0$$

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

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

6.2.7 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

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

6.3 Partitions and subsets

6.3.1 Partition function

Partitions of n with exactly k parts, $p(n,k)$, i.e., writing n as a sum of k positive integers, disregarding the order of the summands.

$$p(n,k) = p(n-1,k-1) + p(n-k,k)$$
$$p(0,0) = p(1,n) = p(n,n) = p(n,n-1) = 1$$

For partitions with any number of parts, $p(n)$ obeys

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

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

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

6.3.2 Binomials

binomial.h

Description: The number of k-element subsets of an n-element set, $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Time: $\mathcal{O}(\min(k, n-k))$ 6 lines

```
ll choose(int n, int k) {
  ll c = 1, to = min(k, n-k);
  if (to < 0) return 0;
  rep(i,0,to) c = c * (n - i) / (i + 1);
  return c;
}
```

binomialModPrime.h

Description: Lucas' thm: 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 $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.

Time: $\mathcal{O}(\log_p n)$ 10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact) {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p * invfact[a - b] % p;
    n /= p; m /= p;
  }
  return c;
}
```

RollingBinomial.h

Description: $\binom{n}{k} \pmod{m}$ in time proportional to the difference between (n, k) and the previous (n, k). 14 lines

```
const ll mod = 1000000007;
vector<ll> invs; // precomputed up to max n, inclusively
struct Bin {
  int N = 0, K = 0;  ll r = 1;
  void m(ll a, ll b) { r = r * a % mod * invs[b] % mod; }
  ll choose(int n, int k) {
    if (k > n || k < 0) return 0;
    while (N < n) ++N, m(N, N-K);
    while (K < k) ++K, m(N-K+1, K);
    while (K > k) m(K, N-K+1), --K;
    while (N > n) m(N-K, N), --N;
    return r;
  }
};
```

multinomial.h

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

Time: $\mathcal{O}((\sum k_i) - k_1)$ 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;
}
```

6.3.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

6.3.4 Bell numbers

Total number of partitions of n distinct elements.

$$B(n) = \sum_{k=1}^{n}\binom{n-1}{k-1}B(n-k) = \sum_{k=1}^{n} S(n,k)$$

$$B(0) = B(1) = 1$$

The first are 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597. For a prime p

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

6.3.5 Triangles

Given rods of length $1, \ldots, n$,

$$T(n) = \frac{1}{24}\begin{cases} n(n-2)(2n-5) & n \text{ even} \\ (n-1)(n-3)(2n-1) & n \text{ odd} \end{cases}$$

is the number of distinct triangles (positive are) that can be constructed, i.e., the # of 3-subsets of $[n]$ s.t. $x \le y \le z$ and $z \ne x + y$.

6.4 General purpose numbers

6.4.1 Catalan numbers

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

$$C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

$$C_0 = 1, C_{n+1} = \sum C_i C_{n-i}$$

First few are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900.

- # of monotonic lattice paths of a $n \times n$-grid which do not pass above the diagonal.
- # of expressions containing n pairs of parenthesis which are correctly matched.
- # of full binary trees with with $n+1$ leaves (0 or 2 children).
- # of non-isomorphic ordered trees with $n+1$ vertices.
- # of ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- # of permutations of $[n]$ with no three-term increasing subsequence.

6.4.2 Super Catalan numbers

The number of monotonic lattice paths of a $n \times n$-grid that do not touch the diagonal.

$$S(n) = \frac{3(2n-3)S(n-1) - (n-3)S(n-2)}{n}$$

$$S(1) = S(2) = 1$$

1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859

6.4.3 Motzkin numbers

Number of ways of drawing any number of nonintersecting chords among n points on a circle. Number of lattice paths from $(0,0)$ to $(n,0)$ never going below the x-axis, using only steps NE, E, SE.

$$M(n) = \frac{3(n-1)M(n-2) + (2n+1)M(n-1)}{n+2}$$

$$M(0) = M(1) = 1$$

1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634

6.4.4 Narayana numbers

Number of lattice paths from $(0,0)$ to $(2n,0)$ never going below the x-axis, using only steps NE and SE, and with k peaks.

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

$$N(n,1) = N(n,n) = 1$$

$$\sum_{k=1}^{n} N(n,k) = C_n$$

1, 1, 1, 1, 3, 1, 1, 6, 6, 1, 1, 10, 20, 10, 1, 1, 15, 50

6.4.5 Schröder numbers

Number of lattice paths from $(0, 0)$ to (n, n) using only steps N,NE,E, never going above the diagonal. Number of lattice paths from $(0, 0)$ to $(2n, 0)$ using only steps NE, SE and double east EE, never going below the x-axis. Twice the Super Catalan number, except for the first term. 1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098

Graph (7)

7.1 Network flow

Dinic.h

Description: wtf

81 lines

```
struct NetworkFlow {
  const int INT_INF = 0x3f3f3f3f;
  const ll LL_INF = 1e18;

  struct Edge {
    int to, flow;
  };

  int n, source, sink;
  vector<int> dst, ptr;
  vector<Edge> edges;
  vector<vector<int>> adj;

  NetworkFlow(int n) : n(n) {
    source = 0;
    sink = n - 1;

    dst.resize(n);
    adj.resize(n);
  }

  void addEdge(int a, int b, int cap) {
    adj[a].push_back(edges.size());
    edges.push_back({b, cap});

    adj[b].push_back(edges.size());
    edges.push_back({a, 0});
  }

  bool bfs(int v) {
    dst.assign(n, INT_INF);

    queue<int> q;
    for(dst[v] = 0, q.push(v); !q.empty(); q.pop()) {
      v = q.front();

      for(auto id : adj[v])
        if(dst[edges[id].to] > 1 + dst[v] && edges[id].flow) {
          dst[edges[id].to] = 1 + dst[v];
          q.push(edges[id].to);
        }
    }

    return dst[sink] != INT_INF;
  }

  ll dfs(int v, ll flow) {
    if(v == sink || !flow) return flow;

    for(; ptr[v] < adj[v].size(); ++ptr[v]) {
      int id = adj[v][ptr[v]];
      int u = edges[id].to;
      if(edges[id].flow && dst[u] == 1 + dst[v]) {
        int pushed = dfs(u, min(flow, (ll) edges[id].flow));
        if(pushed) {
          edges[id].flow -= pushed;
          edges[id ^ 1].flow += pushed;
          return pushed;
        }
      }
    }

    return 0;
  }

  ll dinic() {
    ll flow, total;

    for(total = 0; bfs(source); ) {
      ptr.assign(n, 0);
      while(flow = dfs(source, LL_INF)) total += flow;
    }

    return total;
  }

  void clear() {
    edges.clear();
    for(int i = 0; i < n; ++i) adj[i].clear();
  }
};
```

flowWithLowerBound.cpp

Description: wtf

8 lines

```
Min floww with lower bounds on edges:
Add two new vertices s', t'
Add edge from s' to v with capacity sum{u}(lower_bound(u->v))
Add edge from v to t' with capacity sum{w}(lower_bound(v->w))
Add edge from v to u with capacity cap(v->u) - lower_bound(v->u
     )
Add edge from s to t with capacity INF
After finding a feasible flow, run Dinic again to find a mximum
      feasible flow, ensuring the flow is feasible at every
     step
i.e. do not substract flow from an edge such that the new value
      is less than the lower bound
```

edmonsblossomValeriu.h

Description: None
Usage: ask Djok

<bits/stdc++.h> 94 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 105;

int i, match[N], p[N], base[N], q[N];
bool used[N], viz[N], blossom[N];
vector<int> lda[N];

int lca(int a, int b) {
  memset(viz, 0, sizeof(viz));
  while(1) {
    a = base[a];
    viz[a] = 1;
    if(match[a] == -1) break;
    a = p[match[a]];
  }
  while(1) {
    b = base[b];
    if(viz[b]) break;
    b = p[match[b]];
  }
  return b;
}

void markPath(int x, int y, int children) {
  while(base[x] != y) {
    blossom[base[x]] = blossom[base[match[x]]] = 1;
    p[x] = children;
    children = match[x];
    x = p[match[x]];
  }
}

int findPath(int x) {
  memset(used, 0, sizeof(used));
  memset(p, -1, sizeof(p));
  for(int i = 0; i < N; ++i) base[i] = i;

  int qh = 0, qt = 0;
  q[qt++] = x; used[x] = 1;
  while(qh < qt) {
    int v = q[qh++];
    for(int to : lda[v]) {
      if(base[v] == base[to] || match[v] == to) continue;
      if(to == x || match[to] != -1 && p[match[to]] != -1) {
        int curbase = lca(v, to);
        memset(blossom, 0, sizeof(blossom));
        markPath(v, curbase, to);
        markPath(to, curbase, v);
        for(int i = 0; i < N; ++i)
          if(blossom[base[i]]) {
            base[i] = curbase;
            if(!used[i]) {
              used[i] = 1;
              q[qt++] = i;
            }
          }
      }
      else if(p[to] == -1) {
        p[to] = v;
        if(match[to] == -1) return to;
        to = match[to];
        used[to] = 1;
        q[qt++] = to;
      }
    }
  }
  return -1;
}

int main() {
  // add edge x, y and y, x to lda
  memset(match, -1, sizeof(match));
  for(i = 0; i < N; ++i)
    if(match[i] == -1)
      for(int to : lda[i])
        if(match[to] == -1) {
          match[to] = i;
          match[i] = to;
          break;
        }

  for(i = 0; i < N; ++i)
```

```
    if(match[i] == -1) {
      int v = findPath(i);
      while(v != -1) {
        int pv = p[v], ppv = match[pv];
        match[v] = pv; match[pv] = v; v = ppv;
      }
    }

  return 0;
}
```

minCostMaxFlow.h

Description: wtf

97 lines

```
const int INF = 0x3f3f3f3f;

struct MCMF {
  struct Edge {
    int to, flow, cst;
  };

  int n, source, sink;
  vector<int> d, reald, newd, prv;
  vector<bool> vis;
  vector<Edge> edges;
  vector<vector<int>> adj;

  MCMF(int n) : n(n), source(0), sink(n - 1), d(n), reald(n),
       newd(n), prv(n), vis(n), adj(n) {}

  void addEdge(int a, int b, int cap, int cst) {
    adj[a].push_back(edges.size());
    edges.push_back({b, cap, cst});

    adj[b].push_back(edges.size());
    edges.push_back({a, 0, -cst});
  }

  void bellman() {
    priority_queue<pii> q;
    fill(all(d), INF);

    for(d[source] = 0, q.push({0, source}); !q.empty(); ) {
      int dst = -q.top().fi;
      int v = q.top().se;
      q.pop();

      if(dst != d[v]) continue;

      for(auto id : adj[v]) {
        int u = edges[id].to;
        if(edges[id].flow && d[u] > d[v] + edges[id].cst) {
          d[u] = d[v] + edges[id].cst;
          q.push({-d[u], u});
        }
      }
    }
  }

  bool dijkstra() {
    priority_queue<pii> q;
    fill(all(newd), INF);
    fill(all(vis), false);

    for(reald[source] = newd[source] = 0, q.push({0, source});
         !q.empty(); ) {
      int dst = -q.top().fi;
      int v = q.top().se;
      q.pop();

      if(vis[v]) continue;
      vis[v] = true;

      for(auto id : adj[v]) {
        int u = edges[id].to;
        int w = d[v] + edges[id].cst - d[u];
        if(edges[id].flow && newd[u] > newd[v] + w) {
          newd[u] = newd[v] + w;
          reald[u] = reald[v] + edges[id].cst;
          prv[u] = id;
          q.push({-newd[u], u});
        }
      }
    }

    return newd[sink] < INF;
  }

  pii get() {
    int flow, cst;

    bellman();

    for(flow = cst = 0; dijkstra(); ) {
      int pushed = INF;
      for(int v = sink; v != source; v = edges[prv[v] ^ 1].to)
          {
        pushed = min(pushed, edges[prv[v]].flow);
      }

      flow += pushed;

      for(int v = sink; v != source; v = edges[prv[v] ^ 1].to)
          {
        cst += pushed * edges[prv[v]].cst;
        edges[prv[v]].flow -= pushed;
        edges[prv[v] ^ 1].flow += pushed;
      }

      d = reald;
    }

    return { flow, cst };
  }
};
```

StoerWagnerValeriu.h

Description: None
Usage: ask Djok

<bits/stdc++.h> 84 lines

```
#define sz(x) ((int) (x).size())
#define forn(i,n) for (int i = 0; i < int(n); ++i)
#define forab(i,a,b) for (int i = int(a); i < int(b); ++i)

typedef long long ll;
typedef long double ld;

const int INF = 1000001000;
const ll INFL = 2000000000000001000;

const int maxn = 500;

ll g[maxn][maxn];
ll dist[maxn];
bool used[maxn];

void addEdge(int u, int v, ll c)
{
    g[u][v] += c;
    g[v][u] += c;
}

int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    ll total = 0;
    forn (i, m)
    {
        int k, f;
        scanf("%d%d", &k, &f);
        total += 2 * f;
        vector<int> group;
        forn (j, k)
        {
            int u;
            scanf("%d", &u);
            --u;
            group.push_back(u);
        }
        if (k == 2)
            addEdge(group[0], group[1], 2 * f);
        else
        {
            addEdge(group[0], group[1], f);
            addEdge(group[1], group[2], f);
            addEdge(group[2], group[0], f);
        }
    }
    vector<int> vertices;
    forn (i, n)
        vertices.push_back(i);
    ll mincut = total + 1;
    while (sz(vertices) > 1)
    {
        int u = vertices[0];
        for (auto v: vertices)
            used[v] = false,
            dist[v] = g[u][v];
        used[u] = true;
        forn (ii, sz(vertices) - 2)
        {
            for (auto v: vertices)
                if (!used[v])
                    if (used[u] || dist[v] > dist[u])
                        u = v;
            used[u] = true;
            for (auto v: vertices)
                if (!used[v])
                    dist[v] += g[u][v];
        }
        int t = -1;
        for (auto v: vertices)
            if (!used[v])
                t = v;
        mincut = min(mincut, dist[t]);
        vertices.erase(find(vertices.begin(), vertices.end(), t
             ));
        for (auto v: vertices)
            addEdge(u, v, g[v][t]);
    }

    cout << (total - mincut) / 2 << '\n';
    return 0;
}
```

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.

1 lines

7.2 Matching

matching.cpp

Description: wtf

73 lines

```
struct Matching {
  int m, n;
  vector<int> l, r;
  vector<bool> vis, ok, coverL, coverR;
  vector<vector<int>> adj, adjt;

  Matching(int m, int n) : m(m), n(n), l(n), r(m), vis(m), ok(m
      ), adj(m), adjt(n), coverL(m), coverR(n) {}

  bool pairUp(int v) {
    if(vis[v]) return false;
    vis[v] = true;

    for(auto u : adj[v])
      if(l[u] == -1) return l[u] = v, r[v] = u, true;

    for(auto u : adj[v])
      if(pairUp(l[u])) return l[u] = v, r[v] = u, true;

    return false;
  }

  void bfs(vector<vector<int>> adj, vector<int> l, vector<int>
      r) {
    queue<int> q;
    vis.assign(r.size(), false);
    for(int i = 0; i < r.size(); ++i) if(r[i] == -1) q.push(i),
          vis[i] = true;
    for(; !q.empty(); q.pop()) {
      int v = q.front();
      ok[v] = true;
      for(auto u : adj[v])
        if(!vis[l[u]]) q.push(l[u]), vis[l[u]] = true;
    }
  }

  void cover(int v) {
    for(auto u : adj[v])
      if(!coverR[u]) {
        coverR[u] = true;
        coverL[l[u]] = false;
        cover(l[u]);
      }
  }

  void addEdge(int a, int b) {
    adj[a].push_back(b);
    adjt[b].push_back(a);
  }

  int matching() {
    int sz;
    bool changed;

    l.assign(n, -1);
    r.assign(m, -1);
    for(sz = 0, changed = true; changed; ) {
      vis.assign(m, false);
      changed = false;
      for(int i = 0; i < m; ++i)
        if(r[i] == -1 && pairUp(i)) ++sz, changed = true;
    }
    return sz;
  }

  // if ok[i] == false => i belongs to all maximum matchings
  void computeVerticesBelongingToAllmaximumMatchings() {
    bfs(adj, l, r);
    bfs(adjt, r, l);
  }

  void computeMinimumVertexCover() {
    for(int i = 0; i < m; ++i) if(r[i] != -1) coverL[i] = true;
    for(int i = 0; i < m; ++i) if(r[i] == -1) cover(i);
  }
};
```

WeightedMatching.h

Description: Min cost perfect bipartite matching. Negate costs for max cost.

Time: $\mathcal{O}(N^3)$

57 lines

```
template<typename T>
int MinAssignment(const vector<vector<T>> &c) {
  int n = c.size(), m = c[0].size();       // assert(n <= m);
  vector<T> v(m), dist(m);                 // v: potential
  vector<int> L(n, -1), R(m, -1);          // matching pairs
  vector<int> index(m), prev(m);
  iota(index.begin(), index.end(), 0);

  auto residue = [&](int i, int j) { return c[i][j] - v[j]; };
  for (int f = 0; f < n; ++f) {
    for (int j = 0; j < m; ++j) {
      dist[j] = residue(f, j); prev[j] = f;
    }
    T w; int j, l;
    for (int s = 0, t = 0;;) {
      if (s == t) {
        l = s; w = dist[index[t++]];
        for (int k = t; k < m; ++k) {
          j = index[k]; T h = dist[j];
          if (h <= w) {
            if (h < w) { t = s; w = h; }
            index[k] = index[t]; index[t++] = j;
          }
        }
        for (int k = s; k < t; ++k) {
          j = index[k];
          if (R[j] < 0) goto aug;
        }
      }
      int q = index[s++], i = R[q];
      for (int k = t; k < m; ++k) {
        j = index[k];
        T h = residue(i,j) - residue(i,q) + w;
        if (h < dist[j]) {
          dist[j] = h; prev[j] = i;
          if (h == w) {
            if (R[j] < 0) goto aug;
            index[k] = index[t]; index[t++] = j;
          }
        }
      }
    }
  aug:
    for(int k = 0; k < l; ++k)
      v[index[k]] += dist[index[k]] - w;
    int i;
    do {
      R[j] = i = prev[j];
      swap(j, L[i]);
    } while (i != f);
  }
  T ret = 0;
  for (int i = 0; i < n; ++i) {
    ret += c[i][L[i]]; // (i, L[i]) is a solution
  }
  return ret;
}
```

7.3 DFS algorithms

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected multigraph, 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. HOWEVER, note that we are outputting bridges as BCC's here, because we might be interested in vertex bcc's, not edge bcc's.
To get the articulation points, look for vertices that are in more than 1 BCC.
To get the bridges, look for biconnected components with one edge

Time: $\mathcal{O}(E+V)$

54 lines

```
struct BCC {
  vector<pair<int, int>> edges;
  vector<vector<int>> G;
  vector<int> enter, low, stk;

  BCC(int n) : G(n), enter(n, -1) {}

  int AddEdge(int a, int b) {
    int ret = edges.size();
    edges.emplace_back(a, b);
    G[a].push_back(ret);
    G[b].push_back(ret);
    return ret;
  }

  template<typename Iter>
  void Callback(Iter bg, Iter en) {
    for (Iter it = bg; it != en; ++it) {
      auto edge = edges[*it];
      // Do something useful
    }
  }

  void Solve() {
    for (int i = 0; i < (int)G.size(); ++i)
      if (enter[i] == -1) {
        dfs(i, -1);
      }
  }

  int timer = 0;
  int dfs(int node, int pei) {
    enter[node] = timer++;
    int ret = enter[node];

    for (auto ei : G[node]) if (ei != pei) {
      int vec = (edges[ei].first ^ edges[ei].second ^ node);
      if (enter[vec] != -1) {
        ret = min(ret, enter[vec]);
        if (enter[vec] < enter[node])
          stk.push_back(ei);
      } else {
        int sz = stk.size(), low = dfs(vec, ei);
        ret = min(ret, low);
        stk.push_back(ei);
        if (low >= enter[node]) {
          Callback(stk.begin() + sz, stk.end());
          stk.resize(sz);
        }
      }
    }
    return ret;
  }
};
```

2sat.h

Description: wtf

47 lines

```
struct Sat {
  int n;
  vector<int> ord, val, compId;
  vector<bool> vis;
  vector<vector<int>> adj, adjt;

  Sat(int n) : n(2 * n), vis(2 * n), compId(2 * n), adj(2 * n),
       adjt(2 * n) {}

  void addEdge(int x, int y) {
    x = (x < 0 ? -2 * x - 2 : 2 * x - 1);
    y = (y < 0 ? -2 * y - 2 : 2 * y - 1);
    adj[x].push_back(y);
    adjt[y].push_back(x);
  }

  void addClause(int x, int y) {
    addEdge(-x, y);
    addEdge(-y, x);
  }

  void dfs(int v) {
    vis[v] = true;
    for(auto u : adj[v]) if(!vis[u]) dfs(u);
    ord.push_back(v);
  }

  void dfst(int v, int id) {
    vis[v] = false;
    compId[v] = id;
    if(val[v] == -1) val[v] = 0, val[v ^ 1] = 1;
    for(auto u : adjt[v]) if(vis[u]) dfst(u, id);
  }

  bool solve() {
    val.assign(n, -1);

    for(int i = 0; i < n; ++i) if(!vis[i]) dfs(i);
    for(int nr = 0, i = n - 1; i >= 0; --i) if(vis[ord[i]])
        dfst(ord[i], nr++);

    for(int i = 0; i < n; i += 2) if(compId[i] == compId[i +
        1]) return false;
    return true;
  }

  int get(int i) {
    return val[2 * i - 1];
  }
};
```

7.4 Trees

centroid.cpp

Description: wtf

50 lines

```
int fth[N], sz[N];
bool used[N];
vector<pii> adj[N];

void computeSz(int v, int p = -1) {
  sz[v] = 1;
  for(auto [u, w] : adj[v])
    if(u != p && !used[u]) {
      computeSz(u, v);
      fth[u] = v;
      sz[v] += sz[u];
    }
}

int findCentroid(int v, int n, int p = -1) {
  while(true) {
    int heavyCh = -1;
    for(auto [u, w] : adj[v])
      if(u != p && !used[u] && (heavyCh == -1 || sz[u] > sz[
          heavyCh])) heavyCh = u;

    if(heavyCh == -1 || sz[heavyCh] <= n / 2) return v;
    p = v;
    v = heavyCh;
  }

  return -1;
}

void dfs(int v, int p = -1) {
  // do something with node v

  for(auto [u, w] : adj[v])
    if(u != p && !used[u])
      dfs(u, v);
}

void solve(int v, int n) {
  fth[v] = 0;
  computeSz(v);
  int centroid = findCentroid(v, n);

  dfs(centroid);

  used[centroid] = true;

  for(auto [u, w] : adj[centroid])
    if(!used[u]) solve(u, sz[u]);

  if(fth[centroid] && !used[fth[centroid]]) solve(fth[centroid
      ], n - sz[centroid]);
}
```

CompressTree.h

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S| - 1$) pairwise LCA's and compressing edges. Returns the nodes of the reduced tree, while at the same time populating a link array that stores the new parents. The root points to -1.
Time: $\mathcal{O}(|S| * (\log |S| + LCA\_Q))$

"LCA.h"   18 lines

```
vector<int> CompressTree(vector<int> v, LCA& lca,
                          vector<int>& link) {
  auto cmp = [&](int a, int b) {
    return lca.enter[a] < lca.enter[b];
  };
  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());

  for (int i = (int)v.size() - 1; i > 0; --i)
    v.push_back(lca.Query(v[i - 1], v[i]));

  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());

  for (int i = 0; i < (int)v.size(); ++i)
    link[v[i]] = (i == 0 ? -1 : lca.Query(v[i - 1], v[i]));
  return v;
}
```

HLD.h

Description: wtf

73 lines

```
struct HLD {
  int n, t;
  vector<int> in, out, head, fth, h, sz;
  vector<vector<int>> adj;
  SegTree segTree;
  // in[i] = time entering node i
  // out[i] = time leaving node i
  // head[i] = head of path containing node i
  // fth[i] = parent of node i in original tree
  // h[i] = height of node i in original tree starting from 0
  // sz[i] = size of subtree of i in original tree

  HLD(int n) : n(n), in(n), out(n), head(n), fth(n), h(n), sz(n
      ), adj(n), segTree(n) {}

  void addEdge(int a, int b) {
    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  void dfsSize(int v, int p = -1) {
    sz[v] = 1;
    for(auto &u : adj[v])
      if(u != p) {
        fth[u] = v;
        h[u] = 1 + h[v];
        dfsSize(u, v);
        sz[v] += sz[u];
        if(sz[u] > sz[adj[v][0]]) swap(u, adj[v][0]);
      }
  }

  void dfsHld(int v, int p = -1) {
    in[v] = t++;
    for(auto u : adj[v])
      if(u != p) {
        head[u] = (u == adj[v][0] ? head[v] : u);
        dfsHld(u, v);
      }
    out[v] = t;
  }

  void build(const vector<int> &v) {
    t = 0;
    sz.assign(n, 0);

    dfsSize(0);
    dfsHld(0);

    for(int i = 0; i < n; ++i) segTree.update(in[i], v[i]);
  }

  void update(int v, int val) {
    segTree.update(in[v], val);
  }

  int query(int v, int u) {
    int res = 0;
    while(head[v] != head[u]) {
      if(h[head[v]] > h[head[u]]) swap(v, u);

      res = max(res, segTree.query(in[head[u]], in[u] + 1));
      u = fth[head[u]];
    }

    if(h[v] > h[u]) swap(v, u);
    res = max(res, segTree.query(in[v], in[u] + 1));

    return res;
  }

  // subtree of v: [in_v, out_v)
  // path from v to the heavy path head: [in_head_v, in_v]
};
```

7.5 Matrix tree theorem

MatrixTree.h

Description: To count the number of spanning trees in an undirected graph G: create an $N \times N$ matrix mat, and for each edge $(a, b) \in G$, do mat[a][a]++, mat[b][b]++, mat[a][b]--, mat[b][a]--. Remove the last row and column, and take the determinant.

1 lines

Strings (8)

ZFunction.h

Description: wtf

14 lines

```
int z[N];

void computeZFunction(const string &s) {
  int n = s.length();
  for(int i = 0; i < n; ++i) {
    int l = -1, r = -1;
    for(int j = i + 1; j < n; ++j) {
      int k = (j > r ? 0 : min(z[j - l + i], r - j + 1));
      while(j + k < n && s[i + k] == s[j + k]) ++k;
      z[j] = k;
      if(j + k - 1 > r) l = j, r = j + k - 1;
    }
  }
}
```

Manacher.h

Description: wtf

25 lines

```
int odd[N], even[N];
/*
 * [i - odd[i], i + odd[i]] - longest palindrome with center in
      i
 * [i - even[i], i + even[i] - 1] - longest palindrome with
      center in (i-1, i)
 */

void manacher(const string &s) {
  int l = 0, r = -1, n = s.length();
  for(int i = 0; i < n; ++i) {
    odd[i] = (i > r ? 1 : min(odd[l + r - i], r - i));
    while(i - odd[i] >= 0 && i + odd[i] < n && s[i - odd[i]] ==
         s[i + odd[i]]) ++odd[i];
    --odd[i];

    if(i + odd[i] > r) l = i - odd[i], r = i + odd[i];
  }

  l = 0; r = -1;
  for(int i = 0; i < n; ++i) {
    even[i] = (i > r ? 1 : min(even[l + r - i + 1], r - i));
    while(i - even[i] >= 0 && i + even[i] - 1 < n && s[i - even
         [i]] == s[i + even[i] - 1]) ++even[i];
    --even[i];

    if(i + even[i] - 1 > r) l = i - even[i], r = i + even[i] -
         1;
  }
}
```

PalindromicTree.h

Description: A trie-like structure for keeping track of palindromes of a string s. It has two roots, 0 (for even palindromes) and 1 (for odd palindromes). Each node stores the length of the palindrome, the count and a link to the longest "aligned" subpalindrome. Can be made online from left to right
Time: $\mathcal{O}(N)$

53 lines

```
struct PalTree {
  struct Node {
    map<char, int> leg;
    int link, len, cnt;
  };
  vector<Node> T;
  int nodes = 2;

  PalTree(string str) : T(str.size() + 2) {
    T[1].link = T[1].len = 0;
    T[0].link = T[0].len = -1;

    int last = 0;
    for (int i = 0; i < (int)str.size(); ++i) {
      char now = str[i];

      int node = last;
      while (now != str[i - T[node].len - 1])
        node = T[node].link;

      if (T[node].leg.count(now)) {
        node = T[node].leg[now];
        T[node].cnt += 1;
        last = node;
        continue;
      }

      int cur = nodes++;
      T[cur].len = T[node].len + 2;
      T[node].leg[now] = cur;

      int link = T[node].link;
      while (link != -1) {
        if (now == str[i - T[link].len - 1] &&
            T[link].leg.count(now)) {
          link = T[link].leg[now];
          break;
        }
        link = T[link].link;
      }
      if (link <= 0) link = 1;

      T[cur].link = link;
      T[cur].cnt = 1;

      last = cur;
    }

    for (int node = nodes - 1; node > 0; --node) {
      T[T[node].link].cnt += T[node].cnt;
    }
  }
};
```

MinRotation.h

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

11 lines

```
int MinRotation(string s) {
  int a = 0, n = s.size(); s += s;
  for (int b = 0; b < n; ++b)
    for (int i = 0; i < n; ++i) {
      if (a + i == b || s[a + i] < s[b + i]) {
        b += max(0, i - 1); break;
      }
      if (s[a + i] > s[b + i]) { a = b; break; }
    }
  return a;
}
```

SuffixAutomaton.h

Description: wtf

26 lines

```
const int N = 2000010;
const int A = 26;

int k, last;
int len[N], link[N];
int go[N][A];

void addLetter(int ch) {
  int p = last;
  last = k++;
  len[last] = len[p] + 1;

  while(!go[p][ch]) go[p][ch] = last, p = link[p];
  if(go[p][ch] == last) return void(link[last] = 0);

  int q = go[p][ch];
  if(len[q] == len[p] + 1) return void(link[last] = q);

  int cl = k++;
  memcpy(go[cl], go[q], sizeof go[q]);
  link[cl] = link[q];
  len[cl] = len[p] + 1;
  link[last] = link[q] = cl;

  while(go[p][ch] == q) go[p][ch] = cl, p = link[p];
}
```

SuffixArray.h

Description: wtf

29 lines

```
const int L = 200010;
const int LOGL = 18;

int sa[L];
int p[LOGL][L];

void buildSA(const string &s) {
  int n = s.length();
  for(int j = 0; j < n; ++j) p[0][j] = s[j];
  for(int i = 0; i + 1 < LOGL; ++i) {
    vector<pair<pii, int>> v;
    for(int j = 0; j < n; ++j)
      v.push_back({{p[i][j], j + (1 << i) < n ? p[i][j + (1 <<
           i)] : -1}, j});
    sort(all(v));

    for(int j = 0; j < n; ++j)
      p[i + 1][v[j].se] = (j && v[j - 1].fi == v[j].fi ? p[i +
           1][v[j - 1].se] : j);
  }

  for(int j = 0; j < n; ++j) sa[p[LOGL - 1][j]] = j;

  for(int i = 0, k = 0; i < n; ++i)
    if(p[LOGL - 1][i] != n - 1) {
      for(int j = sa[p[LOGL - 1][i] + 1]; i + k < n && j + k <
           n && s[i + k] == s[j + k]; ) ++k;
      //lcp[p[LOGL - 1][i]] = k;
      if(k) --k;
    }
}
```

aho.cpp
Description: wtf
41 lines

```
const int SIGMA = 26;

struct Node {
  int cnt;
  Node *suff, *next[SIGMA];

  Node(int cnt = 0) : cnt(cnt) {
    suff = nullptr;
    for(int i = 0; i < SIGMA; ++i) next[i] = nullptr;
  }
};
using PNode = Node*;

void insert(PNode root, const string &s) {
  for(auto ch : s) {
    if(!root->next[ch - 'a']) root->next[ch - 'a'] = new Node;
    root = root->next[ch - 'a'];
  }
  ++root->cnt;
}

void buildSuffLinks(PNode root) {
  queue<PNode> q;

  root->suff = root;
  for(int i = 0; i < SIGMA; ++i) {
    if(!root->next[i]) root->next[i] = root;
    else q.push(root->next[i]);

    root->next[i]->suff = root;
  }

  for(; !q.empty(); q.pop()) {
    PNode node = q.front();
    node->cnt += node->suff->cnt;

    for(int i = 0; i < SIGMA; ++i)
      if(node->next[i]) node->next[i]->suff = node->suff->next[
           i], q.push(node->next[i]);
      else node->next[i] = node->suff->next[i];
  }
}
```

Various (9)

9.1 Misc. algorithms

AlphaBeta.h
Description: Uses the alpha-beta pruning method to find score values for states in games (minimax)
8 lines

```
int AlphaBeta(state s, int alpha, int beta) {
  if (s.finished()) return s.score();
  for (state t : s.next()) {
    alpha = max(alpha, -AlphaBeta(t, -beta, -alpha));
    if (alpha >= beta) break;
  }
  return alpha;
}
```

9.2 Dynamic programming

DivideAndConquerDP.h
Description: Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}((N + (hi - lo)) \log N)$
18 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); }
};
```

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 $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
$dp[i] = minj < idp[j] + b[j] * a[i] whenb[j] >= b[j+1], a[i] <= a[i+1] : O(n^2) - > O(n)$
$dp[i][j] = mink < jdp[i-1][k] + b[k] * a[j] whenb[k] >= b[k+1], a[j] <= a[j+1] : O(kn^2) - > O(kn)$
$dp[i][j] = mink < jdp[i-1][k] + C[k][j], optim[i][j] <= optim[i][j+1] : O(kn^2) - > O(knlogn)$

Time: $\mathcal{O}(N^2)$
1 lines

9.3 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` violations generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (`1`), 0-divs (`4`), infinities (`8`) and denormals (`16`).

9.4 Optimization tricks

9.4.1 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.

9.4.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize for loops and optimizes floating points better (assumes associativity and turns off denormals).
- `#pragma GCC target ("avx,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).

rand.cpp
Description: wtf
6 lines

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count()); // for int - returns in [0, 2^32)
//mt19937_64 rng(chrono::steady_clock::now().time_since_epoch()
     .count()); // for long long - returns in [0, 2^64)
//uniform_int_distribution<> uniform(A, B);
//uniform_real_distribution<> uniform(A, B);
// usage: rng()
// usage: uniform(rng)
```

Unrolling.h
5 lines

```
#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment, if needed
while (i + 4 <= to) { F F F F }
while (i < to) F
```

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 contigous subvector sum
  Invariants
  Huffman encoding
Graph teory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algoritm
  MST: Prim's algoritm
  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
  Eulercykler
  Flow networks
  * Augumenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cutvertices, cutedges och 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 programmering
  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
  Euklidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's small 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 polynom 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
```